Data-Driven Science and Engineering

Machine Learning, Dynamical Systems, and Control

Steven L. Brunton
Department of Mechanical Engineering
University of Washington

J. Nathan Kutz
Department of Applied Mathematics
University of Washington

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Contents

Preface **ix**

Acknowledgments **xiv**

Optimization, Equations, Symbols, and Acronyms **xvii**

I Dimensionality Reduction and Transforms 1

1 Singular Value Decomposition (SVD) 3
1.1 Overview . . . 4
1.2 Matrix Approximation . . . 8
1.3 Mathematical Properties and Manipulations . . . 14
1.4 Pseudo-Inverse, Least-Squares, and Regression . . . 19
1.5 Principal Component Analysis (PCA) . . . 27
1.6 Eigenfaces Example . . . 34
1.7 Truncation and Alignment . . . 41
1.8 Randomized Singular Value Decomposition . . . 48
1.9 Tensor Decompositions and N-Way Data Arrays . . . 55

2 Fourier and Wavelet Transforms 64
2.1 Fourier Series and Fourier Transforms . . . 65
2.2 Discrete Fourier Transform (DFT) and Fast Fourier Transform (FFT) . . . 76
2.3 Transforming Partial Differential Equations . . . 85
2.4 Gabor Transform and the Spectrogram . . . 91
2.5 Laplace Transform . . . 98
2.6 Wavelets and Multi-Resolution Analysis . . . 102
2.7 Two-Dimensional Transforms and Image Processing . . . 105

3 Sparsity and Compressed Sensing 118
3.1 Sparsity and Compression . . . 119
3.2 Compressed Sensing . . . 123
3.3 Compressed Sensing Examples . . . 128

3.4 The Geometry of Compression . . . 132
3.5 Sparse Regression . . . 137
3.6 Sparse Representation . . . 142
3.7 Robust Principal Component Analysis (RPCA) . . . 145
3.8 Sparse Sensor Placement . . . 148

II Machine Learning and Data Analysis 158

4 Regression and Model Selection 160
4.1 Classic Curve Fitting . . . 162
4.2 Nonlinear Regression and Gradient Descent . . . 168
4.3 Regression and $\mathbf{Ax} = \mathbf{b}$: Over- and Under-Determined Systems . 174
4.4 Optimization as the Cornerstone of Regression . . . 181
4.5 The Pareto Front and *Lex Parsimoniae* . . . 186
4.6 Model Selection: Cross-Validation . . . 190
4.7 Model Selection: Information Criteria . . . 195

5 Clustering and Classification 203
5.1 Feature Selection and Data Mining . . . 204
5.2 Supervised versus Unsupervised Learning . . . 210
5.3 Unsupervised Learning: k-Means Clustering . . . 215
5.4 Unsupervised Hierarchical Clustering: Dendrogram . . . 219
5.5 Mixture Models and the Expectation-Maximization Algorithm . . 223
5.6 Supervised Learning and Linear Discriminants . . . 228
5.7 Support Vector Machines (SVM) . . . 232
5.8 Classification Trees and Random Forest . . . 239
5.9 Top 10 Algorithms of Data Mining circa 2008 (Before the Deep Learning Revolution) . . . 244

6 Neural Networks and Deep Learning 252
6.1 Neural Networks: Single-Layer Networks . . . 253
6.2 Multi-Layer Networks and Activation Functions . . . 257
6.3 The Backpropagation Algorithm . . . 263
6.4 The Stochastic Gradient Descent Algorithm . . . 268
6.5 Deep Convolutional Neural Networks . . . 271
6.6 Neural Networks for Dynamical Systems . . . 275
6.7 Recurrent Neural Networks . . . 281
6.8 Autoencoders . . . 285
6.9 Generative Adversarial Networks (GANs) . . . 289
6.10 The Diversity of Neural Networks . . . 291

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

III Dynamics and Control 303

7 Data-Driven Dynamical Systems 305
7.1 Overview, Motivations, and Challenges . . . 306
7.2 Dynamic Mode Decomposition (DMD) . . . 313
7.3 Sparse Identification of Nonlinear Dynamics (SINDy) . . . 331
7.4 Koopman Operator Theory . . . 344
7.5 Data-Driven Koopman Analysis . . . 357

8 Linear Control Theory 376
8.1 Closed-Loop Feedback Control . . . 378
8.2 Linear Time-Invariant Systems . . . 383
8.3 Controllability and Observability . . . 389
8.4 Optimal Full-State Control: Linear–Quadratic Regulator (LQR) . . 396
8.5 Optimal Full-State Estimation: the Kalman Filter . . . 401
8.6 Optimal Sensor-Based Control: Linear–Quadratic Gaussian (LQG) 404
8.7 Case Study: Inverted Pendulum on a Cart . . . 406
8.8 Robust Control and Frequency-Domain Techniques . . . 418

9 Balanced Models for Control 435
9.1 Model Reduction and System Identification . . . 435
9.2 Balanced Model Reduction . . . 436
9.3 System Identification . . . 451

IV Advanced Data-Driven Modeling and Control 467

10 Data-Driven Control 469
10.1 Model Predictive Control (MPC) . . . 470
10.2 Nonlinear System Identification for Control . . . 473
10.3 Machine Learning Control . . . 479
10.4 Adaptive Extremum-Seeking Control . . . 491

11 Reinforcement Learning 504
11.1 Overview and Mathematical Formulation . . . 505
11.2 Model-Based Optimization and Control . . . 513
11.3 Model-Free Reinforcement Learning and Q-Learning . . . 516
11.4 Deep Reinforcement Learning . . . 524
11.5 Applications and Environments . . . 529
11.6 Optimal Nonlinear Control . . . 534

12 Reduced-Order Models (ROMs) 541

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

12.1 Proper Orthogonal Decomposition (POD) for Partial Differential Equations . . . 542
12.2 Optimal Basis Elements: the POD Expansion . . . 548
12.3 POD and Soliton Dynamics . . . 555
12.4 Continuous Formulation of POD . . . 560
12.5 POD with Symmetries: Rotations and Translations . . . 566
12.6 Neural Networks for Time-Stepping with POD . . . 571
12.7 Leveraging DMD and SINDy for POD-Galerkin . . . 577

13 Interpolation for Parametric Reduced-Order Models 585
13.1 Gappy POD . . . 585
13.2 Error and Convergence of Gappy POD . . . 590
13.3 Gappy Measurements: Minimize Condition Number . . . 594
13.4 Gappy Measurements: Maximal Variance . . . 600
13.5 POD and the Discrete Empirical Interpolation Method (DEIM) . . . 603
13.6 DEIM Algorithm Implementation . . . 608
13.7 Decoder Networks for Interpolation . . . 613
13.8 Randomization and Compression for ROMs . . . 617
13.9 Machine Learning ROMs . . . 619

14 Physics-Informed Machine Learning 628
14.1 Mathematical Foundations . . . 629
14.2 SINDy Autoencoder: Coordinates and Dynamics . . . 632
14.3 Koopman Forecasting . . . 636
14.4 Learning Nonlinear Operators . . . 640
14.5 Physics-Informed Neural Networks (PINNs) . . . 644
14.6 Learning Coarse-Graining for PDEs . . . 648
14.7 Deep Learning and Boundary Value Problems . . . 653

Glossary 657

References 669

Index 731

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Preface

This book is about the growing intersection of data-driven methods, machine learning, applied optimization, and the classical fields of engineering mathematics and mathematical physics. We developed this material over a number of years, primarily to educate our advanced undergraduate and beginning graduate students from engineering and physical science departments. Typically, such students have backgrounds in linear algebra, differential equations, and scientific computing, with engineers often having some exposure to control theory and/or partial differential equations. However, most undergraduate curricula in engineering and science fields have little or no exposure to data methods and/or optimization. Likewise, computer scientists and statisticians have little exposure to dynamical systems and control. Our goal is to provide a broad entry point to applied machine learning for both of these groups of students. We have chosen the methods discussed in this book for their (1) relevance, (2) simplicity, and (3) generality, and we have attempted to present a range of topics, from basic introductory material up to research-level techniques.

Data-driven discovery is currently revolutionizing how we model, predict, and control complex systems. The most pressing scientific and engineering problems of the modern era are not amenable to empirical models or derivations based on first principles. Increasingly, researchers are turning to data-driven approaches for a diverse range of complex systems, such as turbulence, the brain, climate, epidemiology, finance, robotics, and autonomy. These systems are typically nonlinear, dynamic, multi-scale in space and time, and high-dimensional, with dominant underlying patterns that should be characterized and modeled for the eventual goal of sensing, prediction, estimation, and control. With modern mathematical methods, enabled by the unprecedented availability of data and computational resources, we are now able to tackle previously unattainable problems. A small handful of these new techniques include robust image reconstruction from sparse and noisy random pixel measurements, turbulence control with machine learning, optimal sensor and actuator placement, discovering interpretable nonlinear dynamical systems purely from data, and reduced-order models to accelerate the optimization and control of systems with complex multi-scale physics.

Driving modern data science is the availability of vast and increasing quantities of data, enabled by remarkable innovations in low-cost sensors, orders-

of-magnitude increases in computational power, and virtually unlimited data storage and transfer capabilities. Such vast quantities of data are affording engineers and scientists across all disciplines new opportunities for data-driven discovery, which has been referred to as the fourth paradigm of scientific discovery [326]. This fourth paradigm is the natural culmination of the first three paradigms: empirical experimentation, analytical derivation, and computational investigation. The integration of these techniques provides a transformative framework for data-driven discovery efforts. This process of scientific discovery is not new, and indeed mimics the efforts of leading figures of the scientific revolution: Johannes Kepler (1571–1630) and Sir Isaac Newton (1642–1727). Each played a critical role in developing the theoretical underpinnings of celestial mechanics, based on a combination of empirical data-driven and analytical approaches. Data science is not replacing mathematical physics and engineering, but is instead augmenting it for the twenty-first century, resulting in more of a renaissance than a revolution.

Data science itself is not new, having been proposed more than 50 years ago by John Tukey, who envisioned the existence of a scientific effort focused on learning from data, or *data analysis* [205]. Since that time, data science has been largely dominated by two distinct cultural outlooks on data [109]. The *machine learning* community, which predominantly comprises computer scientists, is typically centered on prediction quality and scalable, fast algorithms. Although not necessarily in contrast, the *statistical learning* community, often centered in statistics departments, focuses on the inference of interpretable models. Both methodologies have achieved significant success and have provided the mathematical and computational foundations for data science methods. For engineers and scientists, the goal is to leverage these broad techniques to infer and compute models (typically nonlinear) from observations that correctly identify the underlying dynamics *and* generalize qualitatively and quantitatively to unmeasured parts of phase, parameter, or application space. Our goal in this book is to leverage the power of both statistical and machine learning to solve engineering problems.

Themes of This Book

There are a number of key themes that have emerged throughout this book. First, many complex systems exhibit *dominant low-dimensional patterns* in the data, despite the rapidly increasing resolution of measurements and computations. This underlying structure enables efficient sensing, and compact representations for modeling and control. Pattern extraction is related to the second theme of finding *coordinate transforms* that simplify the system. Indeed, the rich history of mathematical physics is centered around coordinate transforma-

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

tions (e.g., spectral decompositions, the Fourier transform, generalized functions, etc.), although these techniques have largely been limited to simple idealized geometries and linear dynamics. The ability to derive *data-driven* transformations opens up opportunities to generalize these techniques to new research problems with more complex geometries and boundary conditions. We also take the perspective of *dynamical systems and control* throughout the book, applying data-driven techniques to model and control systems that evolve in time. Perhaps the most pervasive theme is that of *data-driven applied optimization*, as nearly every topic discussed is related to optimization (e.g., finding *optimal* low-dimensional patterns, *optimal* sensor placement, machine learning *optimization*, *optimal* control, etc.). Even more fundamentally, most data is organized into arrays for analysis, where the extensive development of numerical linear algebra tools from the early 1960s onward provides many of the foundational mathematical underpinnings for matrix decompositions and solution strategies used throughout this text.

Overview of Second Edition

The integration of machine learning methods in science and engineering has advanced significantly in the two years since publication of the first edition. The field is fast-moving, with innovations coming in a diversity of application areas that use creative mathematical architectures for advancing the state of the art in data-driven modeling and control. This second edition is aimed at capturing some of the more salient and successful advancements in the field. It helps bring the reader to a modern understanding of what is possible using machine learning in science and engineering. As with the first edition, extensive online supplementary material can be found at the book's website:

`http://databookuw.com`

Major changes in the second edition include the following.

- **Homework:** Extensive homework has been added to every chapter, with additional homework and projects on the book's website. Homework ranges in difficulty from introductory demonstrations and concept-building to advanced problems that reproduce modern research papers and may be the basis of course projects.

- **Code:** Python code has been added throughout, in parallel to existing MATLAB code, and both sets of codes have been streamlined considerably. All extended codes are available in MATLAB and Python on the book's website and GitHub pages.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

  - Python Code:
    `https://github.com/dynamicslab/databook_python`
  - MATLAB Code:
    `https://github.com/dynamicslab/databook_matlab`

  Wherever possible, a minimal representation of code has been presented in the text to improve readability. These code blocks are equivalently expressed in MATLAB and Python. In more advanced examples, it is often advantageous to use either MATLAB or Python, but not both. In such cases, this has been indicated and only a single code block is demonstrated. The full code is available at the above GitHub sites as well as on the book's website. In addition, extensive codes are available in R online. We encourage the reader to read the book and follow along with code to help improve the learning process and experience.

- **New chapters:** Two new chapters have been added on "Reinforcement Learning" and "Physics-Informed Machine Learning," which are two of the most exciting and rapidly growing fields of research in machine learning, modeling, and control.

  - Reinforcement Learning: Reinforcement learning is a third major branch of machine learning that is concerned with how to learn control laws and policies to interact with a complex environment. This is a critical area of research, situated at the growing intersection of control theory and machine learning.
  - Physics-Informed Machine Learning: The integration of physics concepts, constraints, and symmetries is providing exceptional opportunities for training machine learning algorithms that are encoded with knowledge of physics. This chapter features a number of recent innovations aimed at understanding how this can be done in principle and in practice.

- **New sections:** We have added and improved material throughout, including the following.

  - Chapter 1: new sections discussing condition number, connections to the eigendecomposition, and error bounds for SVD (singular value decomposition) based approximations.
  - Chapter 2: new section on the Laplace transform.
  - Chapter 6: new sections devoted to autoencoders, recurrent neural networks and generative adversarial networks.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

  - Chapter 7: addition of recent innovations to DMD (dynamic mode decomposition), Koopman theory, and SINDy (sparse identification of nonlinear dynamics).
  - Chapter 10: new section on model predictive control.
  - Chapter 12 (previously Chapter 11): new sections on using neural networks for time-stepping in reduced-order models, as well as non-intrusive methods such as DMD.
  - Chapter 13 (previously Chapter 12): new sections on decoder networks for interpolation in model reduction as well as randomized linear algebra methods for scalable reduced-order models.

- **Videos:** An extensive collection of video lectures are available on YouTube, covering nearly every topic from each section of the book. Videos may be found on our YouTube channels.
  - `www.youtube.com/c/eigensteve`
  - `www.youtube.com/c/NathanKutzAMATH`
  - `www.youtube.com/c/PhysicsInformedMachineLearning`

- **Typos:** We have corrected typos and mistakes throughout the second edition.

Online Material

We have designed this book to make extensive use of online supplementary material, including codes, data, videos, homework, and suggested course syllabi. All of this material can be found at the book's website: `http://databookuw.com`.

In addition to course resources, all of the code and data used in the book are available on the book's GitHub: `https://github.com/dynamicslab/`. The codes online are more extensive than those presented in the book, including code used to generate publication-quality figures. In addition to the Python and MATLAB used throughout the text, online code is also available in R. Data visualization was ranked as the top-used data science method in the Kaggle 2017 *The State of Data Science and Machine Learning* study, and so we highly encourage readers to download the online codes and make full use of these plotting commands.

We have also recorded and posted video lectures on YouTube for every section in this book, available at `www.youtube.com/c/eigensteve` and `www.youtube.com/c/NathanKutzAMATH`. We include supplementary videos for

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

students to fill in gaps in their background on scientific computing and foundational applied mathematics. We have designed this text to be both a reference as well as the material for several courses at various levels of student preparation. Most chapters are also modular, and may be converted into stand-alone *boot camps*, containing roughly 10 hours of materials each.

How to Use This Book

Our intended audience includes beginning graduate students, or advanced undergraduates, in engineering and science. As such, the machine learning methods are introduced at a beginning level, whereas we assume students know how to model physical systems with differential equations and simulate them with solvers such as **ode45**. The diversity of topics covered thus range from introductory to state-of-the-art research methods. Our aim is to provide an integrated viewpoint and mathematical toolset for solving engineering and science problems. Alternatively, the book can also be useful for computer science and statistics students, who often have limited knowledge of dynamical systems and control. Various courses can be designed from this material, and several example syllabi may be found on the book's website – this includes homework, data sets, and code.

First and foremost, we want this book to be fun, inspiring, eye-opening, and empowering for young scientists and engineers. We have attempted to make everything as simple as possible, while still providing the depth and breadth required to be useful in research. Many of the chapter topics in this text could be entire books in their own right, and many of them are. However, we also wanted to be as comprehensive as may be reasonably expected for a field that is so big and moving so fast. We hope that you enjoy this book, master these methods, and change the world with applied data science!

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Acknowledgments

We are indebted to many wonderful students, collaborators, and colleagues for valuable feedback, suggestions, and support. We are especially grateful to Joshua Proctor, who was instrumental in the origination of this book and who helped guide much of the framing and organization. We have also benefited from extensive interactions and conversations with Bing Brunton, Jean-Christophe Loiseau, Bernd Noack, and Sam Taira. This work would also not have been possible without our many great colleagues and collaborators, with whom we have worked and whose research is featured throughout this book.

Throughout the writing of the first edition and teaching of related courses, we have received great feedback and comments from our excellent students and postdocs: Travis Askham, Michael Au-Yeung, Zhe Bai, Ido Bright, Kathleen Champion, Emily Clark, Charles Delahunt, Daniel Dylewski, Ben Erichson, Charlie Fiesler, Xing Fu, Chen Gong, Taren Gorman, Jacob Grosek, Seth Hirsh, Mikala Johnson, Eurika Kaiser, Mason Kamb, James Kunert, Bethany Lusch, Pedro Maia, Niall Mangan, Krithika Manohar, Ariana Mendible, Thomas Mohren, Megan Morrison, Markus Quade, Sam Rudy, Susanna Sargsyan, Isabel Scherl, Eli Shlizerman, George Stepaniants, Ben Strom, Chang Sun, Roy Taylor, Meghana Velagar, Jake Weholt, and Matt Williams. Our students are our inspiration for this book, and they make it fun and exciting to come to work every day.

For this second edition, special thanks are in order to Daniel Dylewsky for his incredible efforts generating extensive Python code for the book. We are indebted to his efforts in helping generate the bulk of this code. We also thank Richard Knight for generously sharing his R code online. We have also benefited from extensive discussions with Bing Brunton. We are also grateful to Scott Dawson for incredibly helpful, detailed, and insightful comments and errata throughout the entire book. We would also like to thank the following for contributing corrections, typos, errata, and suggestions throughout, and also to all of those we have missed: Asude Aydin, Dammalapati Harshavardhan, Zdenek Hurak, Jamie Johnson, Lance Larsen, Pietro Monticone, Matt Reeves, Joel Rosenfeld, Simon Schauppenlehner, Csaba Szepesvari, Yuchan Tseng, Balint Uveges, and Ning Zheng.

We are grateful for the continued support and encouragement of our publishers Katie Leach and Lauren Cowles at Cambridge University Press. We are especially indebted to Katie for her patient handling of this second edition, and

to Charles Howell and Geoff Amor for greatly improving the book through production and copyediting.

Steven L. Brunton and J. Nathan Kutz
Seattle, WA, March 2018
Second Edition, September 2021

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Common Optimization Techniques, Equations, Symbols, and Acronyms

Most Common Optimization Strategies

Least-squares (discussed in Chapters 1 and 4) minimizes the sum of the squares of the residuals between a given fitting model and data. Linear least-squares, where the residuals are linear in the unknowns, has a closed-form solution which can be computed by taking the derivative of the residual with respect to each unknown and setting it to zero. It is commonly used in the engineering and applied sciences for fitting polynomial functions. Nonlinear least-squares typically requires iterative refinement based upon approximating the nonlinear least-squares with a linear least-squares at each iteration.

Gradient descent (discussed in Chapters 4 and 6) is the industry-leading, convex optimization method for high-dimensional systems. It minimizes residuals by computing the gradient of a given fitting function. The iterative procedure updates the solution by *moving downhill* in the residual space. The Newton–Raphson method is a one-dimensional version of gradient descent. Since it is often applied in high-dimensional settings, it is prone to find only local minima. Critical innovations for big data applications include stochastic gradient descent and the backpropagation algorithm, which makes the optimization amenable to computing the gradient itself.

Alternating descent method (ADM) (discussed in Chapter 4) avoids computations of the gradient by optimizing in one unknown at a time. Thus all unknowns are held constant while a line search (non-convex optimization) can be performed in a single variable. This variable is then updated and held constant while another of the unknowns is updated. The iterative procedure continues through all unknowns and the iteration procedure is repeated until a desired level of accuracy is achieved.

Augmented Lagrange method (ALM) (discussed in Chapters 3 and 8) is a class of algorithms for solving constrained optimization problems. They are similar to penalty methods in that they replace a constrained optimization problem by a series of unconstrained problems and add a penalty term to the objective

which helps enforce the desired constraint. ALM adds another term designed to mimic a Lagrange multiplier. The augmented Lagrangian is not the same as the method of Lagrange multipliers.

Linear program and simplex method are the workhorse algorithms for convex optimization. A linear program has an objective function which is linear in the unknown, and the constraints consist of linear inequalities and equalities. By computing its feasible region, which is a convex polytope, the linear programming algorithm finds a point in the polyhedron where this function has the smallest (or largest) value if such a point exists. The simplex method is a specific iterative technique for linear programs which aims to take a given basic feasible solution to another basic feasible solution for which the objective function is smaller, thus producing an iterative procedure for optimizing.

Most Common Equations and Symbols

Linear Algebra

Linear System of Equations

$$\mathbf{A}\mathbf{x} = \mathbf{b}.$$

The matrix $\mathbf{A} \in \mathbb{R}^{p \times n}$ and vector $\mathbf{b} \in \mathbb{R}^p$ are generally known, and the vector $\mathbf{x} \in \mathbb{R}^n$ is unknown.

Eigenvalue Equation

$$\mathbf{A}\mathbf{T} = \mathbf{T}\mathbf{\Lambda}.$$

The columns $\boldsymbol{\xi}_k$ of the matrix $\mathbf{T}$ are the eigenvectors of $\mathbf{A} \in \mathbb{C}^{n \times n}$ corresponding to the eigenvalue λ_k: $\mathbf{A}\boldsymbol{\xi}_k = \lambda_k \boldsymbol{\xi}_k$. The matrix $\mathbf{\Lambda}$ is a diagonal matrix containing these eigenvalues, in the simple case with n distinct eigenvalues.

Change of Coordinates

$$\mathbf{x} = \mathbf{\Psi}\mathbf{a}.$$

The vector $\mathbf{x} \in \mathbb{R}^n$ may be written as $\mathbf{a} \in \mathbb{R}^n$ in the coordinate system given by the columns of $\mathbf{\Psi} \in \mathbb{R}^{n \times n}$.

Measurement Equation

$$\mathbf{y} = \mathbf{C}\mathbf{x}.$$

The vector $\mathbf{y} \in \mathbb{R}^p$ is a measurement of the state $\mathbf{x} \in \mathbb{R}^n$ by the measurement matrix $\mathbf{C} \in \mathbb{R}^{p \times n}$.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Singular Value Decomposition

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^* \approx \tilde{\mathbf{U}}\tilde{\mathbf{\Sigma}}\tilde{\mathbf{V}}^*.$$

The matrix $\mathbf{X} \in \mathbb{C}^{n\times m}$ may be decomposed into the product of three matrices $\mathbf{U} \in \mathbb{C}^{n\times n}$, $\mathbf{\Sigma} \in \mathbb{C}^{n\times m}$, and $\mathbf{V} \in \mathbb{C}^{m\times m}$. The matrices $\mathbf{U}$ and $\mathbf{V}$ are *unitary*, so that $\mathbf{U}\mathbf{U}^* = \mathbf{U}^*\mathbf{U} = \mathbf{I}_{n\times n}$ and $\mathbf{V}\mathbf{V}^* = \mathbf{V}^*\mathbf{V} = \mathbf{I}_{m\times m}$, where * denotes complex conjugate transpose. The columns of $\mathbf{U}$ (respectively $\mathbf{V}$) are orthogonal, called left (respectively right) *singular vectors*. The matrix $\mathbf{\Sigma}$ contains decreasing, non-negative diagonal entries called *singular values*.

Often, $\mathbf{X}$ is approximated with a low-rank matrix $\tilde{\mathbf{X}} = \tilde{\mathbf{U}}\tilde{\mathbf{\Sigma}}\tilde{\mathbf{V}}^*$, where $\tilde{\mathbf{U}}$ and $\tilde{\mathbf{V}}$ contain the first $r \ll n$ columns of $\mathbf{U}$ and $\mathbf{V}$, respectively, and $\tilde{\mathbf{\Sigma}}$ contains the first $r \times r$ block of $\mathbf{\Sigma}$. The matrix $\tilde{\mathbf{U}}$ is often denoted $\mathbf{\Psi}$ in the context of spatial modes, reduced-order models, and sensor placement.

Regression and Optimization

Over-determined and Under-determined Optimization for Linear Systems

$$\begin{aligned} &\underset{\mathbf{x}}{\operatorname{argmin}}(\|\mathbf{A}\mathbf{x}-\mathbf{b}\|_2 + \lambda g(\mathbf{x})) \qquad \text{or} \\ &\underset{\mathbf{x}}{\operatorname{argmin}}\, g(\mathbf{x}) \quad \text{subject to} \quad \|\mathbf{A}\mathbf{x}-\mathbf{b}\|_2 \leq \epsilon. \end{aligned}$$

Here $g(\mathbf{x})$ is a regression penalty (with penalty parameter λ for over-determined systems). For over- and under-determined linear systems of equations, which result in either no solutions or an infinite number of solutions of $\mathbf{A}\mathbf{x} = \mathbf{b}$, a choice of constraint or penalty, which is also known as *regularization*, must be made in order to produce a solution.

Over-determined and Under-determined Optimization for Nonlinear Systems

$$\begin{aligned} &\underset{\mathbf{x}}{\operatorname{argmin}}(f(\mathbf{A},\mathbf{x},\mathbf{b}) + \lambda g(\mathbf{x})) \qquad \text{or} \\ &\underset{\mathbf{x}}{\operatorname{argmin}}\, g(\mathbf{x}) \quad \text{subject to} \quad f(\mathbf{A},\mathbf{x},\mathbf{b}) \leq \epsilon. \end{aligned}$$

This generalizes the linear system to a nonlinear system $f(\cdot)$ with regularization $g(\cdot)$. These over- and under-determined systems are often solved using gradient descent algorithms.

Compositional Optimization for Neural Networks

$$\underset{\mathbf{A}_j}{\operatorname{argmin}}(f_M(\mathbf{A}_M, \dots f_2(\mathbf{A}_2, (f_1(\mathbf{A}_1,\mathbf{x}))\cdots)) + \lambda g(\mathbf{A}_j)).$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Each $\mathbf{A}_k$ denotes the weight connecting the neural network from the kth to the $(k+1)$th layer. It is typically a massively under-determined system which is regularized by $g(\mathbf{A}_j)$. Composition and regularization are critical for generating expressive representations and preventing overfitting. The full network is often denoted $\mathbf{f}_{\boldsymbol{\theta}}$.

Dynamical Systems and Reduced-Order Models

Nonlinear Ordinary Differential Equation (Dynamical System)

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{x}(t) = \mathbf{f}(\mathbf{x}(t), t; \boldsymbol{\beta}).$$

The vector $\mathbf{x}(t) \in \mathbb{R}^n$ is the state of the system evolving in time t, $\boldsymbol{\beta}$ are parameters, and $\mathbf{f}$ is the vector field. Generally, $\mathbf{f}$ is Lipschitz continuous to guarantee existence and uniqueness of solutions.

Linear Input–Output System

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{x} &= \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u},\\ \mathbf{y} &= \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u}.\end{aligned}$$

The state of the system is $\mathbf{x} \in \mathbb{R}^n$, the inputs (actuators) are $\mathbf{u} \in \mathbb{R}^q$, and the outputs (sensors) are $\mathbf{y} \in \mathbb{R}^p$. The matrices $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, and $\mathbf{D}$ define the dynamics, the effect of actuation, the sensing strategy, and the effect of actuation feedthrough, respectively.

Nonlinear Map (Discrete-Time Dynamical System)

$$\mathbf{x}_{k+1} = \mathbf{F}(\mathbf{x}_k).$$

The state of the system at the kth iteration is $\mathbf{x}_k \in \mathbb{R}^n$, and $\mathbf{F}$ is a possibly nonlinear mapping. Often, this map defines an iteration forward in time, so that $\mathbf{x}_k = \mathbf{x}(k\Delta t)$; in this case the flow map is denoted $\mathbf{F}_{\Delta t}$.

Koopman Operator Equation (Discrete-Time)

$$\mathcal{K}_t g = g \circ \mathbf{F}_t \quad \Longrightarrow \quad \mathcal{K}_t \varphi = \lambda \varphi.$$

The linear Koopman operator $\mathcal{K}_t$ advances measurement functions of the state $g(\mathbf{x})$ with the flow $\mathbf{F}_t$. The eigenvalues and eigenvectors of $\mathcal{K}_t$ are λ and $\varphi(\mathbf{x})$, respectively. The operator $\mathcal{K}_t$ operates on a Hilbert space of measurements.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Nonlinear Partial Differential Equation (PDE)

$$\mathbf{u}_t = \mathbf{N}(\mathbf{u}, \mathbf{u}_x, \mathbf{u}_{xx}, \ldots, x, t; \boldsymbol{\beta}).$$

The state of the PDE is $\mathbf{u}$, the nonlinear evolution operator is $\mathbf{N}$, subscripts denote partial differentiation, and x and t are the spatial and temporal variables, respectively. The PDE is parameterized by values in $\boldsymbol{\beta}$. The state $\mathbf{u}$ of the PDE may be a continuous function $u(x,t)$, or it may be discretized at several spatial locations, $\mathbf{u}(t) = \begin{bmatrix} u(x_1,t) & u(x_2,t) & \ldots & u(x_n,t) \end{bmatrix}^T \in \mathbb{R}^n$.

Galerkin Expansion
The continuous Galerkin expansion is

$$u(x,t) \approx \sum_{k=1}^{r} a_k(t)\psi_k(x).$$

The functions $a_k(t)$ are temporal coefficients that capture the time dynamics, and $\psi_k(x)$ are spatial modes. For a high-dimensional discretized state, the Galerkin expansion becomes $\mathbf{u}(t) \approx \sum_{k=1}^{r} a_k(t)\boldsymbol{\psi}_k$. The spatial modes $\boldsymbol{\psi}_k \in \mathbb{R}^n$ may be the columns of $\boldsymbol{\Psi} = \tilde{\mathbf{U}}$.

Complete Symbols

Dimensions

K	Number of non-zero entries in a K-sparse vector $\mathbf{s}$
m	Number of data snapshots (i.e., columns of $\mathbf{X}$)
n	Dimension of the state, $\mathbf{x} \in \mathbb{R}^n$
p	Dimension of the measurement or output variable, $\mathbf{y} \in \mathbb{R}^p$
q	Dimension of the input variable, $\mathbf{u} \in \mathbb{R}^q$
r	Rank of truncated SVD, or other low-rank approximation

Scalars

s	Frequency in Laplace domain
t	Time
δ	Learning rate in gradient descent
Δt	Time-step
x	Spatial variable
Δx	Spatial step

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

σ Singular value
λ Eigenvalue
λ Sparsity parameter for sparse optimization (Section 7.3)
λ Lagrange multiplier (Sections 3.7, 8.4, and 12.4)
τ Threshold

Vectors

$\mathbf{a}$ Vector of mode amplitudes of $\mathbf{x}$ in basis $\mathbf{\Psi}$, $\mathbf{a} \in \mathbb{R}^r$
$\mathbf{a}$ Action of reinforcement learning agent (Chapter 11)
$\mathbf{b}$ Vector of measurements in linear system $\mathbf{Ax} = \mathbf{b}$
$\mathbf{b}$ Vector of DMD mode amplitudes (Section 7.2)
$\mathbf{Q}$ Vector containing potential function for PDE-FIND
$\mathbf{r}$ Residual error vector
$\mathbf{s}$ Sparse vector, $\mathbf{s} \in \mathbb{R}^n$ (Chapter 3)
$\mathbf{s}$ State of the environment in reinforcement learning (Chapter 11)
$\mathbf{u}$ Control variable (Chapters 8, 9, and 10)
$\mathbf{u}$ PDE state vector (Chapters 12 and 13)
$\mathbf{w}$ Exogenous inputs
$\mathbf{w}_d$ Disturbances to system
$\mathbf{w}_n$ Measurement noise
$\mathbf{w}_r$ Reference to track
$\mathbf{x}$ State of a system, $\mathbf{x} \in \mathbb{R}^n$
$\mathbf{x}_k$ Snapshot of data at time t_k
$\mathbf{x}_j$ Data sample $j \in Z := \{1, 2, \ldots, m\}$ (Chapters 5 and 6)
$\tilde{\mathbf{x}}$ Reduced state, $\tilde{\mathbf{x}} \in \mathbb{R}^r$, so that $\mathbf{x} \approx \tilde{\mathbf{U}}\tilde{\mathbf{x}}$
$\hat{\mathbf{x}}$ Estimated state of a system
$\mathbf{y}$ Vector of measurements, $\mathbf{y} \in \mathbb{R}^p$
$\mathbf{y}_j$ Data label $j \in Z := \{1, 2, \ldots, m\}$ (Chapters 5 and 6)
$\hat{\mathbf{y}}$ Estimated output measurement
$\mathbf{z}$ Transformed state, $\mathbf{x} = \mathbf{Tz}$ (Chapters 8 and 9)
$\boldsymbol{\epsilon}$ Error vector
$\boldsymbol{\beta}$ Bifurcation parameters
$\boldsymbol{\xi}$ Eigenvector of Koopman operator (Sections 7.4 and 7.5)
$\boldsymbol{\xi}$ Sparse vector of coefficients (Section 7.3)
$\boldsymbol{\theta}$ Neural network parameters
$\boldsymbol{\phi}$ DMD mode
$\boldsymbol{\psi}$ POD mode
$\boldsymbol{\Upsilon}$ Vector of PDE measurements for PDE-FIND

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Matrices

$\mathbf{A}$	Matrix for system of equations or dynamics
$\tilde{\mathbf{A}}$	Reduced dynamics on r-dimensional POD subspace
$\mathbf{A}_{\mathbf{X}}$	Matrix representation of linear dynamics on the state $\mathbf{x}$
$\mathbf{A}_{\mathbf{Y}}$	Matrix representation of linear dynamics on the observables $\mathbf{y}$
$(\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D})$	Matrices for continuous-time state-space system
$(\mathbf{A}_d, \mathbf{B}_d, \mathbf{C}_d, \mathbf{D}_d)$	Matrices for discrete-time state-space system
$(\hat{\mathbf{A}}, \hat{\mathbf{B}}, \hat{\mathbf{C}}, \hat{\mathbf{D}})$	Matrices for state-space system in new coordinates $\mathbf{z} = \mathbf{T}^{-1}\mathbf{x}$
$(\tilde{\mathbf{A}}, \tilde{\mathbf{B}}, \tilde{\mathbf{C}}, \tilde{\mathbf{D}})$	Matrices for reduced state-space system with rank r
$\mathbf{B}$	Actuation input matrix
$\mathbf{C}$	Linear measurement matrix from state to measurements
$\mathcal{C}$	Controllability matrix
$\mathcal{F}$	Discrete Fourier transform
$\mathbf{G}$	Matrix representation of linear dynamics on the states and inputs $[\mathbf{x}^T \mathbf{u}^T]^T$
$\mathbf{H}$	Hankel matrix
$\mathbf{H}'$	Time-shifted Hankel matrix
$\mathbf{I}$	Identity matrix
$\mathbf{K}$	Matrix form of Koopman operator (Chapter 7)
$\mathbf{K}$	Closed-loop control gain (Chapter 8)
$\mathbf{K}_f$	Kalman filter estimator gain
$\mathbf{K}_r$	LQR control gain
$\mathbf{L}$	Low-rank portion of matrix $\mathbf{X}$ (Chapter 3)
$\mathcal{O}$	Observability matrix
$\mathbf{P}$	Unitary matrix that acts on columns of $\mathbf{X}$
$\mathbf{Q}$	Weight matrix for state penalty in LQR (Section 8.4)
$\mathbf{Q}$	Orthogonal matrix from QR factorization
$\mathbf{R}$	Weight matrix for actuation penalty in LQR (Section 8.4)
$\mathbf{R}$	Upper triangular matrix from QR factorization
$\mathbf{S}$	Sparse portion of matrix $\mathbf{X}$ (Chapter 3)
$\mathbf{T}$	Matrix of eigenvectors (Chapter 8)
$\mathbf{T}$	Change of coordinates (Chapters 8 and 9)
$\mathbf{U}$	Left singular vectors of $\mathbf{X}$, $\mathbf{U} \in \mathbb{R}^{n \times n}$
$\hat{\mathbf{U}}$	Left singular vectors of economy SVD of $\mathbf{X}$, $\mathbf{U} \in \mathbb{R}^{n \times m}$
$\tilde{\mathbf{U}}$	Left singular vectors (POD modes) of truncated SVD of $\mathbf{X}$, $\mathbf{U} \in \mathbb{R}^{n \times r}$
$\mathbf{V}$	Right singular vectors of $\mathbf{X}$, $\mathbf{V} \in \mathbb{R}^{m \times m}$
$\tilde{\mathbf{V}}$	Right singular vectors of truncated SVD of $\mathbf{X}$, $\mathbf{V} \in \mathbb{R}^{m \times r}$
$\mathbf{\Sigma}$	Matrix of singular values of $\mathbf{X}$, $\mathbf{\Sigma} \in \mathbb{R}^{n \times m}$
$\hat{\mathbf{\Sigma}}$	Matrix of singular values of economy SVD of $\mathbf{X}$, $\mathbf{\Sigma} \in \mathbb{R}^{m \times m}$
$\tilde{\mathbf{\Sigma}}$	Matrix of singular values of truncated SVD of $\mathbf{X}$, $\mathbf{\Sigma} \in \mathbb{R}^{r \times r}$
$\mathbf{W}$	Eigenvectors of $\tilde{\mathbf{A}}$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

$\mathbf{W}_c$ Controllability Gramian
$\mathbf{W}_o$ Observability Gramian
$\mathbf{X}$ Data matrix, $\mathbf{X} \in \mathbb{R}^{n \times m}$
$\mathbf{X}'$ Time-shifted data matrix, $\mathbf{X}' \in \mathbb{R}^{n \times m}$
$\mathbf{Y}$ Projection of $\mathbf{X}$ matrix onto orthogonal basis in randomized SVD (Section 1.8)
$\mathbf{Y}$ Data matrix of observables, $\mathbf{Y} = \mathbf{g}(\mathbf{X})$, $\mathbf{Y} \in \mathbb{R}^{p \times m}$ (Chapter 7)
$\mathbf{Y}'$ Shifted data matrix of observables, $\mathbf{Y}' = \mathbf{g}(\mathbf{X}')$, $\mathbf{Y}' \in \mathbb{R}^{p \times m}$ (Chapter 7)
$\mathbf{Z}$ Sketch matrix for randomized SVD, $\mathbf{Z} \in \mathbb{R}^{n \times r}$ (Section 1.8)
$\mathbf{\Theta}$ Measurement matrix times sparsifying basis, $\mathbf{\Theta} = \mathbf{C}\mathbf{\Psi}$ (Chapter 3)
$\mathbf{\Theta}$ Matrix of candidate functions for SINDy (Section 7.3)
$\mathbf{\Gamma}$ Matrix of derivatives of candidate functions for SINDy (Section 7.3)
$\mathbf{\Xi}$ Matrix of coefficients of candidate functions for SINDy (Section 7.3)
$\mathbf{\Xi}$ Matrix of nonlinear snapshots for DEIM (Section 13.5)
$\mathbf{\Lambda}$ Diagonal matrix of eigenvalues
$\mathbf{\Upsilon}$ Input snapshot matrix, $\mathbf{\Upsilon} \in \mathbb{R}^{q \times m}$
$\mathbf{\Phi}$ Matrix of DMD modes, $\mathbf{\Phi} \triangleq \mathbf{X}'\mathbf{V}\mathbf{\Sigma}^{-1}\mathbf{W}$
$\mathbf{\Psi}$ Orthonormal basis (e.g., Fourier or POD modes)

Tensors

$(\mathcal{A}, \mathcal{B}, \mathcal{M})$ N-way array tensors of size $I_1 \times I_2 \times \cdots \times I_N$

Norms

$\|\cdot\|_0$ ℓ_0 pseudo-norm of a vector $\mathbf{x}$; the number of non-zero elements in $\mathbf{x}$
$\|\cdot\|_1$ ℓ_1-norm of a vector $\mathbf{x}$ given by $\|\mathbf{x}\|_1 = \sum_{i=1}^{n} |x_i|$
$\|\cdot\|_2$ ℓ_2-norm of a vector $\mathbf{x}$ given by $\|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^{n}(x_i^2)}$
$\|\cdot\|_2$ 2-norm of a matrix $\mathbf{X}$ given by $\|\mathbf{X}\|_2 = \max_{\mathbf{v} \neq \mathbf{0}} \|\mathbf{X}\mathbf{v}\|_2 / \|\mathbf{v}\|_2$
$\|\cdot\|_F$ Frobenius norm of a matrix $\mathbf{X}$ given by $\|\mathbf{X}\|_F = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{m} |X_{ij}|^2}$
$\|\cdot\|_*$ Nuclear norm of a matrix $\mathbf{X}$ given by $\|\mathbf{X}\|_* = \text{trace}(\sqrt{\mathbf{X}^*\mathbf{X}}) = \sum_{i=1}^{m} \sigma_i$ (for $m \leq n$)
$\langle \cdot, \cdot \rangle$ Inner product; for functions, $\langle f(x), g(x) \rangle = \int_{-\infty}^{\infty} f(x) g^*(x)\, \mathrm{d}x$.
$\langle \cdot, \cdot \rangle$ Inner product; for vectors, $\langle \mathbf{u}, \mathbf{v} \rangle = \mathbf{u}^*\mathbf{v}$.

Operators, Functions, and Maps

$\mathcal{F}$ Fourier transform
$\mathbf{F}$ Discrete-time dynamical system map
$\mathbf{F}_t$ Discrete-time flow map of dynamical system through time t
$\mathbf{f}_{\boldsymbol{\theta}}$ Neural network (Chapter 6)
$\mathbf{f}$ Continuous-time dynamical system (Chapter 7)

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

$\mathcal{G}$	Gabor transform
$\mathbf{G}$	Transfer function from inputs to outputs (Chapter 8)
g	Scalar measurement function on $\mathbf{x}$
$\mathbf{g}$	Vector-valued measurement functions on $\mathbf{x}$
J	Cost function for control
ℓ	Loss function for support vector machines (Chapter 5)
$\mathcal{K}$	Koopman operator (continuous-time)
$\mathcal{K}_t$	Koopman operator associated with time-t flow map
$\mathcal{L}$	Laplace transform
$\mathbf{L}$	Loop transfer function (Chapter 8)
$\mathbf{L}$	Linear partial differential equation (Chapters 12 and 13)
$\mathbf{N}$	Nonlinear partial differential equation
$\mathcal{O}$	Order of magnitude
Q	Quality function (Chapter 11)
$\Re$	Real part
$\mathbf{S}$	Sensitivity function (Chapter 8)
$\mathbf{T}$	Complementary sensitivity function (Chapter 8)
V	Value function (Chapter 11)
$\mathcal{W}$	Wavelet transform
μ	Incoherence between measurement matrix $\mathbf{C}$ and basis $\mathbf{\Psi}$
κ	Condition number
$\boldsymbol{\pi}$	Policy function for agent in reinforcement learning (Chapter 11)
φ	Koopman eigenfunction
∇	Gradient operator
$*$	Convolution operator

Most Common Acronyms

CNN	Convolutional neural network
DL	Deep learning
DMD	Dynamic mode decomposition
FFT	Fast Fourier transform
ODE	Ordinary differential equation
PCA	Principal component analysis
PDE	Partial differential equation
POD	Proper orthogonal decomposition
RL	Reinforcement learning
ROM	Reduced-order model
SVD	Singular value decomposition

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Other Acronyms

ADM	Alternating directions method
AIC	Akaike information criterion
ALM	Augmented Lagrange multiplier
ANN	Artificial neural network
ARMA	Autoregressive moving average
ARMAX	Autoregressive moving average with exogenous input
BIC	Bayesian information criterion
BPOD	Balanced proper orthogonal decomposition
DMDc	Dynamic mode decomposition with control
CCA	Canonical correlation analysis
CFD	Computational fluid dynamics
CoSaMP	Compressive sampling matching pursuit
CWT	Continuous wavelet transform
DEIM	Discrete empirical interpolation method
DCT	Discrete cosine transform
DFT	Discrete Fourier transform
DMDc	Dynamic mode decomposition with control
DNS	Direct numerical simulation
DP	Dynamic programming
DQN	Deep Q network
DRL	Deep reinforcement learning
DWT	Discrete wavelet transform
ECOG	Electrocorticography
eDMD	Extended DMD
EIM	Empirical interpolation method
EM	Expectation maximization
EOF	Empirical orthogonal functions
ERA	Eigensystem realization algorithm
ESC	Extremum-seeking control
GMM	Gaussian mixture model
HAVOK	Hankel alternative view of Koopman
HER	Hindsight experience replay
HJB	Hamilton-Jacobi-Bellman equation
JL	Johnson–Lindenstrauss
KL	Kullback–Leibler
ICA	Independent component analysis
KLT	Karhunen–Loève transform
LAD	Least absolute deviations
LASSO	Least absolute shrinkage and selection operator
LDA	Linear discriminant analysis

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

LQE	Linear–quadratic estimator
LQG	Linear–quadratic Gaussian controller
LQR	Linear–quadratic regulator
LTI	Linear time-invariant system
MDP	Markov decision process
MIMO	Multiple-input, multiple-output
MLC	Machine learning control
MPE	Missing point estimation
mrDMD	Multi-resolution dynamic mode decomposition
NARMAX	Nonlinear autoregressive model with exogenous inputs
NLS	Nonlinear Schrödinger equation
OKID	Observer Kalman filter identification
PBH	Popov–Belevitch–Hautus test
PCP	Principal component pursuit
PDE-FIND	Partial differential equation functional identification of nonlinear dynamics
PDF	Probability density function
PID	Proportional–integral–derivative control
PINN	Physics-informed neural network
PIV	Particle image velocimetry
RIP	Restricted isometry property
rSVD	Randomized SVD
RKHS	Reproducing kernel Hilbert space
RNN	Recurrent neural network
RPCA	Robust principal component analysis
SGD	Stochastic gradient descent
SINDy	Sparse identification of nonlinear dynamics
SINDYc	SINDy with control
SISO	Single-input, single-output
SRC	Sparse representation for classification
SSA	Singular spectrum analysis
STFT	Short-time Fourier transform
STLS	Sequential thresholded least-squares
SVM	Support vector machine
TICA	Time-lagged independent component analysis
VAC	Variational approach of conformation dynamics

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Part I

Dimensionality Reduction and Transforms

Chapter 1

Singular Value Decomposition (SVD)

The singular value decomposition (SVD) is among the most important matrix factorizations of the computational era, providing a foundation for nearly all of the data methods in this book. The SVD provides a numerically stable matrix decomposition that can be used for a variety of purposes and is guaranteed to exist. We will use the SVD to obtain optimal low-rank approximations to matrices and to perform pseudo-inverses of non-square matrices to find a solution to the system of equations $\mathbf{Ax} = \mathbf{b}$. The SVD will also be used as the underlying algorithm of principal component analysis (PCA), where high-dimensional data is decomposed into its most statistically descriptive factors. SVD/PCA has been applied to a wide variety of problems in science and engineering.

In a sense, the SVD generalizes the concept of the fast Fourier transform (FFT), which will be the subject of the next chapter. Many engineering texts begin with the FFT, as it is the basis of many classical analytical and numerical results. However, the FFT works in idealized settings, and the SVD is a more generic data-driven technique. Because this book is focused on data, we begin with the SVD, which may be thought of as providing a basis that is *tailored* to the specific data, as opposed to the FFT, which provides a *generic* basis.

In many domains, complex systems will generate data that is naturally arranged in large matrices, or more generally in arrays. For example, a time series of data from an experiment or a simulation may be arranged in a matrix, with each column containing all of the measurements at a given time. If the data at each instant in time is multi-dimensional, as in a high-resolution simulation of the weather in three spatial dimensions, it is possible to reshape or *flatten* this data into a high-dimensional column vector, forming the columns of a large matrix. Similarly, the pixel values in a grayscale image may be stored in a matrix, or these images may be reshaped into large column vectors in a matrix to represent the frames of a movie. Remarkably, the data generated by these systems is typically low-rank, meaning that there are a few dominant patterns that explain the high-dimensional data. The SVD is a numerically robust and efficient method of extracting these patterns from data.

1.1 Overview

Here we introduce the singular value decomposition (SVD) and develop an intuition for how to apply the SVD by demonstrating its use on a number of motivating examples. The SVD will provide a foundation for many other techniques developed in this book, including classification methods in Chapter 5, the dynamic mode decomposition (DMD) in Chapter 7, and the proper orthogonal decomposition (POD) in Chapter 12. Detailed mathematical properties are discussed in the following sections.

High dimensionality is a common challenge in processing data from complex systems. These systems may involve large measured data sets including audio, image, or video data. The data may also be generated from a physical system, such as neural recordings from a brain, or fluid velocity measurements from a simulation or experiment. In many naturally occurring systems, it is observed that data exhibit dominant patterns, which may be characterized by a low-dimensional attractor or manifold [334, 335].

As an example, consider images, which typically contain a large number of measurements (pixels), and are therefore elements of a high-dimensional vector space. However, most images are highly compressible, meaning that the relevant information may be represented in a much lower-dimensional subspace. The compressibility of images will be discussed in depth throughout this book. Complex fluid systems, such as the Earth's atmosphere or the turbulent wake behind a vehicle, also provide compelling examples of the low-dimensional structure underlying a high-dimensional state space. Although high-fidelity fluid simulations typically require at least millions or billions of degrees of freedom, there are often dominant coherent structures in the flow, such as periodic vortex shedding behind vehicles or hurricanes in the weather.

The SVD provides a systematic way to determine a low-dimensional approximation to high-dimensional data in terms of dominant patterns. This technique is *data-driven* in that patterns are discovered purely from data, without the addition of expert knowledge or intuition. The SVD is numerically stable and provides a hierarchical representation of the data in terms of a new coordinate system defined by dominant correlations within the data. Moreover, the SVD is guaranteed to exist for any matrix, unlike the eigendecomposition.

The SVD has many powerful applications beyond dimensionality reduction of high-dimensional data. It is used to compute the pseudo-inverse of non-square matrices, providing solutions to under-determined or over-determined matrix equations, $\mathbf{A}\mathbf{x} = \mathbf{b}$. We will also use the SVD to de-noise data sets. The SVD is likewise important to characterize the input and output geometry of a linear map between vector spaces. These applications will all be explored in this chapter, providing an intuition for matrices and high-dimensional data.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Definition of the SVD

Generally, we are interested in analyzing a large data set $\mathbf{X} \in \mathbb{C}^{n \times m}$:

$$\mathbf{X} = \begin{bmatrix} | & | & & | \\ \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_m \\ | & | & & | \end{bmatrix}. \tag{1.1}$$

The columns $\mathbf{x}_k \in \mathbb{C}^n$ may be measurements from simulations or experiments. For example, columns may represent images that have been reshaped into column vectors with as many elements as pixels in the image. The column vectors may also represent the state of a physical system that is evolving in time, such as the fluid velocity at a set of discrete points, a set of neural measurements, or the state of a weather simulation with one square kilometer resolution.

The index k is a label indicating the kth distinct set of measurements. For many of the examples in this book, $\mathbf{X}$ will consist of a *time series* of data, and $\mathbf{x}_k = \mathbf{x}(k\Delta t)$. Often the *state dimension* n is very large, on the order of millions or billions of degrees of freedom. The columns are often called *snapshots*, and m is the number of snapshots in $\mathbf{X}$. For many systems $n \gg m$, resulting in a *tall-skinny* matrix, as opposed to a *short-fat* matrix when $n \ll m$.

The SVD is a unique matrix decomposition that exists for every complex-valued matrix $\mathbf{X} \in \mathbb{C}^{n \times m}$:

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^*, \tag{1.2}$$

where $\mathbf{U} \in \mathbb{C}^{n \times n}$ and $\mathbf{V} \in \mathbb{C}^{m \times m}$ are *unitary* matrices[1] with orthonormal columns, and $\mathbf{\Sigma} \in \mathbb{R}^{n \times m}$ is a matrix with real, non-negative entries on the diagonal and zeros off the diagonal. Here * denotes the complex conjugate transpose.[2] As we will discover throughout this chapter, the condition that $\mathbf{U}$ and $\mathbf{V}$ are unitary is used extensively.

When $n \geq m$, the matrix $\mathbf{\Sigma}$ has at most m non-zero elements on the diagonal, and may be written as

$$\mathbf{\Sigma} = \begin{bmatrix} \hat{\mathbf{\Sigma}} \\ \mathbf{0} \end{bmatrix}.$$

Therefore, it is possible to *exactly* represent $\mathbf{X}$ using the *economy* SVD:

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^* = \begin{bmatrix} \hat{\mathbf{U}} & \hat{\mathbf{U}}^\perp \end{bmatrix} \begin{bmatrix} \hat{\mathbf{\Sigma}} \\ \mathbf{0} \end{bmatrix} \mathbf{V}^* = \hat{\mathbf{U}}\hat{\mathbf{\Sigma}}\mathbf{V}^*. \tag{1.3}$$

The full SVD and economy SVD are shown in Fig. 1.1. The columns of $\hat{\mathbf{U}}^\perp$ span a vector space that is complementary and orthogonal to that spanned by $\hat{\mathbf{U}}$. The

[1] A square matrix $\mathbf{U}$ is unitary if $\mathbf{U}\mathbf{U}^* = \mathbf{U}^*\mathbf{U} = \mathbf{I}$.

[2] For real-valued matrices, this is the same as the regular transpose $\mathbf{X}^* = \mathbf{X}^T$.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

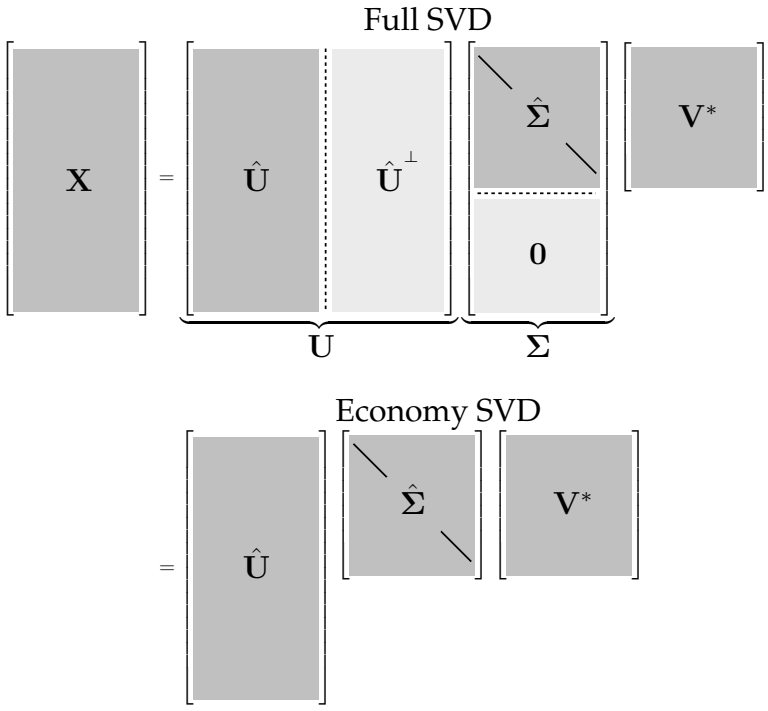


Figure 1.1: Schematic of matrices in the full and economy SVD.

columns of $\mathbf{U}$ are called *left singular vectors* of $\mathbf{X}$ and the columns of $\mathbf{V}$ are *right singular vectors*. The diagonal elements of $\hat{\mathbf{\Sigma}} \in \mathbb{C}^{m\times m}$ are called *singular values* and they are ordered from largest to smallest. The rank of $\mathbf{X}$ is equal to the number of non-zero singular values. We will show in Section 1.2 that the SVD can also be used to obtain an optimal rank-r approximation of $\mathbf{X}$ for $r < m$.

Computing the SVD

The SVD is a cornerstone of computational science and engineering, and the numerical implementation of the SVD is both important and mathematically enlightening. That said, most standard numerical implementations are mature and a simple interface exists in many modern computer languages, allowing us to abstract away the details underlying the SVD computation. For most purposes, we simply use the SVD as a part of a larger effort, and we take for granted the existence of efficient and stable numerical algorithms. Numerically, the SVD may be computed by first reducing the matrix $\mathbf{X}$ to a bidiagonal matrix and then using an iterative algorithm to compute the SVD of the bidiagonal matrix. For matrices with high aspect ratio (i.e., $n \gg m$), then the first step may be achieved by first computing a QR factorization to reduce $\mathbf{X}$ to an upper triangular matrix, followed by Householder reflections to reduce this upper triangular matrix into a bidiagonal form. The second step may be performed using a modified QR algorithm developed by Golub & Kahan [285].

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Details of the QR factorization are beyond the scope of this book, although it is a straightforward greedy method that is closely related to Gram–Schmidt orthogonalization. There are numerous variations, alternatives, and important results on the computation of the SVD; for a more thorough discussion, see [144, 285, 286, 318, 389, 711]. Randomized numerical algorithms are increasingly used to compute the SVD of very large matrices as discussed in Section 1.8.

MATLAB

In MATLAB, computing the SVD is straightforward:

```
>> X = randn(5,3);    % Create a 5x3 random data matrix
>> [U,S,V] = svd(X); % Singular value decomposition
```

For non-square matrices $\mathbf{X}$, the economy SVD is more efficient:

```
>> [Uhat,Shat,V] = svd(X,'econ');  % Economy sized SVD
```

Python[3]

```
>>> import numpy as np
>>> X = np.random.rand(5, 3)  # Create random data matrix
>>> U, S, VT = np.linalg.svd(X,full_matrices=True) #Full SVD
>>> Uhat, Shat, VThat = np.linalg.svd(X,full_matrices=False)
     # Economy SVD
```

R

```
> X <- replicate(3, rnorm(5))
> s <- svd(X)
> U <- s$u
> S <- diag(s$d)
> V <- s$v
```

Mathematica

```
In:= X=RandomReal[{0,1},{5,3}]
In:= {U,S,V} = SingularValueDecomposition[X]
```

[3]Note that Python outputs the transpose of $\mathbf{V}$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Other Languages

The SVD is also available in other languages, such as Fortran and C++. In fact, most SVD implementations are based on the LAPACK (Linear Algebra Package) [19] in Fortran. The SVD routine is designated **DGESVD** in LAPACK, and this is wrapped in the C++ libraries **Armadillo** and **Eigen**.

Historical Perspective

The SVD has a long and rich history, ranging from early work developing the theoretical foundations to modern work on computational stability and efficiency. There is an excellent historical review by Stewart [676], which provides context and many important details. The review focuses on the early theoretical work of Beltrami and Jordan (1873), Sylvester (1889), Schmidt (1907), and Weyl (1912). It also discusses more recent work, including the seminal computational work of Golub and collaborators [285, 286]. In addition, there are many excellent chapters on the SVD in modern texts [24, 420, 711].

Uses in This Book and Assumptions of the Reader

The SVD is the basis for many related techniques in dimensionality reduction. These methods include principal component analysis (PCA) in statistics [339, 340, 552], the Karhunen–Loève transform (KLT) [374, 453], empirical orthogonal functions (EOFs) in climate [459], the proper orthogonal decomposition (POD) in fluid dynamics [335], and canonical correlation analysis (CCA) [176]. Although developed independently in a range of diverse fields, many of these methods only differ in how the data is collected and pre-processed. There is an excellent discussion about the relationship between the SVD, the KLT, and PCA by Gerbrands [275].

The SVD is also widely used in system identification and control theory to obtain reduced-order models that are balanced in the sense that states are hierarchically ordered in terms of their ability to be observed by measurements and controlled by actuation [509].

For this chapter, we assume that the reader is familiar with linear algebra, with some experience in computation and numerics. For review, there are a number of excellent books on numerical linear algebra, with discussions on the SVD [24, 420, 711].

1.2 Matrix Approximation

Perhaps the most useful and defining property of the SVD is that it provides an *optimal* low-rank approximation to a matrix $\mathbf{X}$. In fact, the SVD provides a

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

hierarchy of low-rank approximations, since a rank-r approximation is obtained by keeping the leading r singular values and vectors, and discarding the rest.

Because $\mathbf{\Sigma}$ is diagonal, it is possible to express the matrix $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^*$ as a sum of rank-one matrices:

$$\mathbf{X} = \sum_{k=1}^{m} \sigma_k \mathbf{u}_k \mathbf{v}_k^* = \sigma_1 \mathbf{u}_1 \mathbf{v}_1^* + \sigma_2 \mathbf{u}_2 \mathbf{v}_2^* + \cdots + \sigma_m \mathbf{u}_m \mathbf{v}_m^*, \tag{1.4}$$

where σ_k is the kth diagonal entry of $\mathbf{\Sigma}$, and $\mathbf{u}_k$ and $\mathbf{v}_k$ are the kth columns of $\mathbf{U}$ and $\mathbf{V}$, respectively. This is known as the *dyadic* summation. The singular values σ_k are arranged in decreasing order, $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_m \geq 0$, so each subsequent rank-one matrix $\sigma_k \mathbf{u}_k \mathbf{v}_k^*$ is less important than the previous matrix in capturing the information in $\mathbf{X}$. For many systems, the singular values σ_k decrease rapidly, and it is possible to obtain a good approximation of $\mathbf{X}$ by truncating at some rank r:

$$\mathbf{X} \approx \tilde{\mathbf{X}} = \sum_{k=1}^{r} \sigma_k \mathbf{u}_k \mathbf{v}_k^* = \sigma_1 \mathbf{u}_1 \mathbf{v}_1^* + \sigma_2 \mathbf{u}_2 \mathbf{v}_2^* + \cdots + \sigma_r \mathbf{u}_r \mathbf{v}_r^*. \tag{1.5}$$

Here, we establish the notation that a truncated SVD basis (and the resulting approximated matrix $\tilde{\mathbf{X}}$) will be denoted by $\tilde{\mathbf{X}} = \tilde{\mathbf{U}}\tilde{\mathbf{\Sigma}}\tilde{\mathbf{V}}^*$, where $\tilde{\mathbf{U}}$ and $\tilde{\mathbf{V}}$ contain the first r columns of $\mathbf{U}$ and $\mathbf{V}$, and $\tilde{\mathbf{\Sigma}}$ contains the first $r \times r$ sub-block of $\mathbf{\Sigma}$. The truncated SVD is illustrated in Fig. 1.2, with $\tilde{\mathbf{U}}$, $\tilde{\mathbf{\Sigma}}$, and $\tilde{\mathbf{V}}$ denoting the truncated matrices. If $\mathbf{X}$ does not have full rank, then some of the singular values in $\hat{\mathbf{\Sigma}}$ may be zero, and the truncated SVD may still be exact. However, for truncation values r that are smaller than the number of non-zero singular values (i.e., the rank of $\mathbf{X}$), the truncated SVD only approximates $\mathbf{X}$.

For a given rank r, there is no better approximation for $\mathbf{X}$, in the ℓ_2 sense, than the truncated SVD approximation $\tilde{\mathbf{X}}$. The Eckart–Young theorem below will state this precisely and provide expressions for the error of the truncated approximation. There are numerous choices for the truncation rank r, and they are discussed in Section 1.7. Thus, high-dimensional data may be well described by a few dominant patterns given by the columns of $\tilde{\mathbf{U}}$ and $\tilde{\mathbf{V}}$.

This is an important property of the SVD, and we will return to it many times. There are numerous examples of data sets that contain high-dimensional measurements, resulting in a large data matrix $\mathbf{X}$. However, there are often dominant low-dimensional patterns in the data, and the truncated SVD basis $\tilde{\mathbf{U}}$ provides a coordinate transformation from the high-dimensional measurement space into a low-dimensional pattern space. This has the benefit of *reducing* the size and dimension of large data sets, yielding a tractable basis for visualization and analysis. Finally, many systems considered in this text are *dynamic* (see Chapter 7), and the SVD basis provides a hierarchy of modes that characterize the observed attractor, on which we may project a low-dimensional dynamical system to obtain reduced-order models (see Chapter 13).

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

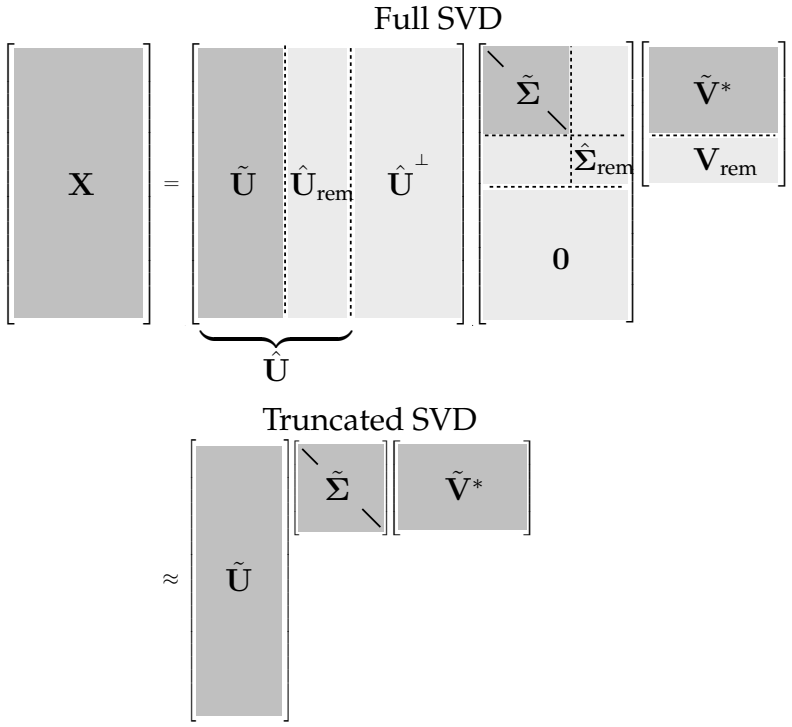


Figure 1.2: Schematic of truncated SVD. The subscript '$_{\text{rem}}$' denotes the remainder of $\hat{\mathbf{U}}$, $\hat{\mathbf{\Sigma}}$, or $\mathbf{V}$ after truncation.

Optimal Approximation and Error Bounds

Schmidt (of Gram–Schmidt) generalized the SVD to function spaces and developed an approximation theorem, establishing the truncated SVD $\tilde{\mathbf{X}}$ as the optimal low-rank approximation of the underlying matrix $\mathbf{X}$ [639]. Schmidt's approximation theorem was rediscovered by Eckart and Young [228], and is sometimes referred to as the Eckart–Young theorem.

Theorem 1.1 (Eckart–Young [228]) *The optimal rank-r approximation to $\mathbf{X}$, in a least-squares sense, is given by the rank-r SVD truncation $\tilde{\mathbf{X}}$:*

$$\underset{\tilde{\mathbf{X}},\ \text{s.t. rank}(\tilde{\mathbf{X}})=r}{\operatorname{argmin}} \|\mathbf{X}-\tilde{\mathbf{X}}\|_F = \tilde{\mathbf{U}}\tilde{\mathbf{\Sigma}}\tilde{\mathbf{V}}^*. \tag{1.6}$$

Again, $\tilde{\mathbf{U}}$ and $\tilde{\mathbf{V}}$ denote the first r leading columns of $\mathbf{U}$ and $\mathbf{V}$, and $\tilde{\mathbf{\Sigma}}$ contains the leading $r \times r$ sub-block of $\mathbf{\Sigma}$. The Frobenius norm above is defined as $\|\mathbf{X}\|_F = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{m}|X_{ij}|^2}$, which is equivalent to the 2-norm of the vectorized matrix $\mathbf{X}(:)$.

Thus, the Eckart–Young theorem guarantees that the truncated SVD provides the best matrix approximation of a given rank in the Frobenius norm. It

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

is also possible to exactly quantify the error of the rank-r SVD approximation:

$$\|\mathbf{X} - \tilde{\mathbf{X}}\|_F^2 = \sum_{k=r+1}^{m} \sigma_k^2. \tag{1.7}$$

Thus, all other rank-r matrices $\tilde{\mathbf{X}}$ will have at least this much error. Because the error scales with the size and magnitude of $\mathbf{X}$, it is often more useful to consider the relative error

$$\frac{\|\mathbf{X} - \tilde{\mathbf{X}}\|_F^2}{\|\mathbf{X}\|_F^2}. \tag{1.8}$$

This expression for the relative error in the Frobenius norm has two intuitive interpretations. If the columns of $\mathbf{X}$ are velocity fields, for example from a discretized fluid flow simulation, then this error is related to the fraction of the *kinetic energy* that is missing in the approximation $\tilde{\mathbf{X}}$. More generally, the squared Frobenius norm error of mean-subtracted data has the interpretation of the amount of missing variance in the approximation $\tilde{\mathbf{X}}$. This statistical interpretation will be explored more in Section 1.5.

Remarkably, the SVD also provides an optimal rank-r approximation in the matrix 2-norm, also known as the spectral norm:

$$\underset{\tilde{\mathbf{X}},\, \text{s.t. rank}(\tilde{\mathbf{X}})=r}{\operatorname{argmin}} \|\mathbf{X} - \tilde{\mathbf{X}}\|_2 = \tilde{\mathbf{U}}\tilde{\mathbf{\Sigma}}\tilde{\mathbf{V}}^*. \tag{1.9}$$

The 2-norm of a matrix $\mathbf{X}$ is induced by the vector 2-norm and is given by

$$\|\mathbf{X}\|_2 = \max_{\mathbf{v} \neq \mathbf{0}} \frac{\|\mathbf{X}\mathbf{v}\|_2}{\|\mathbf{v}\|_2}.$$

The error expression for the rank-r SVD approximation is even simpler in the 2-norm:

$$\|\mathbf{X} - \tilde{\mathbf{X}}\|_2 = \sigma_{r+1}. \tag{1.10}$$

This error expression is rather simple to derive by expanding

$$\mathbf{X} - \tilde{\mathbf{X}} = \sum_{k=r+1}^{m} \sigma_k \mathbf{u}_k \mathbf{v}_k^*. \tag{1.11}$$

Since each of the $\mathbf{v}_k$ vectors is orthonormal, the maximum $\|(\mathbf{X} - \tilde{\mathbf{X}})\mathbf{v}\|_2 = \sigma_{r+1}$ is achieved for $\mathbf{v} = \mathbf{v}_{r+1}$.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Example: Image Compression

We demonstrate the idea of matrix approximation with a simple example: image compression. A recurring theme throughout this book is that large data sets often contain underlying patterns that facilitate low-rank representations. Natural images present a simple and intuitive example of this inherent *compressibility*. A grayscale image may be thought of as a real-valued matrix $\mathbf{X} \in \mathbb{R}^{n \times m}$, where n and m are the number of pixels in the vertical and horizontal directions, respectively.[4] Depending on the basis of representation (pixel space, Fourier frequency domain, SVD transform coordinates), images may have very compact approximations.

Consider the image of Mordecai the snow dog in Fig. 1.3. This image has 2000×1500 pixels. It is possible to take the SVD of this image and plot the diagonal singular values, as in Fig. 1.4. Figure 1.3 shows the approximate matrix $\tilde{\mathbf{X}}$ for various truncation values r. By $r = 100$, the reconstructed image is quite accurate, and the singular values account for almost 80% of the total cumulative sum of the singular values. The squared error is less than 4% in the Frobenius norm. The SVD truncation results in a compression of the original image, since only the first 100 columns of $\mathbf{U}$ and $\mathbf{V}$, along with the first 100 diagonal elements of $\mathbf{\Sigma}$, must be stored in $\tilde{\mathbf{U}}$, $\tilde{\mathbf{\Sigma}}$, and $\tilde{\mathbf{V}}$.

Code 1.1: [MATLAB] Use SVD to compress image.

```
% First, we load the image
A=imread('../DATA/dog.jpg');
X=double(rgb2gray(A)); % Convert RBG->gray, 256 bit->double.
nx = size(X,1); ny = size(X,2);
imagesc(X), axis off, colormap gray

% Take the SVD
[U,S,V] = svd(X);

% Approximate matrix with truncated SVD for various ranks r
for r=[5 20 100]  % Truncation value
    Xapprox = U(:,1:r)*S(1:r,1:r)*V(:,1:r)'; % Approx. image
    figure, imagesc(Xapprox), axis off
    title(['r=',num2str(r,'%d')]);
end

% Plot singular values and cumulative sum
subplot(1,2,1), semilogy(diag(S),'k')
subplot(1,2,2), plot(cumsum(diag(S))/sum(diag(S)),'k')
```

[4] It is not uncommon for image size to be specified as horizontal by vertical, i.e., $\mathbf{X}^T \in \mathbb{R}^{m \times n}$, although we stick with vertical by horizontal to be consistent with generic matrix notation.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

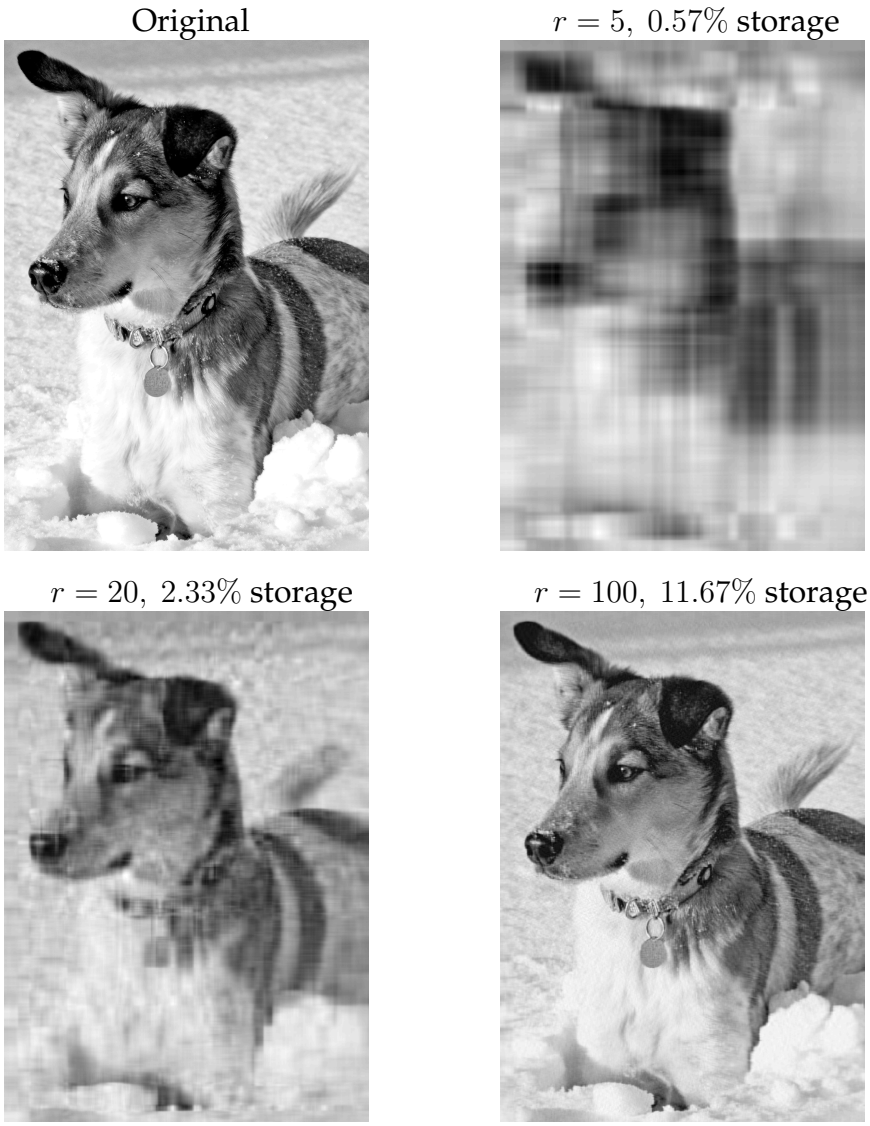


Figure 1.3: Image compression of Mordecai the snow dog, truncating the SVD at various ranks r. Original image resolution is 2000×1500.

Code 1.1: [Python] Use SVD to compress image.

```
# First, we load the image
from matplotlib.image import imread
A = imread(os.path.join('..','DATA','dog.jpg'))
X = np.mean(A, -1); # Convert RGB to grayscale
img = plt.imshow(X)
# Take the SVD
U, S, VT = np.linalg.svd(X,full_matrices=False)
S = np.diag(S)
# Approximate matrix with truncated SVD for various ranks r
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

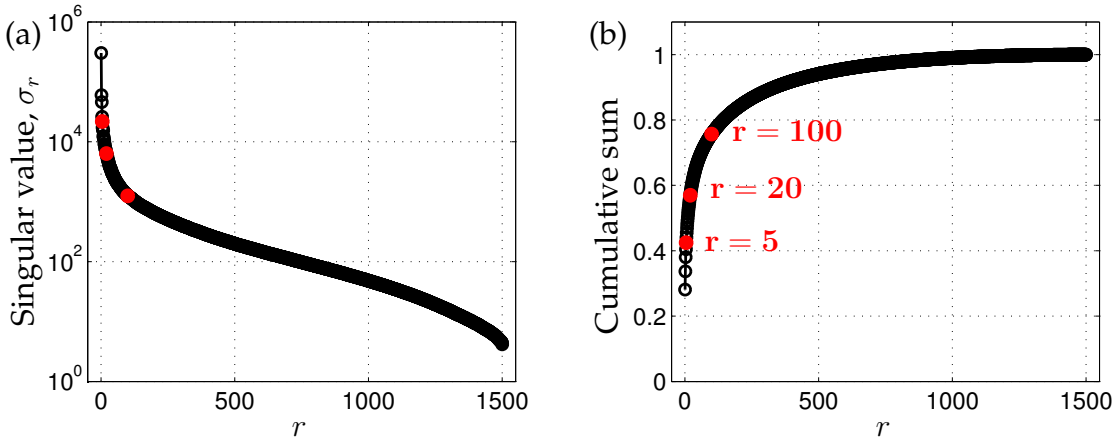


Figure 1.4: (a) Singular values σ_r and (b) cumulative sum $\sum_{k=1}^{r} \sigma_k$ of the first r singular values.

```
for r in (5, 20, 100):     # Construct approximate image
    Xapprox = U[:,:r] @ S[0:r,:r] @ VT[:r,:]
    img = plt.imshow(Xapprox)
    plt.show()
# Plot singular values and cumulative sum
plt.semilogy(np.diag(S))
plt.plot(np.cumsum(np.diag(S))/np.sum(np.diag(S)))
```

1.3 Mathematical Properties and Manipulations

Here we describe important mathematical properties of the SVD, including geometric interpretations of the unitary matrices $\mathbf{U}$ and $\mathbf{V}$, as well as a discussion of the SVD in terms of dominant correlations in the data $\mathbf{X}$. The relationship between the SVD and correlations in the data will be explored more in Section 1.5 on principal component analysis.

Interpretation as Dominant Correlations

The SVD is closely related to an eigenvalue problem involving the correlation matrices $\mathbf{XX}^*$ and $\mathbf{X}^*\mathbf{X}$, shown in Fig. 1.5 for a specific image, and in Figs. 1.6 and 1.7 for generic matrices. If we plug (1.3) into the row-wise correlation matrix $\mathbf{XX}^*$ and the column-wise correlation matrix $\mathbf{X}^*\mathbf{X}$, we find

$$\mathbf{XX}^* = \mathbf{U}\begin{bmatrix}\hat{\mathbf{\Sigma}}\\ \mathbf{0}\end{bmatrix}\mathbf{V}^*\mathbf{V}\begin{bmatrix}\hat{\mathbf{\Sigma}} & \mathbf{0}\end{bmatrix}\mathbf{U}^* = \mathbf{U}\begin{bmatrix}\hat{\mathbf{\Sigma}}^2 & \mathbf{0}\\ \mathbf{0} & \mathbf{0}\end{bmatrix}\mathbf{U}^*, \tag{1.12a}$$

$$\mathbf{X}^*\mathbf{X} = \mathbf{V}\begin{bmatrix}\hat{\mathbf{\Sigma}} & \mathbf{0}\end{bmatrix}\mathbf{U}^*\mathbf{U}\begin{bmatrix}\hat{\mathbf{\Sigma}}\\ \mathbf{0}\end{bmatrix}\mathbf{V}^* = \mathbf{V}\hat{\mathbf{\Sigma}}^2\mathbf{V}^*. \tag{1.12b}$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

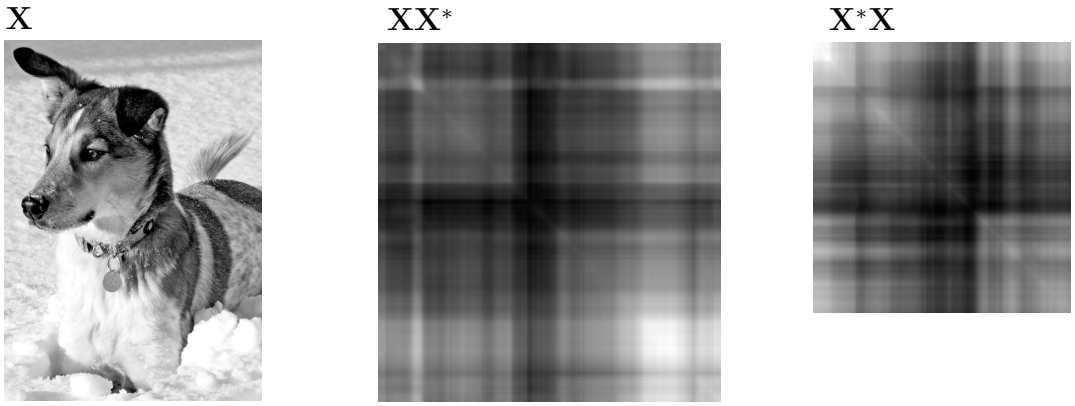


Figure 1.5: Correlation matrices $\mathbf{X}\mathbf{X}^*$ and $\mathbf{X}^*\mathbf{X}$ for a matrix $\mathbf{X}$ obtained from an image of a dog. Note that both correlation matrices are symmetric.

Recalling that $\mathbf{U}$ and $\mathbf{V}$ are unitary, $\mathbf{U}$, $\mathbf{\Sigma}$, and $\mathbf{V}$ are solutions to the following eigenvalue problems:

$$\mathbf{X}\mathbf{X}^*\mathbf{U} = \mathbf{U}\begin{bmatrix}\hat{\mathbf{\Sigma}}^2 & \mathbf{0}\\ \mathbf{0} & \mathbf{0}\end{bmatrix}, \tag{1.13a}$$

$$\mathbf{X}^*\mathbf{X}\mathbf{V} = \mathbf{V}\hat{\mathbf{\Sigma}}^2. \tag{1.13b}$$

In other words, each non-zero singular value of $\mathbf{X}$ is a positive square root of an eigenvalue of $\mathbf{X}^*\mathbf{X}$ and of $\mathbf{X}\mathbf{X}^*$, which have the same non-zero eigenvalues. It follows that, if $\mathbf{X}$ is self-adjoint (i.e., $\mathbf{X} = \mathbf{X}^*$), then the singular values of $\mathbf{X}$ are equal to the absolute value of the eigenvalues of $\mathbf{X}$.

This provides an intuitive interpretation of the SVD, where the columns of $\mathbf{U}$ are eigenvectors of the correlation matrix $\mathbf{X}\mathbf{X}^*$, and the columns of $\mathbf{V}$ are eigenvectors of $\mathbf{X}^*\mathbf{X}$. We choose to arrange the singular values in descending order by magnitude, and thus the columns of $\mathbf{U}$ are hierarchically ordered by how much correlation they capture in the columns of $\mathbf{X}$; similarly $\mathbf{V}$ captures correlation in the rows of $\mathbf{X}$.

Method of Snapshots

It is often impractical to construct the matrix $\mathbf{X}\mathbf{X}^*$ because of the large size of the state dimension n, let alone solve the eigenvalue problem: if $\mathbf{x}$ has a million elements, then $\mathbf{X}\mathbf{X}^*$ has a trillion elements. In 1987, Sirovich observed that it is possible to bypass this large matrix and compute the first m columns of $\mathbf{U}$ using what is now known as the method of snapshots [663].

Instead of computing the eigendecomposition of $\mathbf{X}\mathbf{X}^*$ to obtain the left singular vectors $\mathbf{U}$, we only compute the eigendecomposition of $\mathbf{X}^*\mathbf{X}$, which is

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

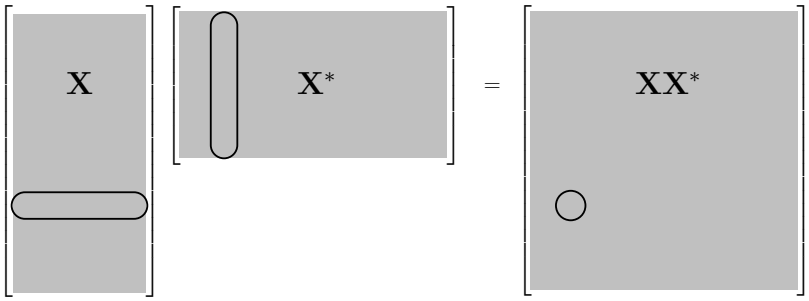


Figure 1.6: Correlation matrix $\mathbf{X}\mathbf{X}^*$ is formed by taking the inner product of rows of $\mathbf{X}$.

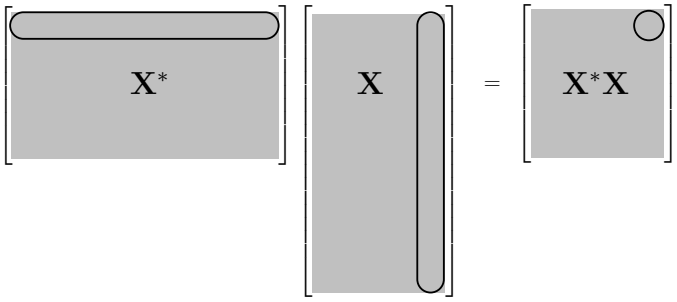


Figure 1.7: Correlation matrix $\mathbf{X}^*\mathbf{X}$ is formed by taking the inner product of columns of $\mathbf{X}$.

much smaller and more manageable. From (1.13b), we then obtain $\mathbf{V}$ and $\hat{\mathbf{\Sigma}}$. If there are zero singular values in $\hat{\mathbf{\Sigma}}$, then we only keep the r non-zero part, $\tilde{\mathbf{\Sigma}}$, and the corresponding columns $\tilde{\mathbf{V}}$ of $\mathbf{V}$. From these matrices, it is then possible to approximate $\tilde{\mathbf{U}}$, the first r columns of $\mathbf{U}$, as follows:

$$\tilde{\mathbf{U}} = \mathbf{X}\tilde{\mathbf{V}}\tilde{\mathbf{\Sigma}}^{-1}. \tag{1.14}$$

Generalization of the Eigendecomposition

In a sense, the singular value decomposition is a generalization of the eigendecomposition that is valid for all matrices, including non-square matrices and defective square matrices that do not have a complete basis of eigenvectors.

The eigendecomposition of a diagonalizable square matrix $\mathbf{X}$ is given by

$$\mathbf{X}\mathbf{V} = \mathbf{V}\mathbf{\Lambda} \quad \Longrightarrow \quad \mathbf{X} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^{-1}, \tag{1.15}$$

where the columns of $\mathbf{V}$ are eigenvectors and the corresponding entries of the diagonal matrix $\mathbf{\Lambda}$ are the eigenvalues. For Hermitian matrices (i.e., self-adjoint

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

matrices such that $\mathbf{X} = \mathbf{X}^*$), the eigendecomposition takes the form

$$\mathbf{XV} = \mathbf{V\Lambda} \quad \Longrightarrow \quad \mathbf{X} = \mathbf{V\Lambda V}^* \tag{1.16}$$

and the eigenvalues are real.

The singular value decomposition may be written in a similar form for a generic matrix $\mathbf{X}$ as

$$\mathbf{XV} = \mathbf{U\Sigma} \quad \Longrightarrow \quad \mathbf{X} = \mathbf{U\Sigma V}^*. \tag{1.17}$$

In this way, the singular value spectrum, given by the collection of singular values in $\mathbf{\Sigma}$, generalizes the notion of an eigenvalue spectrum, given by the collection of eigenvalues in $\mathbf{\Lambda}$. Similarly, the left and right singular vectors have an interpretation as a change of coordinates in the input space $\mathbb{C}^m$ and output space $\mathbb{C}^n$, much as the eigenvectors provide a change of coordinates to diagonalize a square matrix.

Geometric Interpretation

The columns of the matrix $\mathbf{U}$ provide an orthonormal basis for the column space of $\mathbf{X}$. Similarly, the columns of $\mathbf{V}$ provide an orthonormal basis for the row space of $\mathbf{X}$. If the columns of $\mathbf{X}$ are spatial measurements in time, then $\mathbf{U}$ encodes spatial patterns, and $\mathbf{V}$ encodes temporal patterns.

One property that makes the SVD particularly useful is the fact that both $\mathbf{U}$ and $\mathbf{V}$ are *unitary* matrices, so that $\mathbf{UU}^* = \mathbf{U}^*\mathbf{U} = \mathbf{I}_{n\times n}$ and $\mathbf{VV}^* = \mathbf{V}^*\mathbf{V} = \mathbf{I}_{m\times m}$. This means that solving a system of equations involving $\mathbf{U}$ or $\mathbf{V}$ is as simple as multiplication by the transpose, which scales as $\mathcal{O}(n^2)$, as opposed to traditional methods for the generic inverse, which scale as $\mathcal{O}(n^3)$. As noted in the previous section and in [79], the SVD is intimately connected to the spectral properties of the compact self-adjoint operators $\mathbf{XX}^*$ and $\mathbf{X}^*\mathbf{X}$.

The SVD of $\mathbf{X}$ may be interpreted geometrically based on how a hypersphere, given by $S^{n-1} \triangleq \{\mathbf{x} \,|\, \|\mathbf{x}\|_2 = 1\} \subset \mathbb{R}^n$, maps into an ellipsoid, $\{\mathbf{y} \,|\, \mathbf{y} = \mathbf{X}\mathbf{x} \text{ for } \mathbf{x} \in S^{n-1}\} \subset \mathbb{R}^m$, through $\mathbf{X}$. This is shown graphically in Fig. 1.8 for a sphere in $\mathbb{R}^3$ and a mapping $\mathbf{X}$ with three non-zero singular values. Because the mapping through $\mathbf{X}$ (i.e., matrix multiplication) is linear, knowing how it maps the unit sphere determines how all other vectors will map.

For the specific case shown in Fig. 1.8, we construct the matrix $\mathbf{X}$ out of three rotation matrices, $\mathbf{R}_x$, $\mathbf{R}_y$, and $\mathbf{R}_z$, and a fourth matrix to stretch out and

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

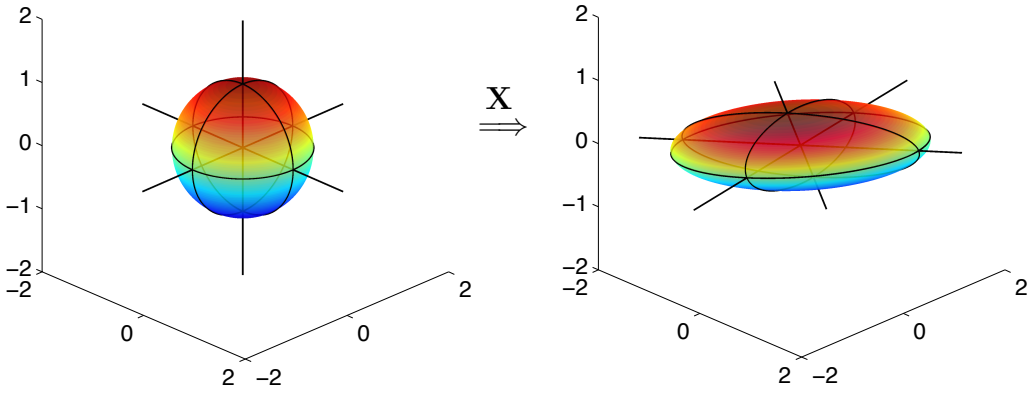


Figure 1.8: Geometric illustration of the SVD as a mapping from a sphere in $\mathbb{R}^n$ to an ellipsoid in $\mathbb{R}^m$.

scale the principal axes:

$$\mathbf{X} = \underbrace{\begin{bmatrix} \cos(\theta_3) & -\sin(\theta_3) & 0 \\ \sin(\theta_3) & \cos(\theta_3) & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\mathbf{R}_z} \underbrace{\begin{bmatrix} \cos(\theta_2) & 0 & \sin(\theta_2) \\ 0 & 1 & 0 \\ -\sin(\theta_2) & 0 & \cos(\theta_2) \end{bmatrix}}_{\mathbf{R}_y}$$
$$\times \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta_1) & -\sin(\theta_1) \\ 0 & \sin(\theta_1) & \cos(\theta_1) \end{bmatrix}}_{\mathbf{R}_x} \begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & \sigma_3 \end{bmatrix}.$$

In this case, $\theta_1 = \pi/15$, $\theta_2 = -\pi/9$, and $\theta_3 = -\pi/20$, and $\sigma_1 = 3$, $\sigma_2 = 1$, and $\sigma_3 = 0.5$. These rotation matrices do not commute, and so the order of rotation matters. If one of the singular values is zero, then a dimension is removed and the ellipsoid collapses onto a lower-dimensional subspace. The product $\mathbf{R}_x\mathbf{R}_y\mathbf{R}_z$ is the unitary matrix $\mathbf{U}$ in the SVD of $\mathbf{X}$. The matrix $\mathbf{V}$ is the identity. Codes to reproduce this example in MATLAB and in Python are provided on the book's GitHub.

Invariance of the SVD to Unitary Transformations

A useful property of the SVD is that if we left- or right-multiply our data matrix $\mathbf{X}$ by a unitary transformation, it preserves the terms in the SVD, except for the corresponding left or right unitary matrix $\mathbf{U}$ or $\mathbf{V}$, respectively. This has important implications, since the discrete Fourier transform (DFT; see Chapter 2) $\mathcal{F}$ is a unitary transform, meaning that the SVD of data $\hat{\mathbf{X}} = \mathcal{F}\mathbf{X}$ will be exactly the same as the SVD of $\mathbf{X}$, except that the modes $\hat{\mathbf{U}}$ will be the DFT of modes $\mathbf{U}$: $\hat{\mathbf{U}} = \mathcal{F}\mathbf{U}$. In addition, the invariance of the SVD to unitary transformations enables the use of compressed measurements to reconstruct SVD modes that are sparse in some transform basis (see Chapter 3).

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

The invariance of SVD to unitary transformations is geometrically intuitive, as unitary transformations rotate vectors in space, but do not change their inner products or correlation structures. We denote a left unitary transformation by $\mathbf{C}$, so that $\mathbf{Y} = \mathbf{CX}$, and a right unitary transformation by $\mathbf{P}^*$, so that $\mathbf{Y} = \mathbf{XP}^*$. The SVD of $\mathbf{X}$ will be denoted $\mathbf{U_X\Sigma_X V_X^*}$ and the SVD of $\mathbf{Y}$ will be $\mathbf{U_Y\Sigma_Y V_Y^*}$.

Left Unitary Transformations

First, consider a left unitary transformation of $\mathbf{X}$: $\mathbf{Y} = \mathbf{CX}$. Computing the correlation matrix $\mathbf{Y^*Y}$, we find

$$\mathbf{Y^*Y} = \mathbf{X^*C^*CX} = \mathbf{X^*X}. \tag{1.18}$$

The projected data has the same eigendecomposition, resulting in the same $\mathbf{V_X}$ and $\mathbf{\Sigma_X}$. Using the method of snapshots to reconstruct $\mathbf{U_Y}$, we find

$$\mathbf{U_Y} = \mathbf{YV_X\Sigma_X^{-1}} = \mathbf{CXV_X\Sigma_X^{-1}} = \mathbf{CU_X}. \tag{1.19}$$

Thus, $\mathbf{U_Y} = \mathbf{CU_X}$, $\mathbf{\Sigma_Y} = \mathbf{\Sigma_X}$, and $\mathbf{V_Y} = \mathbf{V_X}$. The SVD of $\mathbf{Y}$ is then

$$\mathbf{Y} = \mathbf{CX} = \mathbf{CU_X\Sigma_X V_X^*}. \tag{1.20}$$

Right Unitary Transformations

For a right unitary transformation $\mathbf{Y} = \mathbf{XP}^*$, the correlation matrix $\mathbf{Y^*Y}$ is

$$\mathbf{Y^*Y} = \mathbf{PX^*XP^*} = \mathbf{PV_X\Sigma_X^2 V_X^*P^*}, \tag{1.21}$$

with the following eigendecomposition:

$$\mathbf{Y^*YPV_X} = \mathbf{PV_X\Sigma_X^2}. \tag{1.22}$$

Thus, $\mathbf{V_Y} = \mathbf{PV_X}$ and $\mathbf{\Sigma_Y} = \mathbf{\Sigma_X}$. We may use the method of snapshots to reconstruct $\mathbf{U_Y}$:

$$\mathbf{U_Y} = \mathbf{YPV_X\Sigma_X^{-1}} = \mathbf{XV_X\Sigma_X^{-1}} = \mathbf{U_X}. \tag{1.23}$$

Thus, $\mathbf{U_Y} = \mathbf{U_X}$, and we may write the SVD of $\mathbf{Y}$ as

$$\mathbf{Y} = \mathbf{XP}^* = \mathbf{U_X\Sigma_X V_X^*P^*}. \tag{1.24}$$

1.4 Pseudo-Inverse, Least-Squares, and Regression

Many physical systems may be represented as a linear system of equations,

$$\mathbf{Ax} = \mathbf{b}, \tag{1.25}$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

where the constraint matrix $\mathbf{A}$ and vector $\mathbf{b}$ are known, and the vector $\mathbf{x}$ is unknown. If $\mathbf{A}$ is a square, invertible matrix (i.e., $\mathbf{A}$ has non-zero determinant), then there exists a unique solution $\mathbf{x}$ for every $\mathbf{b}$. However, when $\mathbf{A}$ is either singular or rectangular, there may be one, none, or infinitely many solutions, depending on the specific $\mathbf{b}$ and the column and row spaces of $\mathbf{A}$.

First, consider the *under-determined system*, where $\mathbf{A} \in \mathbb{C}^{n\times m}$ and $n \ll m$ (i.e., $\mathbf{A}$ is a short-fat matrix), so that there are fewer equations than unknowns. This type of system is likely to have columns that span all of $\mathbb{R}^n$, since it has many more columns than are required for a linearly independent basis.[5] Generically, if a short-fat $\mathbf{A}$ has n linearly independent columns (i.e., its column space spans $\mathbb{R}^n$), then there are infinitely many solutions $\mathbf{x}$ for every $\mathbf{b}$. The system is called *under-determined* because there are not enough values in $\mathbf{b}$ to uniquely determine the higher-dimensional $\mathbf{x}$.

Similarly, consider the *over-determined system*, where $n \gg m$ (i.e., a tall-skinny matrix), so that there are more equations than unknowns. This matrix cannot have n linearly independent columns, and so it is guaranteed that there are vectors $\mathbf{b}$ that have no solution $\mathbf{x}$. In fact, there will only be a solution $\mathbf{x}$ if $\mathbf{b}$ is in the column space of $\mathbf{A}$, i.e., $\mathbf{b} \in \text{col}(\mathbf{A})$.

Technically, there may be some choices of $\mathbf{b}$ that admit infinitely many solutions $\mathbf{x}$ for a tall-skinny matrix $\mathbf{A}$ and other choices of $\mathbf{b}$ that admit zero solutions even for a short-fat matrix. The solution space to the system in (1.25) is determined by the following four fundamental subspaces of $\mathbf{A} = \tilde{\mathbf{U}}\tilde{\mathbf{\Sigma}}\tilde{\mathbf{V}}^*$, where the rank r is chosen to include all non-zero singular values:

- The column space, $\text{col}(\mathbf{A})$, is the span of the columns of $\mathbf{A}$, also known as the *range*. The column space of $\mathbf{A}$ is the same as the column space of $\tilde{\mathbf{U}}$.
- The orthogonal complement to $\text{col}(\mathbf{A})$ is $\ker(\mathbf{A}^*)$, given by the column space of $\hat{\mathbf{U}}^\perp$ from Fig. 1.1.
- The row space, $\text{row}(\mathbf{A})$, is the span of the rows of $\mathbf{A}$, which is spanned by the columns of $\tilde{\mathbf{V}}$. The row space of $\mathbf{A}$ is equal to $\text{row}(\mathbf{A}) = \text{col}(\mathbf{A}^*)$.
- The kernel space, $\ker(\mathbf{A})$, is the orthogonal complement to $\text{row}(\mathbf{A})$, and is also known as the *null space*. The null space is the subspace of vectors that map through $\mathbf{A}$ to zero, i.e., $\mathbf{A}\mathbf{x} = \mathbf{0}$, given by $\text{col}(\hat{\mathbf{V}}^\perp)$.

More precisely, if $\mathbf{b} \in \text{col}(\mathbf{A})$ and if $\dim(\ker(\mathbf{A})) \neq 0$, then there are infinitely many solutions $\mathbf{x}$. Note that the condition $\dim(\ker(\mathbf{A})) \neq 0$ is guaranteed for

[5] It is easy to construct degenerate examples where the columns of a short-fat matrix do not form a full basis for $\mathbb{R}^n$, such as

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}.$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

a short-fat matrix. Similarly, if $\mathbf{b} \notin \text{col}(\mathbf{A})$, then there are no solutions, and the system of equations in (1.25) is called *inconsistent*.

The fundamental subspaces above satisfy the following properties:

$$\text{col}(\mathbf{A}) \oplus \ker(\mathbf{A}^*) = \mathbb{R}^n, \tag{1.26a}$$
$$\text{col}(\mathbf{A}^*) \oplus \ker(\mathbf{A}) = \mathbb{R}^n. \tag{1.26b}$$

Remark 1.1 *There is an extensive literature on random matrix theory, where the above stereotypes are almost certainly true, meaning that they are true with high probability. For example, a system* $\mathbf{Ax} = \mathbf{b}$ *is extremely unlikely to have a solution for a random matrix* $\mathbf{A} \in \mathbb{R}^{n \times m}$ *and random vector* $\mathbf{b} \in \mathbb{R}^n$ *with* $n \gg m$*, since there is little chance that* $\mathbf{b}$ *is in the column space of* $\mathbf{A}$*. These properties of random matrices will play a prominent role in compressed sensing (see Chapter 3).*

In the over-determined case when no solution exists, we would often like to find the solution $\mathbf{x}$ that minimizes the sum-squared error $\|\mathbf{Ax} - \mathbf{b}\|_2^2$, the so-called *least-squares* solution. Note that the least-squares solution also minimizes $\|\mathbf{Ax} - \mathbf{b}\|_2$. In the under-determined case when infinitely many solutions exist, we may like to find the solution $\mathbf{x}$ with minimum norm $\|\mathbf{x}\|_2$ so that $\mathbf{Ax} = \mathbf{b}$, the so-called *minimum-norm* solution.

The SVD is the technique of choice for these important optimization problems. First, if we substitute an exact truncated SVD $\mathbf{A} = \tilde{\mathbf{U}}\tilde{\mathbf{\Sigma}}\tilde{\mathbf{V}}^*$ in for $\mathbf{A}$, we can "invert" each of the matrices $\tilde{\mathbf{U}}$, $\tilde{\mathbf{\Sigma}}$, and $\tilde{\mathbf{V}}^*$ in turn, resulting in the Moore–Penrose *left pseudo-inverse* [560, 561, 604, 776] $\mathbf{A}^\dagger$ of $\mathbf{A}$:

$$\mathbf{A}^\dagger \triangleq \tilde{\mathbf{V}}\tilde{\mathbf{\Sigma}}^{-1}\tilde{\mathbf{U}}^* \quad \Longrightarrow \quad \mathbf{A}^\dagger\mathbf{A} = \tilde{\mathbf{V}}\tilde{\mathbf{V}}^*. \tag{1.27}$$

Note that $\mathbf{A}^\dagger\mathbf{A}$ will only equal the identity $\mathbf{I}_{m\times m}$ if the truncated SVD captures all non-zero singular values; otherwise $\tilde{\mathbf{V}}\tilde{\mathbf{V}}^* \neq \mathbf{I}_{m\times m}$, and it will only approximate the identity. This may be used to find both the minimum-norm and least-squares solutions to (1.25):

$$\mathbf{A}^\dagger\mathbf{A}\tilde{\mathbf{x}} = \mathbf{A}^\dagger\mathbf{b} \quad \Longrightarrow \quad \tilde{\mathbf{x}} = \tilde{\mathbf{V}}\tilde{\mathbf{\Sigma}}^{-1}\tilde{\mathbf{U}}^*\mathbf{b}. \tag{1.28}$$

Plugging the solution $\tilde{\mathbf{x}}$ back in to (1.25) results in

$$\mathbf{A}\tilde{\mathbf{x}} = \tilde{\mathbf{U}}\tilde{\mathbf{\Sigma}}\tilde{\mathbf{V}}^*\tilde{\mathbf{V}}\tilde{\mathbf{\Sigma}}^{-1}\tilde{\mathbf{U}}^*\mathbf{b} \tag{1.29a}$$
$$= \tilde{\mathbf{U}}\tilde{\mathbf{U}}^*\mathbf{b}. \tag{1.29b}$$

Although $\mathbf{U}^*\mathbf{U} = \mathbf{U}\mathbf{U}^* = \mathbf{I}_{n\times n}$ for the exact SVD, $\tilde{\mathbf{U}}\tilde{\mathbf{U}}^*$ is not necessarily the identity matrix for a truncated basis of left singular vectors $\tilde{\mathbf{U}}$, but is rather a projection onto the column space of $\tilde{\mathbf{U}}$. Therefore, $\tilde{\mathbf{x}}$ will only be an exact solution to (1.25) when $\mathbf{b}$ is in the column space of $\tilde{\mathbf{U}}$, and therefore in the column

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

space of $\mathbf{A}$. Assuming that $\tilde{\mathbf{U}}\tilde{\mathbf{U}}^*$ is equal to the identity is one of the most common accidental misuses of the SVD.[6] However, it is still true that $\tilde{\mathbf{U}}^*\tilde{\mathbf{U}} = \mathbf{I}_{r\times r}$, where r is the rank of $\mathbf{A}$.

Computing the pseudo-inverse $\mathbf{A}^\dagger$ is computationally efficient, after the expensive up-front cost of computing the SVD. Inverting the unitary matrices $\tilde{\mathbf{U}}$ and $\tilde{\mathbf{V}}$ involves matrix multiplications by the transpose matrices, which are $\mathcal{O}(n^2)$ operations. Inverting $\tilde{\mathbf{\Sigma}}$ is even more efficient, since it is a diagonal matrix, requiring $\mathcal{O}(n)$ operations. In contrast, inverting a dense square matrix would require an $\mathcal{O}(n^3)$ operation.

Condition Number

The condition number of a matrix $\mathbf{A}$ is a measure of how sensitive matrix multiplication and inversion are to errors in the input. Larger condition number indicates higher sensitivity and worse performance. The condition number $\kappa(\mathbf{A})$ is directly related to the singular values of the matrix:

$$\kappa(\mathbf{A}) = \frac{\sigma_{\text{max}}(\mathbf{A})}{\sigma_{\text{min}}(\mathbf{A})}. \tag{1.30}$$

The condition number is a central concept in all of numerical linear algebra and applied computation. It is easiest to understand the effect of a large condition number when considering the linear system of equations $\mathbf{Ax} = \mathbf{b}$. If the vector $\mathbf{x}$ is not specified perfectly, but instead has some error $\boldsymbol{\epsilon}_\mathbf{x}$, then the system becomes

$$\mathbf{A}(\mathbf{x} + \boldsymbol{\epsilon}_\mathbf{x}) = \mathbf{b} + \boldsymbol{\epsilon}_\mathbf{b}, \tag{1.31}$$

where $\boldsymbol{\epsilon}_\mathbf{b}$ is the corresponding error in $\mathbf{b}$. If we assume a worst-case scenario, where $\boldsymbol{\epsilon}_\mathbf{x}$ is aligned with the singular vector corresponding to the maximum singular value σ_{max}, and the vector $\mathbf{x}$ is aligned with the singular vector corresponding to the minimum singular value σ_{min}, then the output is

$$\mathbf{A}(\mathbf{x} + \boldsymbol{\epsilon}_\mathbf{x}) = \underbrace{\sigma_{\text{min}}\mathbf{x}}_{\mathbf{b}} + \underbrace{\sigma_{\text{max}}\boldsymbol{\epsilon}_\mathbf{x}}_{\boldsymbol{\epsilon}_\mathbf{b}}. \tag{1.32}$$

The output signal-to-noise $\|\mathbf{b}\|/\|\boldsymbol{\epsilon}_\mathbf{b}\|$ is equal to the input signal-to-noise $\|\mathbf{x}\|/\|\boldsymbol{\epsilon}_\mathbf{x}\|$ multiplied by a factor of $\sigma_{\text{min}}/\sigma_{\text{max}}$. In other words, the signal-to-noise of the output has been reduced by a factor equal to the condition number $\kappa(\mathbf{A})$. Even if the vectors $\mathbf{x}$ and $\boldsymbol{\epsilon}_\mathbf{x}$ are not perfectly aligned with the worst-case directions, they are likely to have some component of all of the singular vector directions.

[6] The authors are not immune to this, having mistakenly used this fictional identity in an early version of [134].

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

In this case, components of the error will still experience large amplification relative to other components of the desired output.

A similar issue arises when solving for $\mathbf{x}$ given an imperfectly specified $\mathbf{b}$ with some error $\boldsymbol{\epsilon}_\mathbf{b}$. Now the worst-case scenario is where $\boldsymbol{\epsilon}_\mathbf{b}$ is aligned with the singular vector corresponding to the minimum singular value σ_{min}, and the vector $\mathbf{b}$ is aligned with the singular vector corresponding to the maximum singular value σ_{max}. Then the estimated solution for $\mathbf{x} + \boldsymbol{\epsilon}_\mathbf{x}$ is

$$\mathbf{x} + \boldsymbol{\epsilon}_\mathbf{x} \approx \mathbf{A}^\dagger(\mathbf{b} + \boldsymbol{\epsilon}_\mathbf{b}) = \frac{1}{\sigma_{\text{max}}}\mathbf{b} + \frac{1}{\sigma_{\text{min}}}\boldsymbol{\epsilon}_\mathbf{b}. \tag{1.33}$$

The signal-to-noise of the estimated $\mathbf{x}$ has also been reduced, or degraded, by a factor equal to the condition number $\kappa(\mathbf{A})$.

One approach to mitigate a large condition number is to truncate the SVD more aggressively, essentially increasing the effective minimum singular value σ_{min}. However, this comes at the cost of decreasing the size of the subspace $\tilde{\mathbf{U}}$ used to approximate the output.

```
>> kappanew = 1.e-5;  % Desired condition number
>> [U,S,V] = svd(A,'econ')
>> r = max(find(diag(S)>max(S(:))*kappanew));
>> invA = V(:,1:r)*inv(S(1:r,1:r))*U(:,1:r)'; % Approximate
```

One-Dimensional Linear Regression

Regression is an important statistical tool to relate variables to one another based on data [477]. Consider the collection of data in Fig. 1.9. The red crosses are obtained by adding Gaussian white noise to the black line, as shown in Code 1.2. We assume that the data is linearly related, as in (1.25), and we use the pseudo-inverse to find the least-squares solution for the slope x below (blue dashed line), shown in Code 1.2:

$$\begin{bmatrix} | \\ \mathbf{b} \\ | \end{bmatrix} = \begin{bmatrix} | \\ \mathbf{a} \\ | \end{bmatrix} x = \tilde{\mathbf{U}}\tilde{\mathbf{\Sigma}}\tilde{\mathbf{V}}^* x. \tag{1.34a}$$

$$\implies \quad x = \tilde{\mathbf{V}}\tilde{\mathbf{\Sigma}}^{-1}\tilde{\mathbf{U}}^*\mathbf{b}. \tag{1.34b}$$

In (1.34b), $\tilde{\mathbf{\Sigma}} = \|\mathbf{a}\|_2$, $\tilde{\mathbf{V}} = 1$, and $\tilde{\mathbf{U}} = \mathbf{a}/\|\mathbf{a}\|_2$. Taking the left pseudo-inverse:

$$x = \frac{\mathbf{a}^*\mathbf{b}}{\|\mathbf{a}\|_2^2}. \tag{1.35}$$

This makes physical sense, if we think of x as the value that best maps our vector $\mathbf{a}$ to the vector $\mathbf{b}$. Then, the best single value x is obtained by taking

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

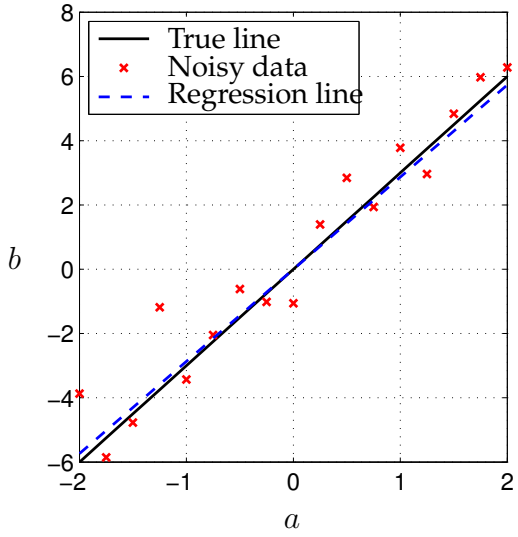


Figure 1.9: Illustration of linear regression using noisy data.

the dot product of $\mathbf{b}$ with the normalized $\mathbf{a}$ direction. We then add a second normalization factor $\|\mathbf{a}\|_2$ because the $\mathbf{a}$ in (1.34a) is not normalized.

Note that strange things happen if you use row vectors instead of column vectors in (1.34) above. Also, if the noise magnitude becomes large relative to the slope x, the pseudo-inverse will undergo a phase change in accuracy, related to the hard-thresholding results in subsequent sections.

Code 1.2: [MATLAB] Least-squares fit of noisy data in Fig. 1.9.

```
% Generate noisy data
x = 3;                              % True slope
a = [-2:.25:2]';
b = a*x + 1*randn(size(a));         % Add noise
plot(a,x*a,'k')                     % True relationship
hold on, plot(a,b,'rx')             % Noisy measurements

% Compute least-squares approximation with the SVD
[U,S,V] = svd(a,'econ');
xtilde = V*inv(S)*U'*b;             % Least-square fit
plot(a,xtilde*a,'b--')              % Plot fit

%% Alternative formulations of least-squares
xtilde1 = V*inv(S)*U'*b
xtilde2 = pinv(a)*b
xtilde3 = regress(b,a)
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Code 1.2: [Python] Least-squares fit of noisy data in Fig. 1.9.

```
x = 3 # True slope
a = np.arange(-2,2,0.25)
a = a.reshape(-1, 1)
b = x*a + np.random.randn(*a.shape) # Add noise
plt.plot(a, x*a, Color='k', LineWidth=2, label='True line')
    # True relationship
plt.plot(a, b, 'x', Color='r', MarkerSize = 10, label='Noisy
    data') # Noisy measurements

# Compute least-squares approximation with the SVD
U, S, VT = np.linalg.svd(a,full_matrices=False)
xtilde = VT.T @ np.linalg.inv(np.diag(S)) @ U.T @ b # Least-
    square fit
plt.plot(a,xtilde * a,'--',Color='b',LineWidth=4, label='
    Regression line')

# Alternative formulations of least squares
xtilde1 = VT.T @ np.linalg.inv(np.diag(S)) @ U.T @ b
xtilde2 = np.linalg.pinv(a) @ b
```

Multi-linear Regression

Example 1: Cement Heat Generation Data

First, we begin with a simple built-in MATLAB data set that describes the heat generation for various cement mixtures that comprise four basic ingredients (see Fig. 1.10). In this problem, we are solving (1.25), where $\mathbf{A} \in \mathbb{R}^{13\times 4}$, since there are four ingredients and heat measurements for 13 unique mixtures. The goal is to determine the weighting $\mathbf{x}$ that relates the proportions of the four ingredients to the heat generation. It is possible to find the minimum error solution using the SVD, as shown in Code 1.3. Alternatives, using **regress** and **pinv**, are also explored.

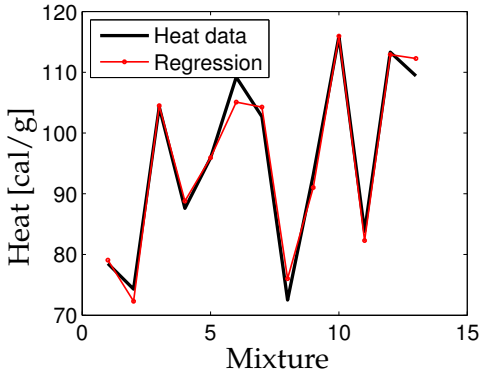
Code 1.3: [MATLAB] Multi-linear regression for cement heat data.

```
load hald;  % Load Portlant Cement dataset
A = ingredients;
b = heat;

[U,S,V] = svd(A,'econ');
x = V*inv(S)*U'*b;          % Solve Ax=b using the SVD

plot(b,'k');  hold on       % Plot data
plot(A*x,'r-o');           % Plot fit
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.



	Ingredient	Regression
1	Tricalcium aluminate	2.1930
2	Tricalcium silicate	1.1533
3	Tetracalcium alumiferrite	0.7585
4	Beta-dicalcium silicate	0.4863

Figure 1.10: Heat data for cement mixtures containing four basic ingredients.

```
x = regress(b,A);           % Alternative 1  (regress)
x = pinv(A)*b;              % Alternative 2  (pinv)
```

Code 1.3: [Python] Multi-linear regression for cement heat data.

```
# Load dataset
A = np.loadtxt(os.path.join('..','DATA','hald_ingredients.
    csv'),delimiter=',')
b = np.loadtxt(os.path.join('..','DATA','hald_heat.csv'),
    delimiter=',')

# Solve Ax=b using SVD
U, S, VT = np.linalg.svd(A,full_matrices=0)
x = VT.T @ np.linalg.inv(np.diag(S)) @ U.T @ b

plt.plot(b, Color='k', LineWidth=2, label='Heat Data')
plt.plot(A@x, '-o', Color='r', label='Regression')

x = np.linalg.pinv(A)*b    # Alternative
```

Example 2: Boston Housing Data

In this example, we explore a larger data set to determine which factors best predict prices in the Boston housing market [313]. This data is available from the UCI Machine Learning Repository [33].

There are 13 attributes that are correlated with house price, such as per-capita crime rate and property-tax rate. These features are regressed onto the price data, the best-fit price prediction is plotted against the true house value in Fig. 1.11, and the regression coefficients are shown in Fig. 1.12. Although the house value is not perfectly predicted, the trend agrees quite well. It is often the case that the highest value outliers are not well captured by simple linear fits,

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

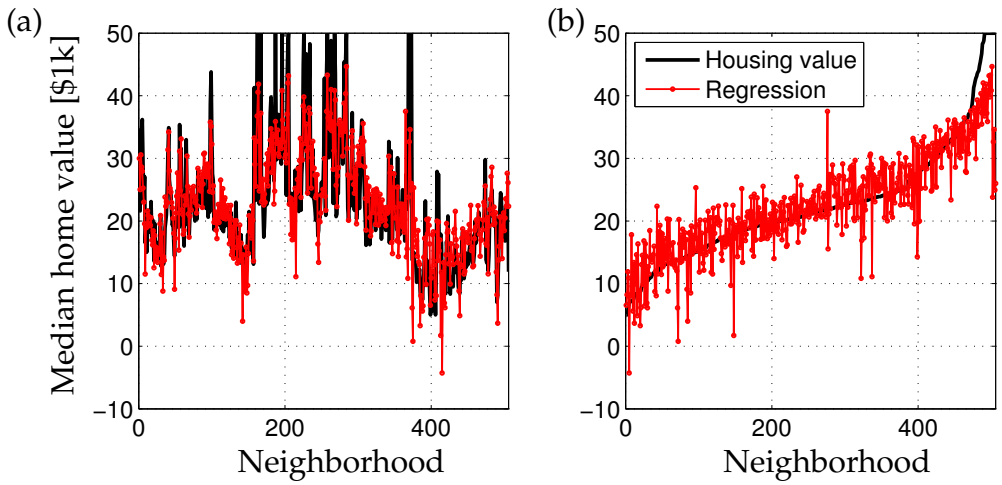


Figure 1.11: Multi-linear regression of home prices using various factors: (a) unsorted data and (b) data sorted by home value.

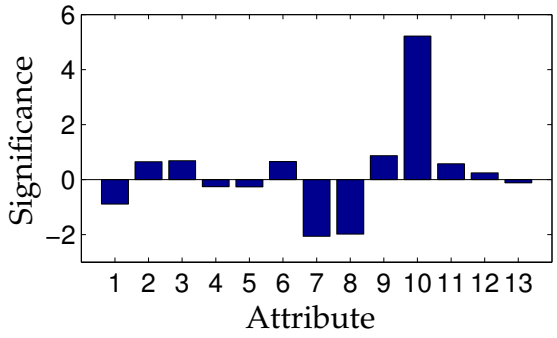


Figure 1.12: Significance of various attributes in the regression.

as in this example.

This data contains prices and attributes for 506 homes, so the attribute matrix is of size 506×13. It is important to pad this matrix with an additional column of ones, to take into account the possibility of a non-zero constant offset in the regression formula. This corresponds to the "y intercept" in a simple one-dimensional linear regression. The code for this example is nearly identical to the example above, and is available on the book's GitHub.

1.5 Principal Component Analysis (PCA)

Principal component analysis (PCA) is one of the central applications of the SVD, providing a statistical interpretation of the data-driven, hierarchical coordinate system used to represent high-dimensional correlated data. This coordinate system involves the correlation matrices described in Section 1.3. Impor-

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

tantly, PCA pre-processes the data by mean subtraction and setting the variance to unity before performing the SVD. The geometry of the resulting coordinate system is determined by principal components (PCs) that are uncorrelated (orthogonal) to each other, but have maximal correlation with the measurements. This theory was developed in 1901 by Pearson [552], and independently by Hotelling in the 1930s [339, 340]. Jolliffe [352] provides a good reference text.

Often in statistics, a number of measurements are collected in a single experiment, and these measurements are typically arranged into a row vector. The measurements may be features of an observable, such as demographic features of a specific human individual. A number of experiments are conducted, and each measurement vector is arranged as a row in a large matrix $\mathbf{X}$, resembling the structure of how data is recorded in a spreadsheet. In the example of demography, the collection of experiments may be gathered via polling. Note that this convention for $\mathbf{X}$, consisting of rows of features, is different than the convention throughout the remainder of this chapter, where individual feature "snapshots" are arranged as columns. However, we choose to be consistent with PCA literature in this section. The matrix will still be size $n \times m$, although it may have more rows than columns, or vice versa.

Computation

We now compute the average row $\bar{\mathbf{x}}$ (i.e., the mean of all rows), and subtract it from $\mathbf{X}$. The mean $\bar{\mathbf{x}}$ is given by

$$\bar{\mathbf{x}}_j = \frac{1}{n}\sum_{i=1}^{n} \mathbf{X}_{ij}, \tag{1.36}$$

and the mean matrix is

$$\bar{\mathbf{X}} = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \bar{\mathbf{x}}. \tag{1.37}$$

Subtracting $\bar{\mathbf{X}}$ from $\mathbf{X}$ results in the mean-subtracted data $\mathbf{B}$:

$$\mathbf{B} = \mathbf{X} - \bar{\mathbf{X}}. \tag{1.38}$$

The covariance matrix of $\mathbf{B}$ is given by

$$\mathbf{C} = \frac{1}{n-1}\mathbf{B}^*\mathbf{B}. \tag{1.39}$$

Note that the covariance is normalized by $n-1$ instead of n, even though there are n sample points. This is known as Bessel's correction, which compensates

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

for the fact that the sample variance is biased because it does not capture the variance of the sample mean $\bar{\mathbf{X}}$ about the true mean. The covariance matrix $\mathbf{C}$ is symmetric and positive semi-definite, having non-negative real eigenvalues. Each entry C_{ij} quantifies the correlation of the i and j features across all experiments.

The principal components are the eigenvectors of $\mathbf{C}$, and they define a change of coordinates in which the covariance matrix is diagonal:

$$\mathbf{CV} = \mathbf{VD} \quad \Longrightarrow \quad \mathbf{C} = \mathbf{VDV}^* \quad \Longrightarrow \quad \mathbf{D} = \mathbf{V}^*\mathbf{CV}. \tag{1.40}$$

The columns of the eigenvector matrix $\mathbf{V}$ are the principal components, and the elements of the diagonal matrix $\mathbf{D}$ are the variances of the data along these directions. This transformation is guaranteed to exist, since $\mathbf{C}$ is Hermitian and the columns of $\mathbf{V}$ are orthonormal. In these principal component coordinates, all features are linearly uncorrelated with each other.

The matrix of principal components $\mathbf{V}$ is also the matrix of right singular vectors of $\mathbf{B}$. Substituting $\mathbf{B} = \mathbf{U\Sigma V}^*$ into (1.39) and comparing with (1.40) yields

$$\mathbf{C} = \frac{1}{n-1}\mathbf{B}^*\mathbf{B} = \frac{1}{n-1}\mathbf{V\Sigma}^2\mathbf{V}^* \quad \Longrightarrow \quad \mathbf{D} = \frac{1}{n-1}\mathbf{\Sigma}^2. \tag{1.41}$$

The variance of the data in these coordinates, given by the diagonal elements λ_k of $\mathbf{D}$, is related to the singular values as

$$\lambda_k = \frac{\sigma_k^2}{n-1}. \tag{1.42}$$

Thus, the SVD provides a numerically robust approach for computing the principal components. An approximation $\tilde{\mathbf{B}}$ obtained by keeping only the first r principal components will have a missing variance related to the squared Frobenius norm error in (1.7).

The pca Command

In MATLAB, there the additional commands **pca** and **princomp** (based on **pca**) for the principal component analysis:

```
>> [V,score,s2] = pca(X);
```

The matrix $\mathbf{V}$ is equivalent to the $\mathbf{V}$ matrix from the SVD of $\mathbf{B} = \mathbf{X} - \bar{\mathbf{X}}$, up to sign changes of the columns. The vector **s2** contains eigenvalues of the covariance of $\mathbf{B}$, also known as principal component variances; these values are the squares of the singular values. The variable **score** simply contains the coordinates of each row of $\mathbf{B}$ (the mean-subtracted data) in the principal component directions. In general, we often prefer to use the **svd** command with the various pre-processing steps described above.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Table 1.1: Standard deviation of data and normalized singular values.

	σ_1	σ_2
Data	2	0.5
SVD	1.974	0.503

Example: Noisy Gaussian Data

Consider the noisy cloud of data in Fig. 1.13(a), generated using Code 1.4. The data is generated by selecting $10\,000$ vectors from a two-dimensional normal distribution with zero mean and unit variance. These vectors are then scaled in the x and y directions by the values in Table 1.1 and rotated by $\pi/3$. Finally, the entire cloud of data is translated so that it has a non-zero center $\mathbf{x}_C = \begin{bmatrix} 2 & 1 \end{bmatrix}^T$.

Using Code 1.4, the PCA is performed and used to plot confidence intervals using multiple standard deviations, shown in Fig. 1.13(b). The singular values, shown in Table 1.1, match the data scaling. The matrix $\mathbf{U}$ from the SVD also closely matches the rotation matrix, up to a sign on the rows:

$$\mathbf{R}_{\pi/3} = \begin{bmatrix} 0.5 & 0.8660 \\ -0.8660 & 0.5 \end{bmatrix}, \qquad \mathbf{U} = \begin{bmatrix} -0.4998 & -0.8662 \\ -0.8662 & 0.4998 \end{bmatrix}.$$

Note that $\mathbf{R}_{\pi/3}$ is a rotation matrix designed to rotate row vectors by multiplication on the right by $\mathbf{R}_{\pi/3}$. For rotation of column vectors by left multiplication, this matrix would be transposed.

Code 1.4: [MATLAB] PCA example on noisy cloud of data.

```
% Generate noisy cloud of data
xC = [2, 1];                            % Center of data (mean)
sig = [2, .5];                          % Principal axes

theta = pi/3;                           % Rotate cloud by pi/3
R = [cos(theta) sin(theta);             % Rotation matrix
    -sin(theta) cos(theta)];

nPoints = 10000;                        % Create 10,000 points
X = randn(nPoints,2)*diag(sig)*R + ones(nPoints,2)*diag(xC);
scatter(X(:,1),X(:,2),'k.','LineWidth',2)  % Plot data

% Compute PCA and plot confidence intervals
Xavg = mean(X,1);                         % Compute mean
B = X - ones(nPoints,1)*Xavg;             % Mean-subtracted Data
[U,S,V] = svd(B/sqrt(nPoints),'econ'); % PCA via SVD
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

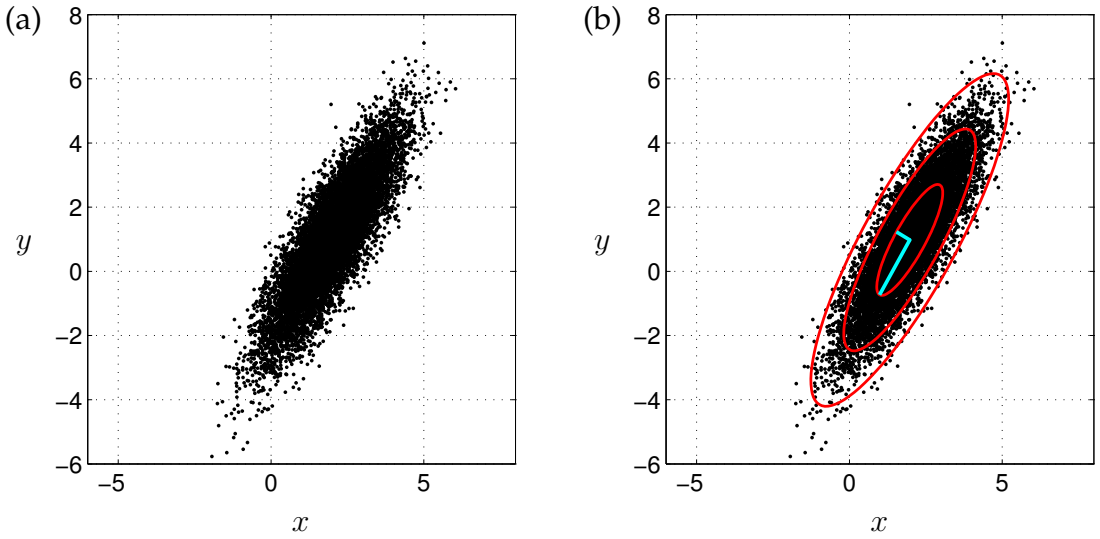


Figure 1.13: (a) Principal components capture the variance of mean-subtracted Gaussian data. (b) The first three standard deviation ellipsoids (red), and the two left singular vectors, scaled by singular values ($\sigma_1 v_1 + x_C$ and $\sigma_2 v_2 + x_C$, cyan), are shown.

```
theta = (0:.01:1)*2*pi;
Xstd = [cos(theta') sin(theta')]*S*V'; % 1std conf. interval
hold on, plot(Xavg(1)+Xstd(:,1),Xavg(2) + Xstd(:,2),'r-')
plot(Xavg(1)+2*Xstd(:,1),Xavg(2) + 2*Xstd(:,2),'r-')
plot(Xavg(1)+3*Xstd(:,1),Xavg(2) + 3*Xstd(:,2),'r-')
```

Code 1.4: [Python] PCA example on noisy cloud of data.

```
# Generate noisy cloud of data
xC = np.array([2, 1])       # Center of data (mean)
sig = np.array([2, 0.5])    # Principal axes

theta = np.pi/3             # Rotate cloud by pi/3
R = np.array([[np.cos(theta),-np.sin(theta)], # Rotation mat
              [np.sin(theta),np.cos(theta)]])

nPoints = 10000             # Create 10,000 points
X = R @ np.diag(sig) @ np.random.randn(2,nPoints) + np.diag(
   xC) @ np.ones((2,nPoints))
ax1.plot(X[0,:],X[1,:], '.', Color='k') # Plot data
Xavg = np.mean(X,axis=1)                # Compute mean
B = X - np.tile(Xavg,(nPoints,1)).T  # Mean-subtracted data
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

```
# Find principal components (SVD)
U, S, VT = np.linalg.svd(B/np.sqrt(nPoints),full_matrices=0)
theta = 2 * np.pi * np.arange(0,1,0.01)
Xstd = U @ np.diag(S) @ np.array([np.cos(theta),np.sin(theta
    )])
ax2.plot(Xavg[0] + Xstd[0,:], Xavg[1] + Xstd[1,:],'-',color=
    'r',LineWidth=3)
ax2.plot(Xavg[0] + 2*Xstd[0,:], Xavg[1] + 2*Xstd[1,:],'-',
    color='r',LineWidth=3)
ax2.plot(Xavg[0] + 3*Xstd[0,:], Xavg[1] + 3*Xstd[1,:],'-',
    color='r',LineWidth=3)
```

Finally, it is also possible to compute using the **pca** command in MATLAB:

```
>> [V,score,s2] = pca(X);
>> norm(score*V - B)

ans =
    1.4900e-13
```

Example: Ovarian Cancer Data

The ovarian cancer data set, which is built into MATLAB, provides a more realistic example to illustrate the benefits of PCA. This example consists of gene data for 216 patients, 121 of whom have ovarian cancer, and 95 of whom do not. For each patient, there is a vector of data containing the expression of 4000 genes. There are multiple challenges with this type of data, namely the high dimension of the data features. However, we see from Fig. 1.14 that there is significant variance captured in the first few PCA modes. Said another way, the gene data is highly correlated, so that many patients have significant overlap in their gene expression. The ability to visualize patterns and correlations in high-dimensional data is an important reason to use PCA, and PCA has been widely used to find patterns in high-dimensional biological and genetic data [588].

More importantly, patients with ovarian cancer appear to cluster separately from patients without cancer when plotted in the space spanned by the first three PCA modes. This is shown in Fig. 1.15, which is generated by Code 1.5. This inherent clustering in PCA space of data by category is a foundational element of machine learning and pattern recognition. For example, we will see in Section 1.6 that images of different human faces will form clusters in PCA space. The use of these clusters will be explored in greater detail in Chapter 5.

Code 1.5: [MATLAB] Compute PCA for ovarian cancer data.

```
load ovariancancer;  % Load ovarian cancer data
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

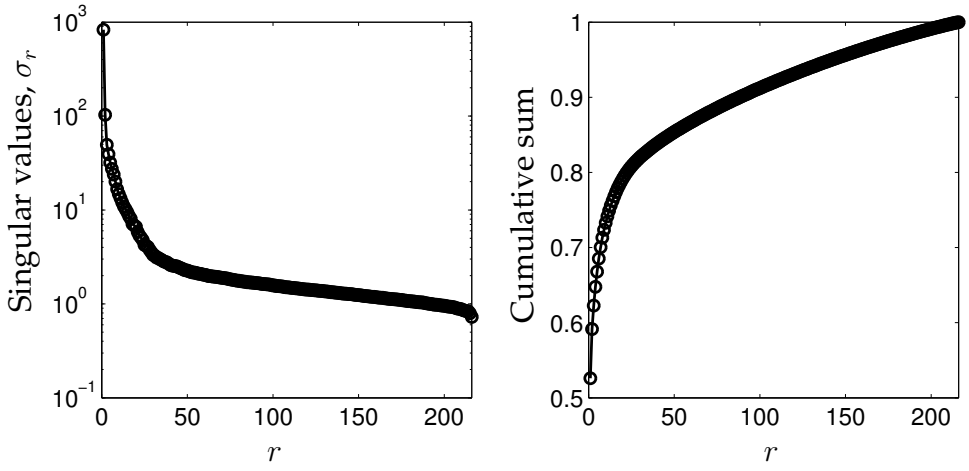


Figure 1.14: Singular values for the ovarian cancer data.

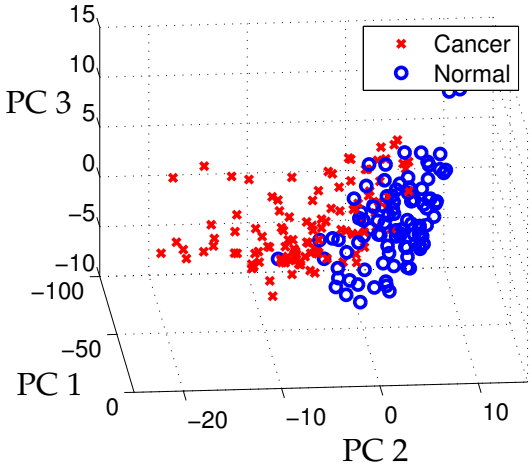


Figure 1.15: Clustering of samples that are normal and those that have cancer in the first three principal component coordinates.

```
[U,S,V] = svd(obs,'econ');
for i=1:size(obs,1)
    x = V(:,1)'*obs(i,:)';
    y = V(:,2)'*obs(i,:)';
    z = V(:,3)'*obs(i,:)';
    if(grp{i}=='Cancer')
        plot3(x,y,z,'rx','LineWidth',2);
    else
        plot3(x,y,z,'bo','LineWidth',2);
    end
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

```
end
```

Code 1.5: [Python] Compute PCA for ovarian cancer data.

```
obs = np.loadtxt(os.path.join('..','DATA','ovariancancer_obs
    .csv'),delimiter=',')
f = open(os.path.join('..','DATA','ovariancancer_grp.csv'),
    "r")
grp = f.read().split("\n")

U, S, VT = np.linalg.svd(obs,full_matrices=0)
for j in range(obs.shape[0]):
    x = VT[0,:] @ obs[j,:].T
    y = VT[1,:] @ obs[j,:].T
    z = VT[2,:] @ obs[j,:].T

    if grp[j] == 'Cancer':
        ax.scatter(x,y,z,marker='x',color='r',s=50)
    else:
        ax.scatter(x,y,z,marker='o',color='b',s=50)
```

1.6 Eigenfaces Example

One of the most striking demonstrations of SVD/PCA is the so-called eigenfaces example. In this problem, PCA (i.e., SVD on mean-subtracted data) is applied to a large library of facial images to extract the most dominant correlations between images. The result of this decomposition is a set of *eigenfaces* that define a new coordinate system. Images may be represented in these coordinates by taking the dot product with each of the principal components. It will be shown in Chapter 5 that images of the same person tend to cluster in the eigenface space, making this a useful transformation for facial recognition and classification [67, 687]. The eigenface problem was first studied by Sirovich and Kirby in 1987 [664] and expanded on in [388]. Its application to automated facial recognition was presented by Turk and Pentland in 1991 [728].

Here, we demonstrate this algorithm using the Extended Yale Face Database B [274], consisting of cropped and aligned images [435] of 38 individuals (28 from the extended database, and 10 from the original database) under nine poses and 64 lighting conditions.[7] Each image is 192 pixels tall and 168 pixels wide. Unlike the previous image example in Section 1.2, each of the facial images in our library has been reshaped into a large column vector with $192 \times 168 = 32\,256$ elements. We use the first 36 people in the database (left

[7]The Yale database can be downloaded at `http://vision.ucsd.edu/~iskwak/ExtYaleDatabase/ExtYaleB.html`.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.



Figure 1.16: (left) A single image for each person in the Yale database, and (right) all images for a specific person. Left panel generated by Code 1.6.

panel of Fig. 1.16) as our training data for the eigenfaces example, and we hold back two people as a test set. An example of all 64 images of one specific person are shown in the right panel. These images are loaded and plotted using Code 1.6.

Code 1.6: [MATLAB] Plot image for each person in the Yale database (Fig. 1.16).

```
load ../DATA/allFaces.mat

allPersons = zeros(n*6,m*6); % Make array to fit all faces
count = 1;
for i=1:6          % 6 x 6 grid of faces
    for j=1:6
        allPersons(1+(i-1)*n:i*n,1+(j-1)*m:j*m) ...
            =reshape(faces(:,1+sum(nfaces(1:count-1))),n,m);
        count = count + 1;
    end
end
imagesc(allPersons), colormap gray
```

Code 1.6: [Python] Plot image for each person in the Yale database (Fig. 1.16).

```
mat_contents = scipy.io.loadmat(os.path.join('..','DATA','
    allFaces.mat'))
faces = mat_contents['faces']
m = int(mat_contents['m'])
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

```
n = int(mat_contents['n'])
nfaces = np.ndarray.flatten(mat_contents['nfaces'])

allPersons = np.zeros((n*6,m*6))
count = 0

for j in range(6):
    for k in range(6):
        allPersons[j*n : (j+1)*n, k*m : (k+1)*m] = np.
           reshape(faces[:,np.sum(nfaces[:count])],(m,n)).T
        count += 1

img = plt.imshow(allPersons)
```

As mentioned before, each image is reshaped into a large column vector, and the average face is computed and subtracted from each column vector. The mean-subtracted image vectors are then stacked horizontally as columns in the data matrix $\mathbf{X}$, as shown in Fig. 1.17. Thus, taking the SVD of the mean-subtracted matrix $\mathbf{X}$ results in the PCA. The columns of $\mathbf{U}$ are the eigenfaces, and they may be reshaped back into 192×168 images. This is illustrated in Code 1.7.

Code 1.7: [MATLAB] Compute eigenfaces on mean-subtracted data.

```
% We use the first 36 people for training data
trainingFaces = faces(:,1:sum(nfaces(1:36)));
avgFace = mean(trainingFaces,2);  % size n*m by 1;

% Compute eigenfaces on mean-subtracted training data
X = trainingFaces-avgFace*ones(1,size(trainingFaces,2));
[U,S,V] = svd(X,'econ');

imagesc(reshape(avgFace,n,m)) % Plot avg face
imagesc(reshape(U(:,1),n,m))  % Plot first eigenface
```

Code 1.7: [Python] Compute eigenfaces on mean-subtracted data.

```
# We use the first 36 people for training data
trainingFaces = faces[:,:np.sum(nfaces[:36])]
avgFace = np.mean(trainingFaces,axis=1) # size n*m by 1

# Compute eigenfaces on mean-subtracted training data
X = trainingFaces - np.tile(avgFace,(trainingFaces.shape
    [1],1)).T
U, S, VT = np.linalg.svd(X,full_matrices=0)
img_avg = ax1.imshow(np.reshape(avgFace,(m,n)).T)
img_u1 = ax2.imshow(np.reshape(U[:,0],(m,n)).T)
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

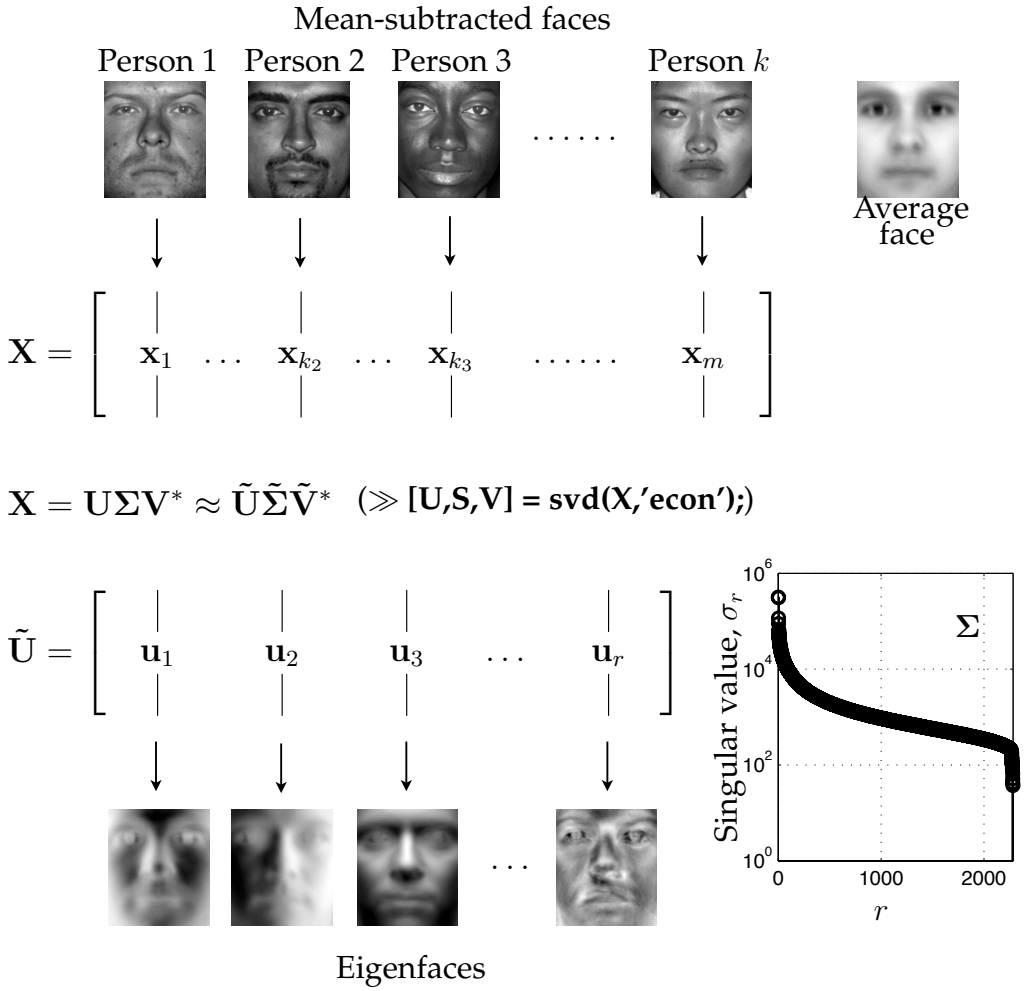


Figure 1.17: Schematic procedure to obtain eigenfaces from library of faces $\mathbf{X}$ after subtracting off average face $\bar{\mathbf{X}}$.

Using the eigenfaces library, $\tilde{\mathbf{U}}$, obtained above, we now attempt to approximately represent an image that was not in the training data. At the beginning, we held back two individuals (the 37th and 38th people), and we now use one of their images as a test image, $\mathbf{x}_{\text{test}}$. We will see how well a rank-r SVD basis will approximate this image using the following projection:

$$\tilde{\mathbf{x}}_{\text{test}} = \tilde{\mathbf{U}}\tilde{\mathbf{U}}^*\mathbf{x}_{\text{test}}.$$

The eigenface approximation for various values of r is shown in Fig. 1.18, as computed using Code 1.8. The approximation is relatively poor for $r \leq 200$, although for $r > 400$ it converges to a passable representation of the test image.

It is interesting to note that the eigenface space is not only useful for representing human faces, but may also be used to approximate a dog (Fig. 1.19) or a cappuccino (Fig. 1.20). This is possible because the 1600 eigenfaces span a large subspace of the $32\,256$-dimensional image space corresponding to broad, smooth, non-localized spatial features, such as cheeks, forehead, mouths, etc.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

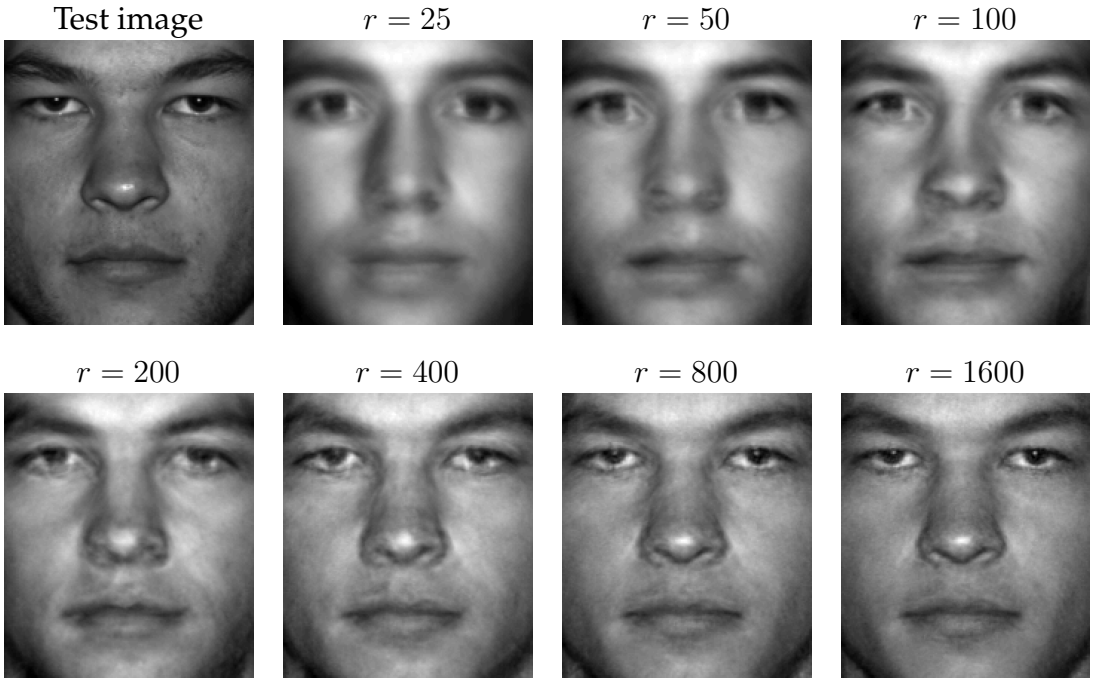


Figure 1.18: Approximate representation of test image using eigenfaces basis of various order r. Test image is not in training set.

Code 1.8: [MATLAB] Approximate test image omitted from training data.

```
testFace = faces(:,1+sum(nfaces(1:36))); % Person 37
testFaceMS = testFace - avgFace;
for r=[25 50 100 200 400 800 1600]
    reconFace = avgFace + (U(:,1:r)*(U(:,1:r)'*testFaceMS));
    imagesc(reshape(reconFace,n,m))
end
```

Code 1.8: [Python] Approximate test image omitted from training data.

```
testFace = faces[:,np.sum(nfaces[:36])] # Person 37
testFaceMS = testFace - avgFace
r_list = [25, 50, 100, 200, 400, 800, 1600]
for r in r_list:
    reconFace = avgFace + U[:,:r]  @ U[:,:r].T @ testFaceMS
    img = plt.imshow(np.reshape(reconFace,(m,n)).T)
```

We further investigate the use of the eigenfaces as a coordinate system, defining an eigenface space. By projecting an image $\mathbf{x}$ onto the first r PCA modes, we obtain a set of coordinates in this space: $\tilde{\mathbf{x}} = \tilde{\mathbf{U}}^*\mathbf{x}$. Some principal components may capture the most common features shared among all human faces, while other principal components will be more useful for distinguishing between individuals. Additional principal components may capture differ-

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

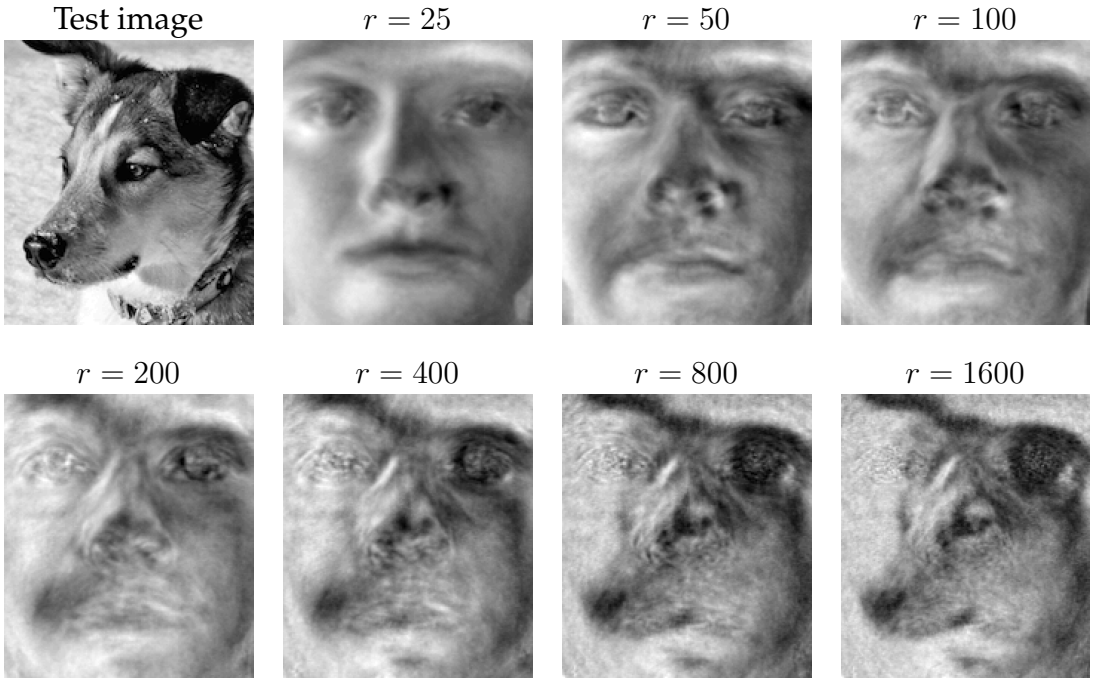


Figure 1.19: Approximate representation of an image of a dog using eigenfaces.

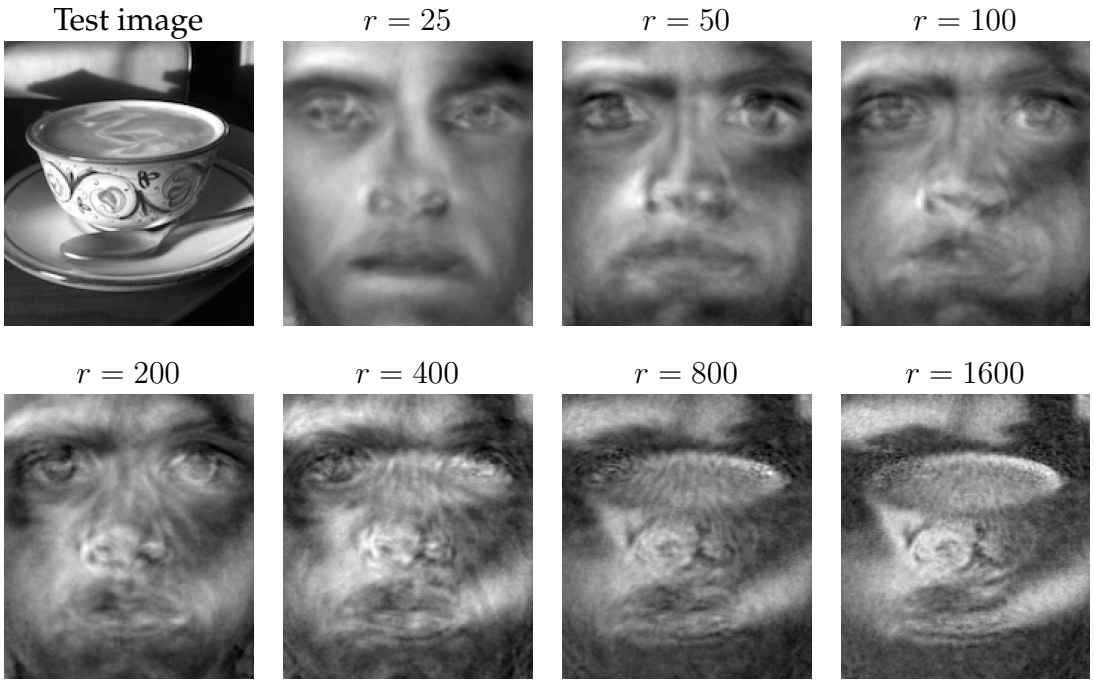


Figure 1.20: Approximate representation of a cappuccino using eigenfaces.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

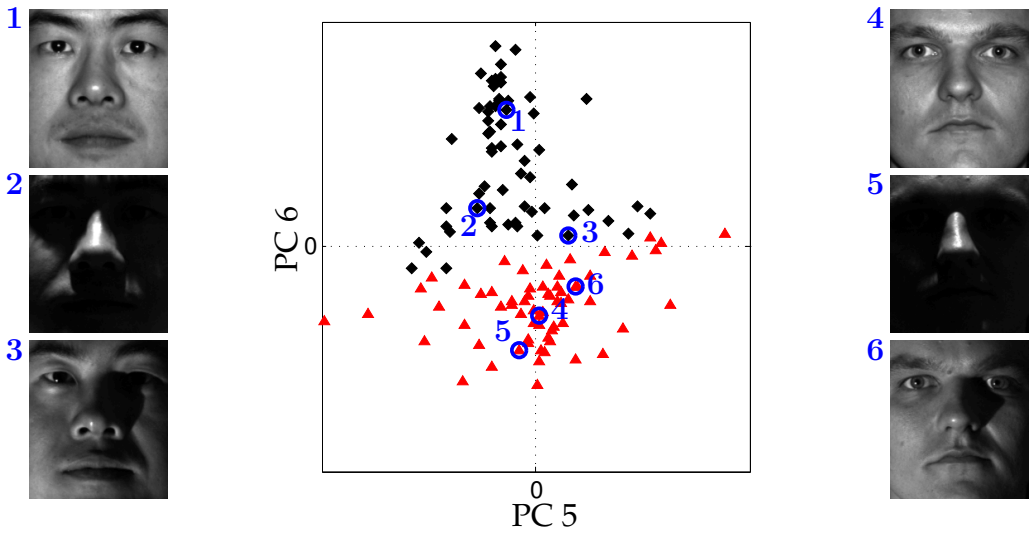


Figure 1.21: Projection of all images from two individuals onto the 5th and 6th PCA modes. Projected images of the first individual are indicated with black diamonds, and projected images of the second individual are indicated with red triangles. Three examples from each individual are circled in blue, and the corresponding image is shown.

ences in lighting angles. Figure 1.21 shows the coordinates of all 64 images of two individuals projected onto the 5th and 6th principal components, generated by Code 1.9. Images of the two individuals appear to be well separated in these coordinates. This is the basis for image recognition and classification in Chapter 5.

Code 1.9: [MATLAB] Project images for two specific people onto the 5th and 6th eigenfaces to illustrate the potential for automated classification.

```
P1num = 2;   % Person number 2
P2num = 7;   % Person number 7

P1 = faces(:,1+sum(nfaces(1:P1num-1)):sum(nfaces(1:P1num)));
P2 = faces(:,1+sum(nfaces(1:P2num-1)):sum(nfaces(1:P2num)));

P1 = P1 - avgFace*ones(1,size(P1,2));
P2 = P2 - avgFace*ones(1,size(P2,2));

PCAmodes = [5 6];     % Project onto PCA modes 5 and 6
PCACoordsP1 = U(:,PCAmodes)'*P1;
PCACoordsP2 = U(:,PCAmodes)'*P2;

plot(PCACoordsP1(1,:),PCACoordsP1(2,:),'kd'), hold on
plot(PCACoordsP2(1,:),PCACoordsP2(2,:),'r^')
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Code 1.9: [Python] Project images for two specific people onto the 5th and 6th eigenfaces to illustrate the potential for automated classification.

```
P1num = 2 # Person number 2
P2num = 7 # Person number 7

P1 = faces[:,np.sum(nfaces[:(P1num-1)]):np.sum(nfaces[:P1num
    ])]
P2 = faces[:,np.sum(nfaces[:(P2num-1)]):np.sum(nfaces[:P2num
    ])]

P1 = P1 - np.tile(avgFace,(P1.shape[1],1)).T
P2 = P2 - np.tile(avgFace,(P2.shape[1],1)).T

PCAmodes = [5, 6] # Project onto PCA modes 5 and 6
PCACoordsP1 = U[:,PCAmodes-np.ones_like(PCAmodes)].T @ P1
PCACoordsP2 = U[:,PCAmodes-np.ones_like(PCAmodes)].T @ P2

plt.plot(PCACoordsP1[0,:],PCACoordsP1[1,:],'d',Color='k')
plt.plot(PCACoordsP2[0,:],PCACoordsP2[1,:],'^',Color='r')
```

1.7 Truncation and Alignment

Deciding how many singular values to keep, i.e., where to truncate, is one of the most important and contentious decisions when using the SVD. There are many factors, including specifications on the desired rank of the system, the magnitude of noise, and the distribution of the singular values. Often, one truncates the SVD at a rank r that captures a predetermined amount of the variance or energy in the original data, such as 90% or 99% truncation. Although crude, this technique is commonly used. Other techniques involve identifying "elbows" or "knees" in the singular value distribution, which may denote the transition from singular values that represent important patterns from those that represent noise. Truncation may be viewed as a hard threshold on singular values, where values larger than a threshold τ are kept, while remaining singular values are truncated. Recent work by Gavish and Donoho [267] provides an optimal truncation value, or hard threshold, under certain conditions, providing a principled approach to obtaining low-rank matrix approximations using the SVD.

In addition, the alignment of data significantly impacts the rank of the SVD approximation. The SVD essentially relies on a separation of variables between the columns and rows of a data matrix. In many situations, such as when analyzing traveling waves or misaligned data, this assumption breaks down, resulting in an artificial rank inflation.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Optimal Hard Threshold

A recent theoretical breakthrough determines the *optimal* hard threshold τ for singular value truncation under the assumption that a matrix has a low-rank structure contaminated with Gaussian white noise [267]. This work builds on a significant literature surrounding various techniques for hard and soft thresholding of singular values. In this section, we summarize the main results and demonstrate the thresholding on various examples. For more details, see [267].

First, we assume that the data matrix $\mathbf{X}$ is the sum of an underlying low-rank, or approximately low-rank, matrix $\mathbf{X}_{\text{true}}$ and a noise matrix $\mathbf{X}_{\text{noise}}$:

$$\mathbf{X} = \mathbf{X}_{\text{true}} + \gamma \mathbf{X}_{\text{noise}}. \tag{1.43}$$

The entries of $\mathbf{X}_{\text{noise}}$ are assumed to be independent, identically distributed (i.i.d.) Gaussian random variables with zero mean and unit variance. The magnitude of the noise is characterized by γ, which deviates from the notation in [267].[8]

When the noise magnitude γ is known, there are closed-form solutions for the optimal hard threshold τ:

1. If $\mathbf{X} \in \mathbb{R}^{n \times n}$ is square, then

$$\tau = (4/\sqrt{3})\sqrt{n}\,\gamma. \tag{1.44}$$

2. If $\mathbf{X} \in \mathbb{R}^{n \times m}$ is rectangular and $m \ll n$, then the constant $4/\sqrt{3}$ is replaced by a function of the aspect ratio $\beta = m/n$:

$$\tau = \lambda(\beta)\sqrt{n}\,\gamma, \tag{1.45}$$

$$\lambda(\beta) = \left(2(\beta+1) + \frac{8\beta}{(\beta+1) + (\beta^2 + 14\beta + 1)^{1/2}} \right)^{1/2}. \tag{1.46}$$

   Note that this expression reduces to (1.44) when $\beta = 1$. If $n \ll m$, then $\beta = n/m$.

When the noise magnitude γ is unknown, which is more typical in real-world applications, then it is possible to estimate the noise magnitude and scale the distribution of singular values by using σ_{med}, the *median* singular value. In this case, there is no closed-form solution for τ, and it must be approximated numerically:

3. For unknown noise γ, and a rectangular matrix $\mathbf{X} \in \mathbb{R}^{n \times m}$, the optimal hard threshold is given by

$$\tau = \omega(\beta)\sigma_{\text{med}}. \tag{1.47}$$

[8] In [267], σ is used to denote standard deviation and y_k denotes the kth singular value.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Here, $\omega(\beta) = \lambda(\beta)/\mu_\beta$, where μ_β is the solution to the following problem:

$$\int_{(1-\beta)^2}^{\mu_\beta} \frac{\{[(1+\sqrt{\beta})^2 - t][t - (1-\sqrt{\beta})^2]\}^{1/2}}{2\pi t} \mathrm{d}t = \frac{1}{2}.$$

Solutions to the expression above must be approximated numerically. Fortunately [267] has a MATLAB code supplement[9] [206] to approximate μ_β.

The new method of optimal hard thresholding works remarkably well, as demonstrated on the examples below.

Example 1: Toy Problem

In the first example, shown in Fig. 1.22 and Code 1.10, we artificially construct a rank-two matrix and contaminate the data with Gaussian white noise. A denoised and dimensionally reduced matrix is then obtained using the threshold from (1.44), as well as a truncation keeping 90% of the cumulative sum of singular values. It is clear that the hard threshold is able to filter the noise more effectively. Plotting the singular values in Fig. 1.23, it is clear that there are two values that are above threshold.

Code 1.10: [MATLAB] Compare various thresholding approaches on noisy low-rank data (Fig. 1.22).

```
% Generate underlying low-rank data
t = (-3:.01:3)';

Utrue = [cos(17*t).*exp(-t.^2) sin(11*t)];
Strue = [2 0; 0 .5];
Vtrue = [sin(5*t).*exp(-t.^2) cos(13*t)];

X = Utrue*Strue*Vtrue';
figure, imshow(X);

% Contaminate signal with noise
sigma = 1;
Xnoisy = X+sigma*randn(size(X));
figure, imshow(Xnoisy);

% Truncate using optimal hard threshold
[U,S,V] = svd(Xnoisy);

N = size(Xnoisy,1);
```

[9] http://purl.stanford.edu/vg705qn9070

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

```
cutoff = (4/sqrt(3))*sqrt(N)*sigma; % Hard threshold
r = max(find(diag(S)>cutoff)); % Keep modes w/ sig > cutoff
Xclean = U(:,1:r)*S(1:r,1:r)*V(:,1:r)';
figure, imshow(Xclean)

% Truncate to keep 90% of cumulative sum
cdS = cumsum(diag(S))./sum(diag(S));  % Cumulative sum
r90 = min(find(cdS>0.90));  % Find r to capture 90% of sum

X90 = U(:,1:r90)*S(1:r90,1:r90)*V(:,1:r90)';
figure, imshow(X90)
```

Code 1.10: [Python] Compare various thresholding approaches on noisy low-rank data (Fig. 1.22).

```
# Generate underlying low-rank data
t = np.arange(-3,3,0.01)

Utrue = np.array([np.cos(17*t) * np.exp(-t**2), np.sin(11*t)
    ]).T
Strue = np.array([[2, 0],[0, 0.5]])
Vtrue = np.array([np.sin(5*t) * np.exp(-t**2), np.cos(13*t)
    ]).T

X = Utrue @ Strue @ Vtrue.T
plt.imshow(X)

# Contaminate signal with noise
sigma = 1
Xnoisy = X + sigma*np.random.randn(*X.shape)
plt.imshow(Xnoisy)

# Truncate using optimal hard threshold
U, S, VT = np.linalg.svd(Xnoisy,full_matrices=0)

N = Xnoisy.shape[0]
cutoff = (4/np.sqrt(3)) * np.sqrt(N) * sigma #Hard threshold
r = np.max(np.where(S > cutoff)) # Keep modes w/ S > cutoff
Xclean = U[:,:(r+1)] @ np.diag(S[:(r+1)]) @ VT[:(r+1),:]
plt.imshow(Xclean)

# Truncate to keep 90% of cumulative sum
cdS = np.cumsum(S) / np.sum(S) # Cumulative energy
r90 = np.min(np.where(cdS > 0.90)) # Find r to keep 90% sum

X90 = U[:,:(r90+1)] @ np.diag(S[:(r90+1)]) @ VT[:(r90+1),:]
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

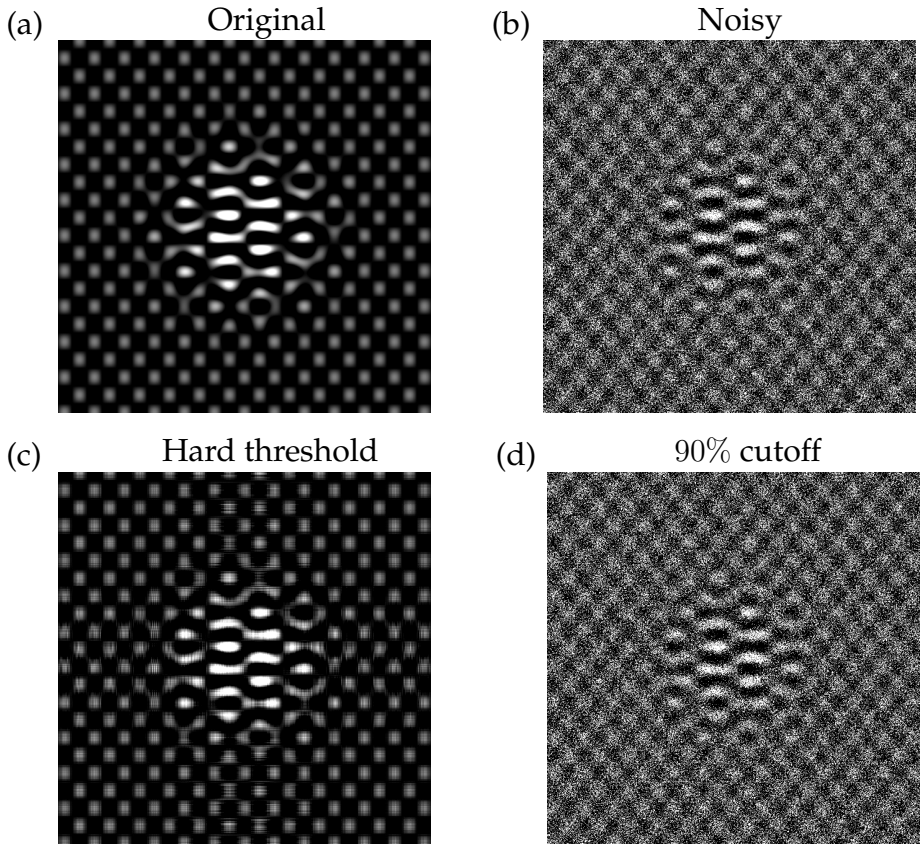


Figure 1.22: (a) Underlying rank-two matrix, (b) matrix with noise, (c) clean matrix after optimal hard threshold $(4/\sqrt{3})\sqrt{n}\,\sigma$, and (d) truncation keeping 90% of the cumulative sum of singular values.

```
plt.imshow(X90)
```

Example 2: Eigenfaces

In the second example, we revisit the eigenfaces problem from Section 1.6. This provides a more typical example, since the data matrix $\mathbf{X}$ is rectangular, with aspect ratio $\beta = 3/4$, and the noise magnitude is unknown. It is also not clear that the data is contaminated with white noise. Nonetheless, the method determines a threshold τ, above which columns of $\mathbf{U}$ appear to have strong facial features, and below which columns of $\mathbf{U}$ consist mostly of noise, shown in Fig. 1.24.

Importance of Data Alignment

Here, we discuss common pitfalls of the SVD associated with misaligned data. The following example is designed to illustrate one of the central weaknesses of

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

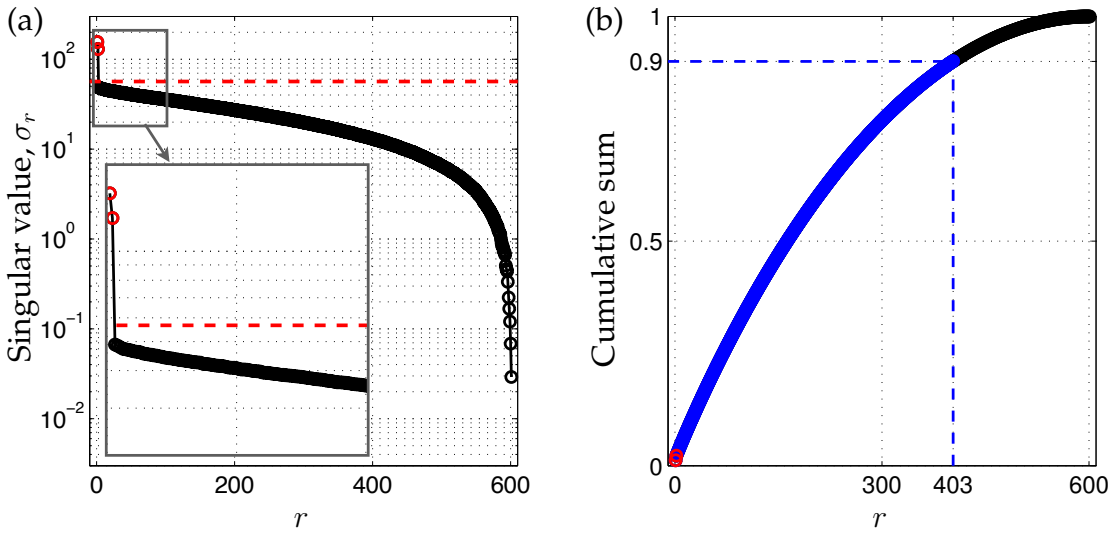


Figure 1.23: (a) Singular values σ_r and (b) cumulative sum in first r modes. The optimal hard threshold $\tau = (4/\sqrt{3})\sqrt{n}\,\sigma$ is shown as a red dashed line, and the 90% cutoff is shown as a blue dashed line. For this case, $n = 600$ and $\sigma = 1$, so that the optimal cutoff is approximately $\tau = 56.6$.

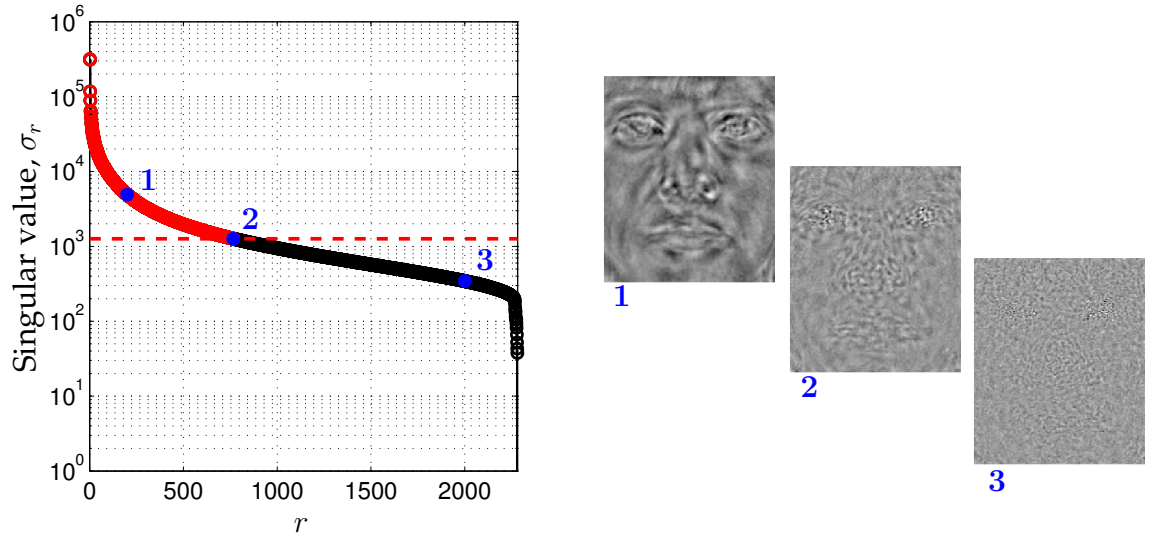


Figure 1.24: Hard thresholding for eigenfaces example.

the SVD for dimensionality reduction and coherent feature extraction in data. Consider a matrix of zeros with a rectangular sub-block consisting of ones. As an image, this would look like a white rectangle placed on a black background (see Fig. 1.25(a)). If the rectangle is perfectly aligned with the x- and y-axes of the figure, then the SVD is simple, having only one non-zero singular value σ_1 (see Fig. 1.25(c)) and corresponding singular vectors $\mathbf{u}_1$ and $\mathbf{v}_1$ that define the

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

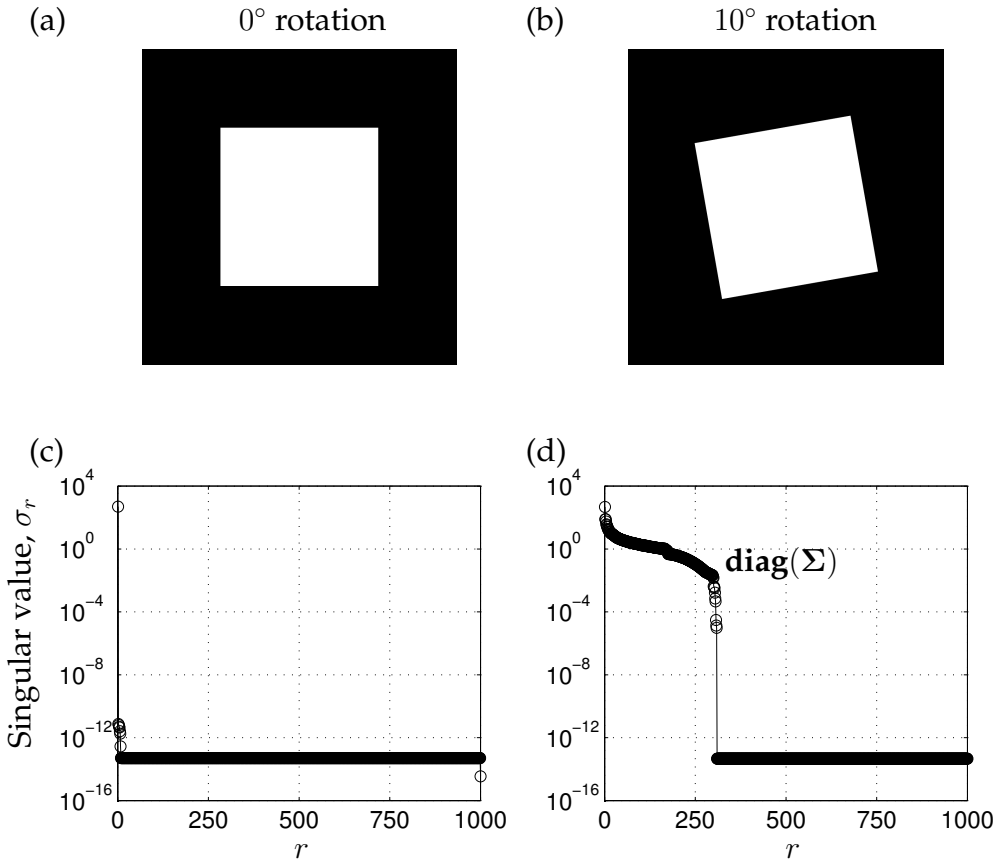


Figure 1.25: A data matrix consisting of ones with a square sub-block of zeros (a), and its SVD spectrum (c). If we rotate the image by 10°, as in (b), the SVD spectrum becomes significantly more complex (d).

width and height of the white rectangle.

When we begin to rotate the inner rectangle so that it is no longer aligned with the image axes, additional non-zero singular values begin to appear in the spectrum (see Figs. 1.25(b,d) and 1.26). Code to reproduce this example is provided on the book's GitHub.

The reason that this example breaks down is that the SVD is fundamentally *geometric*, meaning that it depends on the coordinate system in which the data is represented. As we have seen earlier, the SVD is only generically invariant to unitary transformations, meaning that the transformation preserves the inner product. This fact may be viewed as both a strength and a weakness of the method. First, the dependence of SVD on the inner product is essential for the various useful geometric interpretations. Moreover, the SVD has meaningful units and dimensions. However, this makes the SVD sensitive to the alignment of the data. In fact, the SVD rank explodes when objects in the columns trans-

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

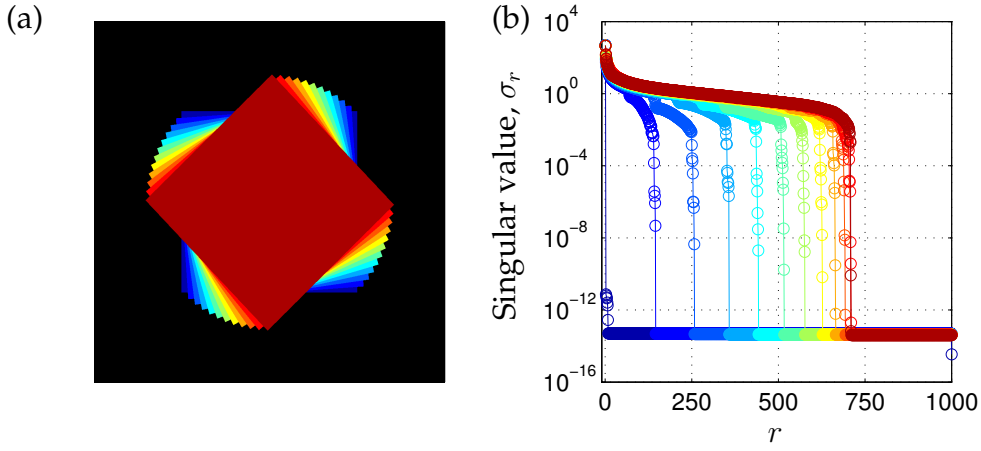


Figure 1.26: A data matrix consisting of zeros with a square sub-block of ones at various rotations (a), and the corresponding SVD spectrum, **diag(S)**, (b).

late, rotate, or scale, which severely limits its use for data that has not been heavily pre-processed.

For instance, the eigenfaces example was built on a library of images that had been meticulously cropped, centered, and aligned according to a stencil. Without taking these important pre-processing steps, the features and clustering performance would be underwhelming.

The inability of the SVD to capture translations and rotations of the data is a major limitation. For example, the SVD is still the method of choice for the low-rank decomposition of data from partial differential equations (PDEs), as will be explored in Chapters 12 and 13. However, the SVD is fundamentally a data-driven separation of variables, which we know will not work for many types of PDE, for example, those that exhibit traveling waves. Generalized decompositions that retain the favorable properties and are applicable to data with symmetries is a significant open challenge in the field.

1.8 Randomized Singular Value Decomposition

The accurate and efficient decomposition of large data matrices is one of the cornerstones of modern computational mathematics and data science. In many cases, matrix decompositions are explicitly focused on extracting dominant low-rank structure in the matrix, as illustrated throughout the examples in this chapter. Recently, it has been shown that if a matrix $\mathbf{X}$ has low-rank structure, then there are extremely efficient matrix decomposition algorithms based on the theory of random sampling; this is closely related to the idea of sparsity and the high-dimensional geometry of sparse vectors, which will be explored in Chapter 3. These so-called *randomized* numerical methods have the potential to

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

transform computational linear algebra, providing accurate matrix decompositions at a fraction of the cost of deterministic methods. Moreover, with increasingly vast measurements (e.g., from 4K and 8K video, Internet of Things, etc.), it is often the case that the *intrinsic* rank of the data does not increase appreciably, even though the dimension of the ambient measurement space grows. Thus, the computational savings of randomized methods will only become more important in the coming years and decades with the growing deluge of data.

Randomized Linear Algebra

Randomized linear algebra is a much more general concept than the treatment presented here for the SVD. In addition to the randomized SVD [488, 621], randomized algorithms have been developed for principal component analysis [308, 605], the pivoted LU decomposition [651], the pivoted QR decomposition [219], and the dynamic mode decomposition [234]. Most randomized matrix decompositions can be broken into a few common steps, as described here. There are also several excellent surveys on the topic [236, 309, 445, 471]. We assume that we are working with tall-skinny matrices, so that $n > m$, although the theory readily generalizes to short-fat matrices.

Step 0: Identify a target rank, $r < m$.

Step 1: Using random projections $\mathbf{P}$ to sample the column space, find a matrix $\mathbf{Q}$ whose columns approximate the column space of $\mathbf{X}$, i.e., so that $\mathbf{X} \approx \mathbf{Q}\mathbf{Q}^*\mathbf{X}$.

Step 2: Project $\mathbf{X}$ onto the $\mathbf{Q}$ subspace, $\mathbf{Y} = \mathbf{Q}^*\mathbf{X}$, and compute the matrix decomposition on $\mathbf{Y}$.

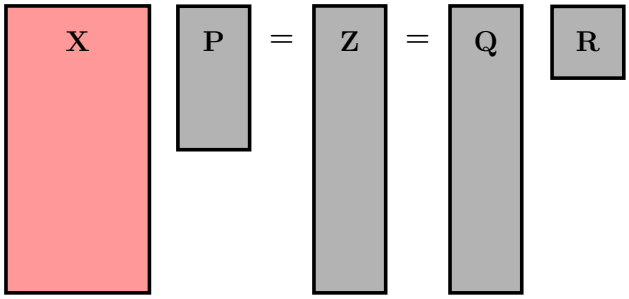
Step 3: Reconstruct high-dimensional modes $\mathbf{U} = \mathbf{Q}\mathbf{U}_\mathbf{Y}$ using $\mathbf{Q}$ and the modes computed from $\mathbf{Y}$.

Randomized SVD Algorithm

Over the past two decades, there have been several randomized algorithms proposed to compute a low-rank SVD, including the *Monte Carlo* SVD [253] and more robust approaches based on random projections [446, 488, 621]. These methods were improved by incorporating structured sampling matrices for faster matrix multiplications [761]. Here, we use the randomized SVD algorithm of Halko, Martinsson, and Tropp [309], which combined and expanded on these previous algorithms, providing favorable error bounds. Additional analysis and numerical implementation details are found in Voronin and Martinsson [739]. A schematic of the rSVD algorithm is shown in Fig. 1.27.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Step 1



Step 2

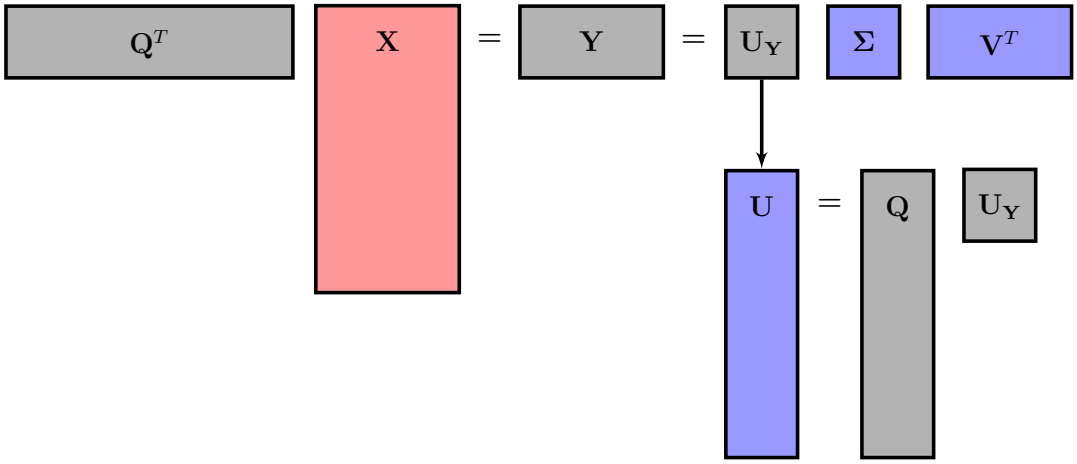


Figure 1.27: Schematic of randomized SVD algorithm. The high-dimensional data $\mathbf{X}$ is depicted in red, intermediate steps in gray, and the outputs in blue. This algorithm requires two passes over $\mathbf{X}$.

Step 1. We construct a random projection matrix $\mathbf{P} \in \mathbb{R}^{m \times r}$ to sample the column space of $\mathbf{X} \in \mathbb{R}^{n \times m}$:

$$\mathbf{Z} = \mathbf{X}\mathbf{P}. \tag{1.48}$$

The matrix $\mathbf{Z}$ may be much smaller than $\mathbf{X}$, especially for low-rank matrices with $r \ll m$. It is highly unlikely that a random projection matrix $\mathbf{P}$ will project out important components of $\mathbf{X}$, and so $\mathbf{Z}$ approximates the column space of $\mathbf{X}$ with high probability. Thus, it is possible to compute the low-rank QR decomposition of $\mathbf{Z}$ to obtain an orthonormal basis for $\mathbf{X}$:

$$\mathbf{Z} = \mathbf{Q}\mathbf{R}. \tag{1.49}$$

Step 2. With the low-rank basis $\mathbf{Q}$, we may project $\mathbf{X}$ into a smaller space:

$$\mathbf{Y} = \mathbf{Q}^*\mathbf{X}. \tag{1.50}$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

It also follows that $\mathbf{X} \approx \mathbf{QY}$, with better agreement when the singular values σ_k decay rapidly for $k > r$.

It is now possible to compute the singular value decomposition on $\mathbf{Y}$:

$$\mathbf{Y} = \mathbf{U}_{\mathbf{Y}}\mathbf{\Sigma}\mathbf{V}^*. \tag{1.51}$$

Because $\mathbf{Q}$ is orthonormal and approximates the column space of $\mathbf{X}$, the matrices $\mathbf{\Sigma}$ and $\mathbf{V}$ are the same for $\mathbf{Y}$ and $\mathbf{X}$, as discussed in Section 1.3.

Step 3. Finally, it is possible to reconstruct the high-dimensional left singular vectors $\mathbf{U}$ using $\mathbf{U}_{\mathbf{Y}}$ and $\mathbf{Q}$:

$$\mathbf{U} = \mathbf{QU}_{\mathbf{Y}}. \tag{1.52}$$

Oversampling

Most matrices $\mathbf{X}$ do not have an exact low-rank structure, given by r modes. Instead, there are non-zero singular values σ_k for $k > r$, and the sketch $\mathbf{Z}$ will not exactly span the column space of $\mathbf{X}$. In general, increasing the number of columns in $\mathbf{P}$ from r to $r + p$ significantly improves results, even with p adding around 5–10 columns [487]. This is known as *oversampling*, and increasing p decreases the variance of the singular value spectrum of the sketched matrix.

Power Iterations

A second challenge in using randomized algorithms is when the singular value spectrum decays slowly, so that the remaining truncated singular values contain significant variance in the data $\mathbf{X}$. In this case, it is possible to pre-process $\mathbf{X}$ through q *power iterations* [299, 309, 605] to create a new matrix $\mathbf{X}^{(q)}$ with a more rapid singular value decay:

$$\mathbf{X}^{(q)} = (\mathbf{XX}^*)^q\mathbf{X}. \tag{1.53}$$

Power iterations dramatically improve the quality of the randomized decomposition, as the singular value spectrum of $\mathbf{X}^{(q)}$ decays more rapidly:

$$\mathbf{X}^{(q)} = \mathbf{U}\mathbf{\Sigma}^{2q+1}\mathbf{V}^*. \tag{1.54}$$

However, power iterations are expensive, requiring q additional passes through the data $\mathbf{X}$. In some extreme examples, the data in $\mathbf{X}$ may be stored in a distributed architecture, so that every additional pass adds considerable expense.

Guaranteed Error Bounds

One of the most important properties of the randomized SVD is the existence of tunable error bounds, which are explicit functions of the singular value spectrum, the desired rank r, the oversampling parameter p, and the number of

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

power iterations q. The best attainable error bound for a deterministic algorithm is

$$\|\mathbf{X} - \mathbf{QY}\|_2 \geq \sigma_{r+1}(\mathbf{X}). \tag{1.55}$$

In other words, the approximation with the best possible rank-r subspace $\mathbf{Q}$ will have error greater than or equal to the next truncated singular value of $\mathbf{X}$. For randomized methods, it is possible to bound the *expectation* of the error:

$$\mathbb{E}(\|\mathbf{X} - \mathbf{QY}\|_2) \leq \left(1 + \sqrt{\frac{r}{p-1}} + \frac{e\sqrt{r+p}}{p}\sqrt{m-r}\right)^{1/(2q+1)} \sigma_{k+1}(\mathbf{X}), \tag{1.56}$$

where e is Euler's number.

Choice of Random Matrix P

There are several suitable choices of the random matrix $\mathbf{P}$. Gaussian random projections (e.g., the elements of $\mathbf{P}$ are i.i.d. Gaussian random variables) are frequently used because of favorable mathematical properties and the richness of information extracted in the sketch $\mathbf{Z}$. In particular, it is very unlikely that a Gaussian random matrix $\mathbf{P}$ will be chosen *badly* so as to project out important information in $\mathbf{X}$. However, Gaussian projections are expensive to generate, store, and compute. Uniform random matrices are also frequently used, and have similar limitations. There are several alternatives, such as Rademacher matrices, where the entries can be $+1$ or -1 with equal probability [720]. Structured random projection matrices may provide efficient sketches, reducing computational costs to $\mathcal{O}(nm\log(r))$ [761]. Yet another choice is a sparse projection matrix $\mathbf{P}$, which improves storage and computation, but at the cost of including less information in the sketch. In the extreme case, when even a single pass over the matrix $\mathbf{X}$ is prohibitively expensive, the matrix $\mathbf{P}$ may be chosen as random columns of the $m \times m$ identity matrix, so that it randomly selects columns of $\mathbf{X}$ for the sketch $\mathbf{Z}$. This is the fastest option, but should be used with caution, as information may be lost if the structure of $\mathbf{X}$ is highly localized in a subset of columns, which may be lost by column sampling.

Example of Randomized SVD

To demonstrate the randomized SVD algorithm, we will decompose a high-resolution image. This particular implementation is only for illustrative purposes, as it has not been optimized for speed, data transfer, or accuracy. In practical applications, care should be taken [236, 309].

Code 1.11 computes the randomized SVD of a matrix $\mathbf{X}$, and Code 1.12 uses this function to obtain a rank-400 approximation to a high-resolution image, shown in Fig. 1.28.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

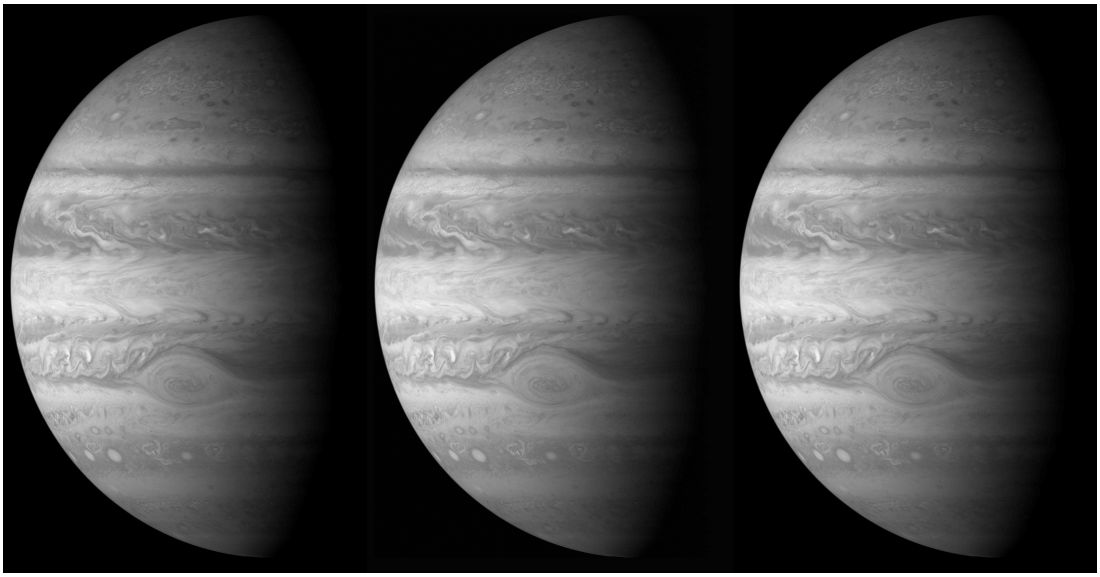


Figure 1.28: Original high-resolution (left) and rank-400 approximations from the SVD (middle) and rSVD (right).

Code 1.11: [MATLAB] Randomized SVD algorithm.

```
function [U,S,V] = rsvd(X,r,q,p);

% Step 1: Sample column space of X with P matrix
ny = size(X,2);
P = randn(ny,r+p);
Z = X*P;
for k=1:q
    Z = X*(X'*Z);
end
[Q,R] = qr(Z,0);

% Step 2: Compute SVD on projected Y=Q'*X;
Y = Q'*X;
[UY,S,V] = svd(Y,'econ');
U = Q*UY;
```

Code 1.11: [Python] Randomized SVD algorithm.

```
def rSVD(X,r,q,p):
    # Step 1: Sample column space of X with P matrix
    ny = X.shape[1]
    P = np.random.randn(ny,r+p)
    Z = X @ P
    for k in range(q):
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

```
        Z = X @ (X.T @ Z)

    Q, R = np.linalg.qr(Z,mode='reduced')

    # Step 2: Compute SVD on projected Y = Q.T @ X
    Y = Q.T @ X
    UY, S, VT = np.linalg.svd(Y,full_matrices=0)
    U = Q @ UY

    return U, S, VT
```

Code 1.12: [MATLAB] Compute the randomized SVD of high-resolution image.

```
A=imread('jupiter.jpg');
X=double(rgb2gray(A));
[U,S,V] = svd(X,'econ');     % Deterministic SVD

r = 400; % Target rank
q = 1;   % Power iterations
p = 5;   % Oversampling parameter
[rU,rS,rV] = rsvd(X,r,q,p); % Randomized SVD

%% Reconstruction
XSVD = U(:,1:r)*S(1:r,1:r)*V(:,1:r)';      % SVD approx.
errSVD = norm(X-XSVD,2)/norm(X,2);
XrSVD = rU(:,1:r)*rS(1:r,1:r)*rV(:,1:r)'; % rSVD approx.
errrSVD = norm(X-XrSVD,2)/norm(X,2);
```

Code 1.12: [Python] Compute the randomized SVD of high-resolution image.

```
A = imread(os.path.join('..','DATA','jupiter.jpg'))
X = np.mean(A,axis=2) # Convert RGB -> grayscale
U, S, VT = np.linalg.svd(X,full_matrices=0) # Full SVD

r = 400 # Target rank
q = 1   # Power iterations
p = 5   # Oversampling parameter
rU, rS, rVT = rSVD(X,r,q,p)

## Reconstruction
XSVD = U[:,:(r+1)] @ np.diag(S[:(r+1)]) @ VT[:(r+1),:]
errSVD = np.linalg.norm(X-XSVD,ord=2)/np.linalg.norm(X,ord
    =2)
XrSVD = rU[:,:(r+1)] @ np.diag(rS[:(r+1)]) @ rVT[:(r+1),:]
errSVD = np.linalg.norm(X-XrSVD,ord=2)/np.linalg.norm(X,ord
    =2)
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

1.9 Tensor Decompositions and N-Way Data Arrays

Low-rank decompositions can be generalized beyond matrices. This is important, as the SVD requires that disparate types of data be flattened into a single vector in order to evaluate correlated structures. For instance, different time snapshots (columns) of a matrix may include measurements as diverse as temperature, pressure, concentration of a substance, etc. Additionally, there may be categorical data. Vectorizing this data generally does not make sense. Ultimately, what is desired is to preserve the various data structures and types in their own, independent directions. Matrices can be generalized to N-way arrays, or tensors, where the data is more appropriately arranged without forcing a data-flattening process.

The construction of data tensors requires that we revisit the notation associated with tensor addition, multiplication, and inner products [401]. We denote the rth column of a matrix $\mathbf{A}$ by $\mathbf{a}_r$. Given matrices $\mathbf{A} \in \mathbb{R}^{I\times K}$ and $\mathbf{B} \in \mathbb{R}^{J\times K}$, their Khatri–Rao product is denoted by $\mathbf{A} \odot \mathbf{B}$ and is defined to be the $IJ \times K$ matrix of column-wise Kronecker products, namely

$$\mathbf{A} \odot \mathbf{B} = \begin{pmatrix} \mathbf{a}_1 \otimes \mathbf{b}_1 & \cdots & \mathbf{a}_K \otimes \mathbf{b}_K \end{pmatrix}.$$

For an N-way tensor $\boldsymbol{\mathcal{A}}$ of size $I_1 \times I_2 \times \cdots \times I_N$, we denote its $\mathbf{i} = (i_1, i_2, \ldots, i_N)$ entry by $a_{\mathbf{i}}$.

The inner product between two N-way tensors $\boldsymbol{\mathcal{A}}$ and $\boldsymbol{\mathcal{B}}$ of compatible dimensions is given by

$$\langle \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{B}} \rangle = \sum_{\mathbf{i}} a_{\mathbf{i}} b_{\mathbf{i}}.$$

The Frobenius norm of a tensor $\boldsymbol{\mathcal{A}}$, denoted by $\|\boldsymbol{\mathcal{A}}\|_{\mathrm{F}}$, is the square root of the inner product of $\boldsymbol{\mathcal{A}}$ with itself, namely $\|\boldsymbol{\mathcal{A}}\|_{\mathrm{F}} = \sqrt{\langle \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{A}} \rangle}$. Finally, the mode-$n$ matricization or unfolding of a tensor $\boldsymbol{\mathcal{A}}$ is denoted by $\mathbf{mA}_{(n)}$.

Let $\boldsymbol{\mathcal{M}}$ represent an N-way data tensor of size $I_1 \times I_2 \times \cdots \times I_N$. We are interested in an R-component CANDECOMP/PARAFAC (CP) [166, 314, 401] factor model

$$\boldsymbol{\mathcal{M}} = \sum_{r=1}^{R} \lambda_r \, \mathbf{ma}_r^{(1)} \circ \cdots \circ \mathbf{ma}_r^{(N)}, \tag{1.57}$$

where $\circ$ represents outer product and $\mathbf{ma}_r^{(n)}$ represents the rth column of the *factor matrix* $\mathbf{mA}^{(n)}$ of size $I_n \times R$. The CP decomposition refers to *canonical decomposition* (CANDECOMP) and *parallel factors analysis* (PARAFAC), respectively. We refer to each summand as a *component*. Assuming each factor matrix has been column-normalized to have unit Euclidean length, we refer to the λ_r

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

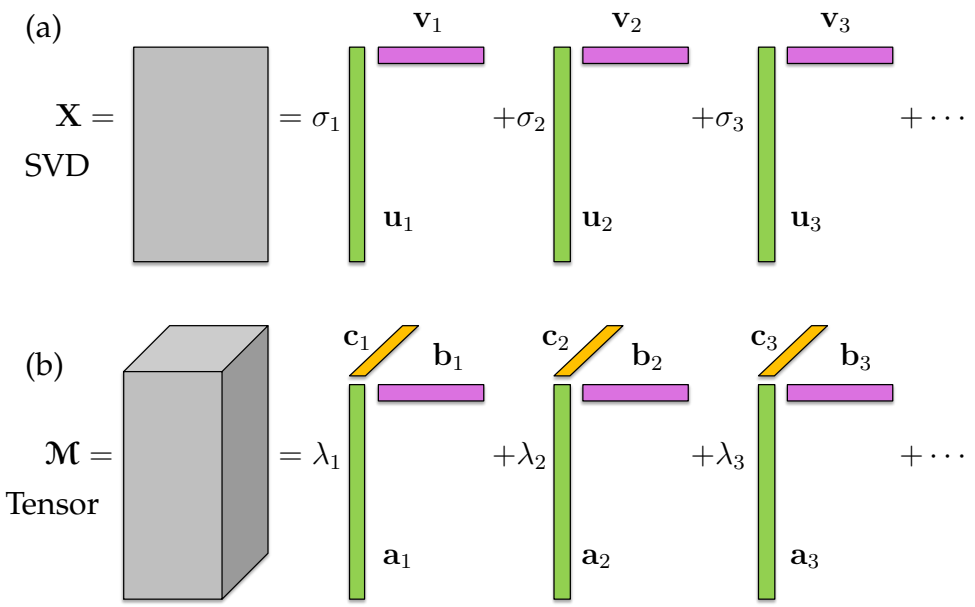


Figure 1.29: Comparison of the SVD and tensor decomposition frameworks. Both methods produce an approximation to the original data matrix by sums of outer products. Specifically, the tensor decomposition generalizes the concept of the SVD to N-way arrays of data without having to flatten (vectorize) the data.

as *weights*. We will use the shorthand notation where $\lambda = (\lambda_1, \ldots, \lambda_R)^T$ [35]. A tensor that has a CP decomposition is sometimes referred to as a Kruskal tensor.

For the rest of this chapter, we consider a three-way CP tensor decomposition (see Fig. 1.29), where two modes index state variation and the third mode indexes time variation:

$$\mathcal{M} = \sum_{r=1}^{R} \lambda_r \, \mathbf{A}_r \circ \mathbf{B}_r \circ \mathbf{C}_r.$$

Let $\mathbf{A} \in \mathbb{R}^{I_1 \times R}$ and $\mathbf{B} \in \mathbb{R}^{I_2 \times R}$ denote the factor matrices corresponding to the two state modes and $\mathbf{C} \in \mathbb{R}^{I_3 \times R}$ denote the factor matrix corresponding to the time mode. This three-way decomposition is compared to the SVD in Fig. 1.29.

To illustrate the tensor decomposition, we use the MATLAB N-way toolbox developed by Bro and co-workers [21, 116], which is available on the Mathworks file exchange. This simple-to-use package provides a variety of tools to extract tensor decompositions and evaluate the factor models generated. In the specific example considered here, we generate data from a spatio-temporal

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

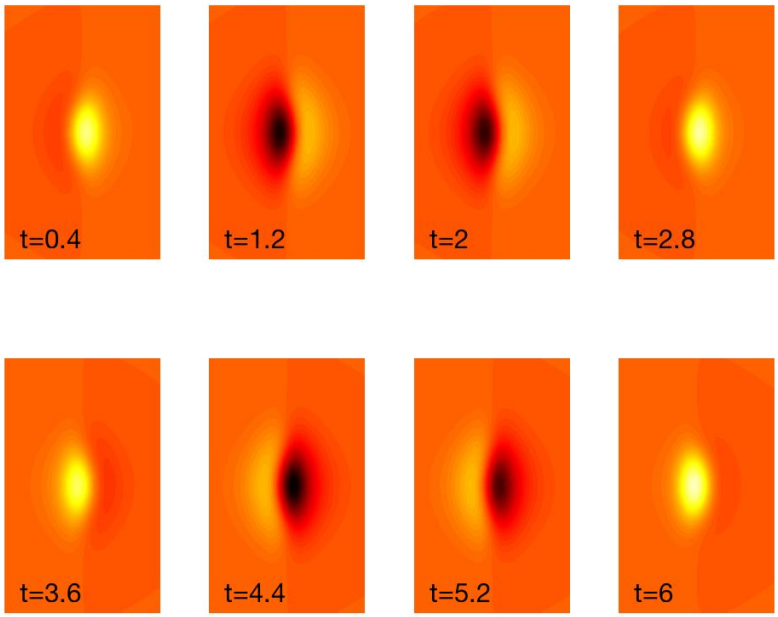


Figure 1.30: Example N-way array data set created from the function (1.58). The data matrix is $\mathbf{A} \in \mathbb{R}^{121\times101\times315}$. A CP tensor decomposition can be used to extract the two underlying structures that produced the data.

function (see Fig. 1.30):

$$F(x,y,t) = \exp(-x^2 - 0.5y^2)\cos(2t) + \operatorname{sech}(x)\tanh(x)\exp(-0.2y^2)\sin(t). \quad (1.58)$$

This model has two spatial modes with two distinct temporal frequencies, thus a two-factor model should be sufficient to extract the underlying spatial and temporal modes. To construct this function, Code 1.13 is used.

Code 1.13: [MATLAB] Creating tensor data.

```
x=-5:0.1:5; y=-6:0.1:6; t=0:0.1:10*pi;
[X,Y,T]=meshgrid(x,y,t);
A=exp(-(X.^2+0.5*Y.^2)).*(cos(2*T))+ ...
    (sech(X).*tanh(X).*exp(-0.2*Y.^2)).*sin(T);
```

Code 1.13: [Python] Creating tensor data.

```
x = np.arange(-5,5.01,0.1)
y = np.arange(-6,6.01,0.1)
t = np.arange(0,10*np.pi+0.1,0.1)
X,Y,T = np.meshgrid(x,y,t)
A = np.exp(-(X**2 + 0.5*Y**2)) * np.cos(2*T) +      (np.
   divide(np.ones_like(X),np.cosh(X)) * np.tanh(X) * np.exp
    (-0.2*Y**2)) * np.sin(T)
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

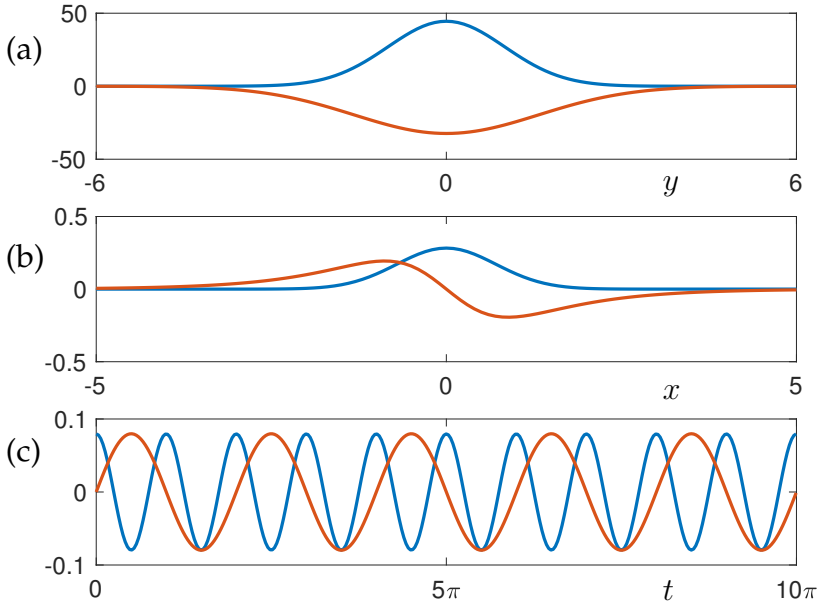


Figure 1.31: Three-way tensor decomposition of the function (1.58) discretized so that the data matrix is $\mathbf{A} \in \mathbb{R}^{121\times101\times315}$. A CP tensor decomposition can be used to extract the two underlying structures that produced the data. The first factor is in blue, and the second factor is in red. The three distinct directions of the data (parallel factors) are illustrated in (a) the y direction, (b) the x direction, and (c) the time t.

Note that the **meshgrid** command is capable of generating N-way arrays. Indeed, MATLAB and Python have no difficulties specifying higher-dimensional arrays and tensors. Specifically, one can easily generate N-way data matrices with arbitrary dimensions. The MATLAB command $\mathbf{A} = \mathbf{randn}(\mathbf{10}, \mathbf{10}, \mathbf{10}, \mathbf{10}, \mathbf{10})$ generates a five-way hypercube with random values in each of the five directions of the array.

Figure 1.30 shows eight snapshots of the function (1.58) discretized with the code above. The N-way array data generated from the MATLAB code produces $\mathbf{A} \in \mathbb{R}^{121\times101\times315}$, which is of total dimension 10^6.The CP tensor decomposition can be used to extract a two-factor model for this three-way array, thus producing two vectors in each direction of space x, space y, and time t.

The N-way toolbox provides a simple architecture for performing tensor decompositions. The PARAFAC command structure can easily take the input function (1.58), which is discretized in the code above, and provide a two-factor model. Code 1.14 produces the tensor model.

Code 1.14: [MATLAB] Two-factor tensor model.

```
model=parafac(A,2);
[A1,A2,A3]=fac2let(model);
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Code 1.14: [Python] Two-factor tensor model.

```
from tensorly.decomposition import parafac
A1, A2, A3 = parafac(A,2)
```

Note that in the above MATLAB code, the **fac2let** command turns the factors in the model into their component matrices. Further note that the **meshgrid** arrangement of the data is different from **parafac** since the x and y directions are switched.

Figure 1.31 shows the results of the N-way tensor decomposition for the prescribed two-factor model. Specifically, the two vectors along each of the three directions of the array are illustrated. For this example, the exact answer is known since the data was constructed from the rank-two model (1.58). The first set of two modes (along the original y direction) are Gaussian as prescribed. The second set of two modes (along the original x direction) include a Gaussian for the first function, and the antisymmetric $\mathrm{sech}(x)\tanh(x)$ for the second function. The third set of two modes correspond to the time dynamics of the two functions: $\cos(2t)$ and $\sin(t)$, respectively. Thus, the two-factor model produced by the CP tensor decomposition returns the expected, low-rank functions that produced the high-dimensional data matrix $\mathbf{A}$.

Recent theoretical and computational advances in N-way decompositions are opening up the potential for tensor decompositions in many fields. For N large, such decompositions can be computationally intractable due to the size of the data. Indeed, even in the simple example illustrated in Figs. 1.30 and 1.31, there are 10^6 data points. Ultimately, the CP tensor decomposition does not scale well with additional data dimensions. However, randomized techniques are helping yield tractable computations even for large data sets [214, 234]. As with the SVD, randomized methods exploit the underlying low-rank structure of the data in order to produce an accurate approximation through the sum of rank-one outer products. Additionally, tensor decompositions can be combined with constraints on the form of the parallel factors in order to produce more easily interpretable results [464]. This gives a framework for producing interpretable and scalable computations of N-way data arrays.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Suggested Reading

Texts

(1) Matrix computations, by G. H. Golub and C. F. Van Loan, 2012 [287].

Papers and reviews

(1) Calculating the singular values and pseudo-inverse of a matrix, by G. H. Golub and W. Kahan, *Journal of the Society for Industrial & Applied Mathematics, Series B: Numerical Analysis*, 1965 [285].

(2) A low-dimensional procedure for the characterization of human faces, by L. Sirovich and M. Kirby, *Journal of the Optical Society of America A*, 1987 [664].

(3) Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions, by N. Halko, P.-G. Martinsson, and J. A. Tropp, *SIAM Review*, 2011 [309].

(4) A randomized algorithm for the decomposition of matrices, by P.-G. Martinsson, V. Rokhlin, and M. Tygert, *Applied and Computational Harmonic Analysis*, 2011 [488].

(5) The optimal hard threshold for singular values is $4/\sqrt{3}$, by M. Gavish and D. L. Donoho, *IEEE Transactions on Information Theory*, 2014 [267].

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Homework

Exercise 1-1. Load the image dog.jpg and compute the full SVD. Choose a rank $r < m$ and confirm that the matrix $\mathbf{U}^*\mathbf{U}$ is the $r \times r$ identity matrix. Now confirm that $\mathbf{U}\mathbf{U}^*$ is *not* the identity matrix. Compute the norm of the error between $\mathbf{U}\mathbf{U}^*$ and the $n \times n$ identity matrix as the rank r varies from 1 to n and plot the error.

Exercise 1-2. Load the image dog.jpg and compute the economy SVD. Compute the relative reconstruction error of the truncated SVD in the Frobenius norm as a function of the rank r. Square this error to compute the fraction of missing variance as a function of r. You may also decide to plot 1 minus the error or missing variance to visualize the amount of norm or variance captured at a given rank r. Plot these quantities along with the cumulative sum of singular values as a function of r. Find the rank r where the reconstruction captures 99% of the total variance. Compare this with the rank r where the reconstruction captures 99% in the Frobenius norm and with the rank r that captures 99% of the cumulative sum of singular values.

Exercise 1-3. Load the Yale B image database and compute the economy SVD using a standard `svd` command. Now compute the SVD with the method of snapshots. Compare the singular value spectra on a log plot. Compare the first 10 left singular vectors using each method (remember to reshape them into the shape of a face). Now compare a few singular vectors farther down the spectrum. Explain your findings.

Exercise 1-4. Generate a random 100×100 matrix, i.e., a matrix whose entries are sampled from a normal distribution. Compute the SVD of this matrix and plot the singular values. Repeat this 100 times and plot the distribution of singular values in a box-and-whisker plot. Plot the mean and median singular values as a function of r. Now repeat this for different matrix sizes (e.g., 50×50, 200×200, 500×500, 1000×1000, etc.).

Exercise 1-5. Compare the singular value distributions for a 1000×1000 uniformly distributed random matrix and a Gaussian random matrix of the same size. Adapt the Gavish–Donoho algorithm to filter uniform noise based on this singular value distribution. Add uniform noise to a data set (either an image or the test low-rank signal) and apply this thresholding algorithm to filter the noise. Vary the magnitude of the noise and compare the results. Is the filtering good or bad?

Exercise 1-6. This exercise will test the concept of condition number. We will

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

test the accuracy of solving $\mathbf{Ax} = \mathbf{b}$ when noise is added to $\mathbf{b}$ for matrices $\mathbf{A}$ with different condition numbers.

(a) To build the two matrices, generate a random $\mathbf{U} \in \mathbb{R}^{100\times100}$ and $\mathbf{V} \in \mathbb{R}^{100\times100}$ and then create two $\mathbf{\Sigma}$ matrices: the first $\mathbf{\Sigma}$ will have singular values spaced logarithmically from 100 to 1, and the second $\mathbf{\Sigma}$ will have singular values spaced logarithmically from 100 to 10^{-6}. Use these matrices to create two $\mathbf{A}$ matrices, one with a condition number of 100 and the other with a condition number of 100 million. Now create a random $\mathbf{b}$ vector, solve for $\mathbf{x}$ using the two methods, and compare the results. Add a small $\boldsymbol{\epsilon}$ to $\mathbf{b}$, with norm 10^{-6} smaller than the norm of $\mathbf{b}$. Now solve for $\mathbf{x}$ using this new $\mathbf{b} + \boldsymbol{\epsilon}$ and compare the results.

(b) Now repeat the experiment above with many different noise vectors $\boldsymbol{\epsilon}$ and compute the distribution of the error; plot this error as a histogram and explain the shape.

(c) Repeat the above experiment comparing two $\mathbf{A}$ matrices with different singular value distributions: the first $\mathbf{\Sigma}$ will have values spaced linearly from 100 to 1 and the second $\mathbf{\Sigma}$ will have value spaced logarithmically from 100 to 1. Does anything change? Please explain why yes or no.

(d) Repeat the above experiment, but now with an $\mathbf{A}$ matrix that has size 100×10. Explain any changes.

Exercise 1-7. Load the data set for fluid flow past a cylinder (you can either download this from our book `http://DMDbook.com` or generate it using the IBPM code on GitHub). Each column is a flow field that has been reshaped into a vector.

(a) Compute the SVD of this data set and plot the singular value spectrum and the leading singular vectors. The $\mathbf{U}$ matrix contains eigenflow fields and the $\mathbf{\Sigma V}^*$ represents the amplitudes of these eigenflows as the flow evolves in time.

(b) Write a code to plot the reconstructed movie for various truncation values r. Compute the r value needed to capture 90%, 99%, and 99.9% of the flow energy based on the singular value spectrum (recall that energy is given by the Frobenius norm squared). Plot the movies for each of these truncation values and compare the fidelity. Also compute the squared Frobenius norm of the error between the true matrix $\mathbf{X}$ and the reconstructed matrix $\tilde{\mathbf{X}}$, where $\mathbf{X}$ is the flow field movie.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

(c) Fix a value $r = 10$ and compute the truncated SVD. Each column $\mathbf{w}_k \in \mathbb{R}^{10}$ of the matrix $\mathbf{W} = \tilde{\mathbf{\Sigma}}\tilde{\mathbf{V}}^*$ represents the mixture of the first 10 eigenflows in the kth column of $\mathbf{X}$. Verify this by comparing the kth snapshot of $\mathbf{X}$ with $\tilde{\mathbf{U}}\mathbf{w}_k$.

(d) Now, build a linear regression model for how the amplitudes $\mathbf{w}_k$ evolve in time. This will be a dynamical system:

$$\mathbf{w}_{k+1} = \mathbf{A}\mathbf{w}_k.$$

Create a matrix $\mathbf{W}$ with the first 1 through $m-1$ columns of $\mathbf{\Sigma V}^*$ and another matrix $\mathbf{W}'$ with the 2 through m columns of $\mathbf{\Sigma V}^*$. We will now try to solve for a best-fit $\mathbf{A}$ matrix so that

$$\mathbf{W}' \approx \mathbf{AW}.$$

Compute the SVD of $\mathbf{W}$ and use this to compute the pseudo-inverse of $\mathbf{W}$ to solve for $\mathbf{A}$. Compute the eigenvalues of $\mathbf{A}$ and plot them in the complex plane.

(e) Use this $\mathbf{A}$ matrix to advance the state $\mathbf{w}_k = \mathbf{A}^{k-1}\mathbf{w}_1$ starting from $\mathbf{w}_1$. Plot the reconstructed flow field using these predicted amplitude vectors and compare with the true values.

This exercise derived the dynamic mode decomposition from Section 7.2.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Chapter 2

Fourier and Wavelet Transforms

A central concern of mathematical physics and engineering mathematics involves the transformation of equations into a coordinate system where expressions simplify, decouple, and are amenable to computation and analysis. This is a common theme throughout this book, in a wide variety of domains, including data analysis (e.g., the singular value decomposition, SVD), dynamical systems (e.g., spectral decomposition into eigenvalues and eigenvectors), and control (e.g., defining coordinate systems by controllability and observability). Perhaps the most foundational and ubiquitous coordinate transformation was introduced by J.-B. Joseph Fourier in the early 1800s to investigate the theory of heat [249]. Fourier introduced the concept that sine and cosine functions of increasing frequency provide an orthogonal *basis* for the space of solution functions. Indeed, the Fourier transform basis of sines and cosines are eigenfunctions of the heat equation, with the specific frequencies serving as the eigenvalues, determined by the geometry, and amplitudes determined by the boundary conditions.

Fourier's seminal work provided the mathematical foundation for Hilbert spaces, operator theory, approximation theory, and the subsequent revolution in analytical and computational mathematics. Fast forward 200 years, and the fast Fourier transform has become the cornerstone of computational mathematics, enabling real-time image and audio compression, global communication networks, modern devices and hardware, numerical physics and engineering at scale, and advanced data analysis. Simply put, the fast Fourier transform has had a more significant and profound role in shaping the modern world than any other algorithm to date.

With increasingly complex problems, data sets, and computational geometries, simple Fourier sine and cosine bases have given way to *tailored* bases, such as the data-driven SVD. In fact, the SVD basis can be used as a direct analogue of the Fourier basis for solving partial differential equations (PDEs) with complex geometries, as will be discussed later. In addition, related functions, called wavelets, have been developed for advanced signal processing and com-

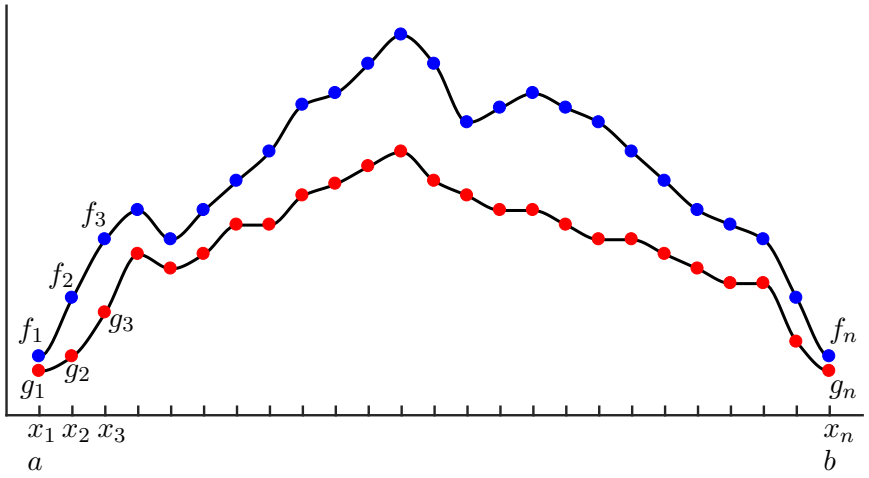


Figure 2.1: Discretized functions used to illustrate the inner product.

pression efforts. In this chapter, we will demonstrate a few of the many uses of Fourier and wavelet transforms.

2.1 Fourier Series and Fourier Transforms

Before describing the computational implementation of Fourier transforms on vectors of data, here we introduce the analytic Fourier series and Fourier transform, defined for continuous functions. Naturally, the discrete and continuous formulations should match in the limit of data with infinitely fine resolution. The Fourier series and transform are intimately related to the geometry of infinite-dimensional function spaces, or *Hilbert* spaces, which generalize the notion of vector spaces to include functions with infinitely many degrees of freedom. Thus, we begin with an introduction to function spaces.

Inner Products of Functions and Vectors

In this section, we will make use of inner products and norms of functions. In particular, we will use the common Hermitian inner product for functions $f(x)$ and $g(x)$ defined for x on a domain $x \in [a, b]$:

$$\langle f(x), g(x) \rangle = \int_a^b f(x)\bar{g}(x)\,\mathrm{d}x, \tag{2.1}$$

where $\bar{g}$ denotes the complex conjugate.

The inner product of functions may seem strange or unmotivated at first, but this definition becomes clear when we consider the inner product of vectors of data. In particular, if we discretize the functions $f(x)$ and $g(x)$ into vectors of data, as in Fig. 2.1, we would like the vector inner product to converge to

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

the function inner product as the sampling resolution is increased. The inner product of the data vectors $\mathbf{f} = \begin{bmatrix} f_1 & f_2 & \cdots & f_n \end{bmatrix}^T$ and $\mathbf{g} = \begin{bmatrix} g_1 & g_2 & \cdots & g_n \end{bmatrix}^T$ is defined by

$$\langle \mathbf{f}, \mathbf{g} \rangle = \mathbf{g}^* \mathbf{f} = \sum_{k=1}^{n} f_k \bar{g}_k = \sum_{k=1}^{n} f(x_k) \bar{g}(x_k). \tag{2.2}$$

The magnitude of this inner product will grow as more data points are added; i.e., as n increases. Thus, we may normalize by $\Delta x = (b-a)/(n-1)$, where $b = x_n$ and $a = x_1$:

$$\frac{b-a}{n-1} \langle \mathbf{f}, \mathbf{g} \rangle = \sum_{k=1}^{n} f(x_k) \bar{g}(x_k) \Delta x, \tag{2.3}$$

which is the Riemann approximation to the continuous function inner product in (2.1). It is now clear that as we take the limit of $n \to \infty$ (i.e., infinite data resolution, with $\Delta x \to 0$), the vector inner product converges to the inner product of functions in (2.1).

This inner product also induces a norm on functions, given by

$$\|f\|_2 = (\langle f, f \rangle)^{1/2} = \sqrt{\langle f, f \rangle} = \left(\int_a^b f(x) \bar{f}(x) \, \mathrm{d}x \right)^{1/2}. \tag{2.4}$$

The set of all functions with bounded norm defines the set of square integrable functions, denoted by $L^2([a,b])$; this is also known as the set of Lebesgue integrable functions. The interval $[a,b]$ may also be chosen to be infinite (e.g., $(-\infty, \infty)$), semi-infinite (e.g., $[a, \infty)$), or periodic (e.g., $[-\pi, \pi)$). A fun example of a function in $L^2([1,\infty))$ is $f(x) = 1/x$. The square of f has finite integral from 1 to ∞, although the integral of the function itself diverges. The shape obtained by rotating this function about the x-axis is known as Gabriel's horn, as the volume is finite (related to the integral of f^2), while the surface area is infinite (related to the integral of f).

As in finite-dimensional vector spaces, the inner product may be used to *project* a function into an new coordinate system defined by a basis of orthogonal functions. A Fourier series representation of a function f is precisely a projection of this function onto the orthogonal set of sine and cosine functions with integer period on the domain $[a,b]$. This is the subject of the following sections.

Fourier Series

A fundamental result in Fourier analysis is that if $f(x)$ is periodic and piecewise smooth, then it can be written in terms of a Fourier series, which is an infinite

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

sum of cosines and sines of increasing frequency. In particular, if $f(x)$ is 2π-periodic, it may be written as

$$f(x) = \frac{a_0}{2} + \sum_{k=1}^{\infty} (a_k \cos(kx) + b_k \sin(kx)). \tag{2.5}$$

The coefficients a_k and b_k are given by

$$a_k = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \cos(kx)\, \mathrm{d}x, \tag{2.6a}$$

$$b_k = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \sin(kx)\, \mathrm{d}x, \tag{2.6b}$$

which may be viewed as the coordinates obtained by projecting the function onto the orthogonal cosine and sine basis $\{\cos(kx), \sin(kx)\}_{k=0}^{\infty}$. In other words, the integrals in (2.6) may be rewritten in terms of the inner product as

$$a_k = \frac{1}{\|\cos(kx)\|^2} \langle f(x), \cos(kx) \rangle, \tag{2.7a}$$

$$b_k = \frac{1}{\|\sin(kx)\|^2} \langle f(x), \sin(kx) \rangle, \tag{2.7b}$$

where $\|\cos(kx)\|^2 = \|\sin(kx)\|^2 = \pi$. This factor of $1/\pi$ is easy to verify by numerically integrating $\cos(x)^2$ and $\sin(x)^2$ from $-\pi$ to π.

The Fourier series for an L-periodic function on $[0, L)$ is similarly given by

$$f(x) = \frac{a_0}{2} + \sum_{k=1}^{\infty} \left(a_k \cos\left(\frac{2\pi kx}{L}\right) + b_k \sin\left(\frac{2\pi kx}{L}\right) \right), \tag{2.8}$$

with coefficients a_k and b_k given by

$$a_k = \frac{2}{L} \int_0^L f(x) \cos\left(\frac{2\pi kx}{L}\right) \mathrm{d}x, \tag{2.9a}$$

$$b_k = \frac{2}{L} \int_0^L f(x) \sin\left(\frac{2\pi kx}{L}\right) \mathrm{d}x. \tag{2.9b}$$

Because we are expanding functions in terms of sine and cosine functions, it is also natural to use Euler's formula $e^{ikx} = \cos(kx) + i \sin(kx)$ to write a Fourier

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

series in complex form with complex coefficients $c_k = \alpha_k + i\beta_k$:

$$\begin{aligned} f(x) &= \sum_{k=-\infty}^{\infty} c_k e^{ikx} \\ &= \sum_{k=-\infty}^{\infty} (\alpha_k + i\beta_k)(\cos(kx) + i\sin(kx)) \\ &= (\alpha_0 + i\beta_0) + \sum_{k=1}^{\infty}[(\alpha_{-k} + \alpha_k)\cos(kx) + (\beta_{-k} - \beta_k)\sin(kx)] \\ &\quad + i\sum_{k=1}^{\infty}[(\beta_{-k} + \beta_k)\cos(kx) - (\alpha_{-k} - \alpha_k)\sin(kx)]. \end{aligned} \tag{2.10}$$

If $f(x)$ is real-valued, then $\alpha_{-k} = \alpha_k$ and $\beta_{-k} = -\beta_k$, so that $c_{-k} = \bar{c}_k$.

Thus, the functions $\psi_k = e^{ikx}$ for $k \in \mathbb{Z}$ (i.e., for integer k) provide a basis for periodic, complex-valued functions on an interval $[0, 2\pi)$. It is simple to see that these functions are orthogonal:

$$\langle \psi_j, \psi_k \rangle = \int_{-\pi}^{\pi} e^{ijx} e^{-ikx}\, \mathrm{d}x = \int_{-\pi}^{\pi} e^{i(j-k)x}\, \mathrm{d}x = \left[\frac{e^{i(j-k)x}}{i(j-k)}\right]_{-\pi}^{\pi} = \begin{cases} 0 & \text{if } j \neq k, \\ 2\pi & \text{if } j = k. \end{cases}$$

So $\langle \psi_j, \psi_k \rangle = 2\pi\delta_{jk}$, where δ is the Kronecker delta function. Similarly, the functions $e^{i2\pi kx/L}$ provide a basis for $L^2([0, L))$, the space of square integrable functions defined on $x \in [0, L)$.

In principle, a Fourier series is just a change of coordinates of a function $f(x)$ into an infinite-dimensional orthogonal function space spanned by sines and cosines (i.e., $\psi_k = e^{ikx} = \cos(kx) + i\sin(kx)$):

$$f(x) = \sum_{k=-\infty}^{\infty} c_k \psi_k(x) = \frac{1}{2\pi} \sum_{k=-\infty}^{\infty} \langle f(x), \psi_k(x) \rangle \psi_k(x). \tag{2.11}$$

The coefficients are given by $c_k = (1/(2\pi))\langle f(x), \psi_k(x) \rangle$. The factor of $1/(2\pi)$ normalizes the projection by the square of the norm of ψ_k, i.e., $\|\psi_k\|^2 = 2\pi$. This is consistent with our standard finite-dimensional notion of change of basis, as in Fig. 2.2. A vector $\vec{f}$ may be written in the $(\vec{x}, \vec{y})$ or $(\vec{u}, \vec{v})$ coordinate systems, via projection onto these orthogonal bases:

$$\vec{f} = \langle \vec{f}, \vec{x} \rangle \frac{\vec{x}}{\|\vec{x}\|^2} + \langle \vec{f}, \vec{y} \rangle \frac{\vec{y}}{\|\vec{y}\|^2} \tag{2.12a}$$

$$= \langle \vec{f}, \vec{u} \rangle \frac{\vec{u}}{\|\vec{u}\|^2} + \langle \vec{f}, \vec{v} \rangle \frac{\vec{v}}{\|\vec{v}\|^2}. \tag{2.12b}$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

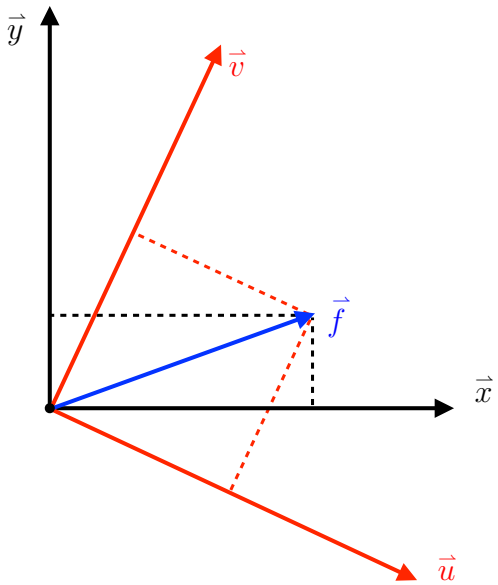


Figure 2.2: Change of coordinates of a vector in two dimensions.

Example: Fourier Series for a Continuous Hat Function

As a simple example, we demonstrate the use of Fourier series to approximate a continuous hat function, defined from $-\pi$ to π:

$$f(x) = \begin{cases} 0 & \text{for } x \in [-\pi, \pi/2), \\ 1 + 2x/\pi & \text{for } x \in [-\pi/2, 0), \\ 1 - 2x/\pi & \text{for } x \in [0, \pi/2), \\ 0 & \text{for } x \in [\pi/2, \pi). \end{cases} \tag{2.13}$$

Because this function is even, it may be approximated with cosines alone. The Fourier series for $f(x)$ is shown in Fig. 2.3 for an increasing number of cosines.

Figure 2.4 shows the coefficients a_k of the even cosine functions, along with the approximation error, for an increasing number of modes. The error decreases monotonically, as expected. The coefficients b_k corresponding to the odd sine functions are not shown, as they are identically zero since the hat function is even.

Code 2.1: [MATLAB] Fourier series approximation to a hat function.

```
% Define domain
dx = 0.001;
L = pi;
x = (-1+dx:dx:1)*L;
n = length(x);    nquart = floor(n/4);

% Define hat function
f = 0*x;
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

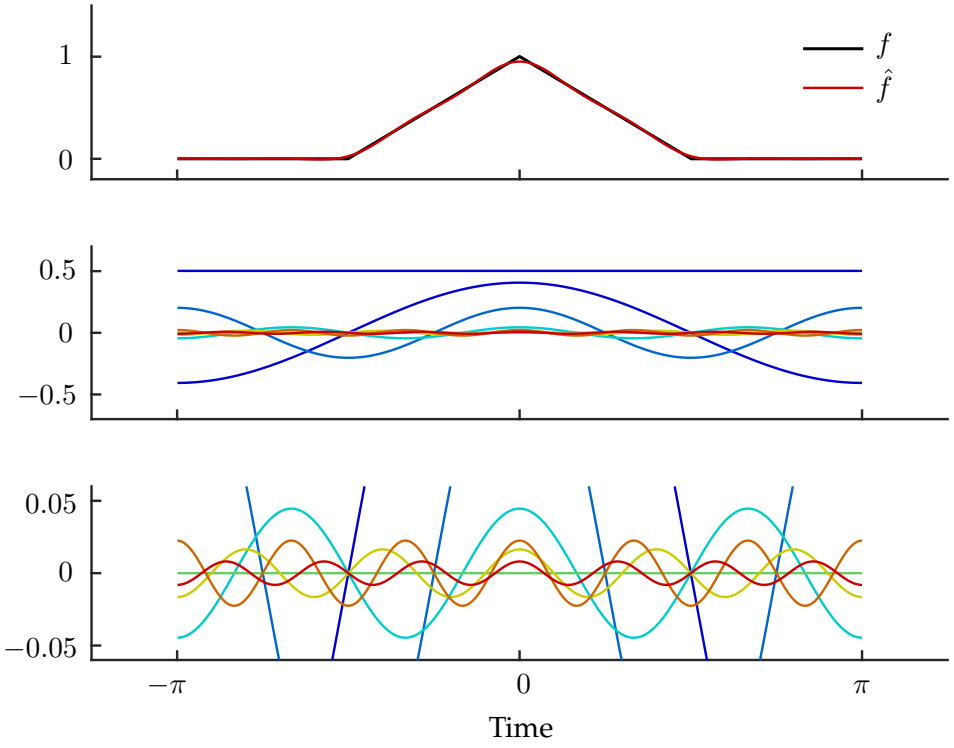


Figure 2.3: (top) Hat function and Fourier cosine series approximation for $n = 7$. (middle) Fourier cosines used to approximate the hat function. (bottom) Zoom in of modes with small amplitude and high frequency.

```
f(nquart:2*nquart) = 4*(1:nquart+1)/n;
f(2*nquart+1:3*nquart) = 1-4*(0:nquart-1)/n;
plot(x,f,'-k','LineWidth',1.5), hold on

% Compute Fourier series
CC = jet(20);
A0 = sum(f.*ones(size(x)))*dx;
fFS = A0/2;
for k=1:20
    A(k) = sum(f.*cos(pi*k*x/L))*dx; % Inner product
    B(k) = sum(f.*sin(pi*k*x/L))*dx;
    fFS = fFS + A(k)*cos(k*pi*x/L) + B(k)*sin(k*pi*x/L);
    plot(x,fFS,'-','Color',CC(k,:),'LineWidth',1.2)
end
```

Code 2.1: [Python] Fourier series approximation to a hat function.

```
# Define domain
dx = 0.001
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

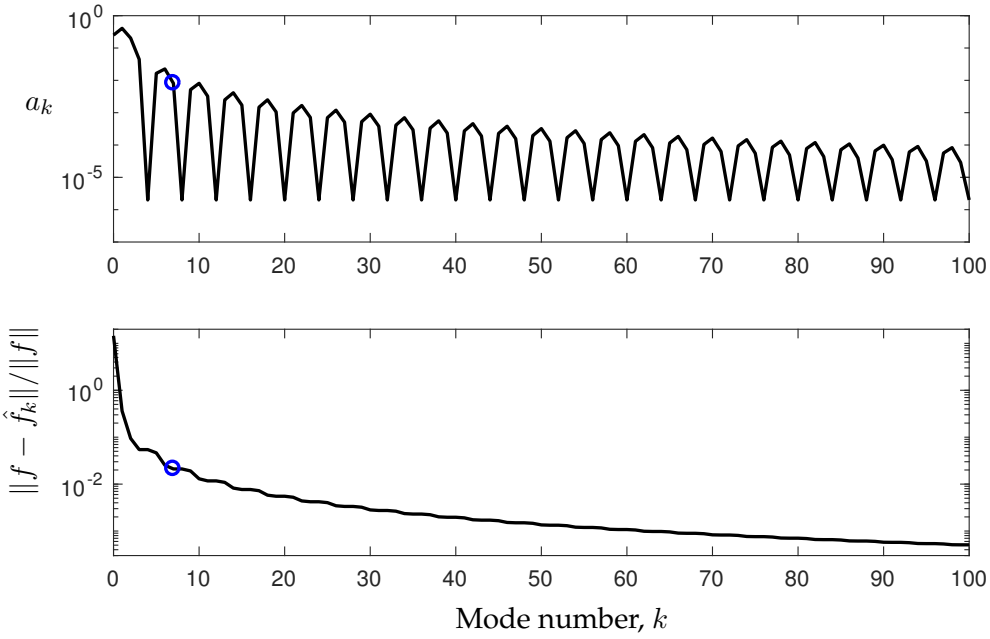


Figure 2.4: Fourier coefficients (top) and relative error of Fourier cosine approximation with true function (bottom) for the hat function in Fig. 2.3. The $n = 7$ approximation is highlighted with a blue circle.

```
L = np.pi
x = L * np.arange(-1+dx,1+dx,dx)
n = len(x)
nquart = int(np.floor(n/4))

# Define hat function
f = np.zeros_like(x)
f[nquart:2*nquart] = (4/n)*np.arange(1,nquart+1)
f[2*nquart:3*nquart] = np.ones(nquart) - (4/n)*np.arange(0,
    nquart)

# Compute Fourier series
A0 = np.sum(f * np.ones_like(x)) * dx
fFS = A0/2

A = np.zeros(20)
B = np.zeros(20)
for k in range(20):
    A[k] = np.sum(f * np.cos(np.pi*(k+1)*x/L)) * dx # Inner
        product
    B[k] = np.sum(f * np.sin(np.pi*(k+1)*x/L)) * dx
    fFS = fFS + A[k]*np.cos((k+1)*np.pi*x/L) + B[k]*np.sin((
        k+1)*np.pi*x/L)
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

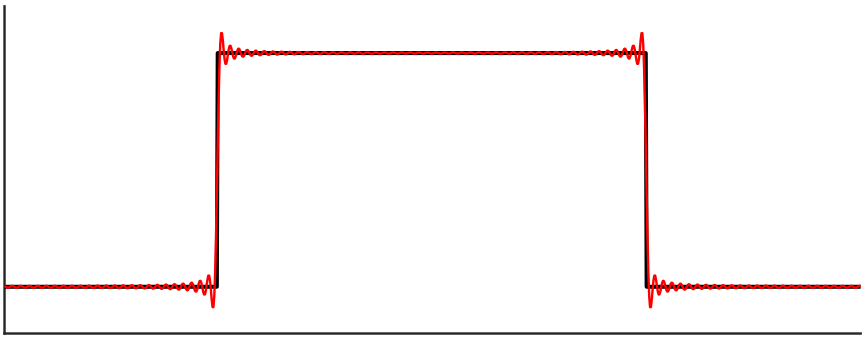


Figure 2.5: Gibbs phenomenon is characterized by high-frequency oscillations near discontinuities. The black curve is discontinuous, and the red curve is the Fourier approximation.

```
    ax.plot(x,fFS,'-')
```

Example: Fourier Series for a Discontinuous Hat Function

We now consider the discontinuous square hat function, defined on $[0, L)$, shown in Fig. 2.5. The function is given by:

$$f(x) = \begin{cases} 0 & \text{for } x \in [0, L/4), \\ 1 & \text{for } x \in [L/4, 3L/4), \\ 0 & \text{for } x \in [3L/4, L). \end{cases} \tag{2.14}$$

The truncated Fourier series is plagued by ringing oscillations, known as the Gibbs phenomenon, around the sharp corners of the step function. This example highlights the challenge of applying the Fourier series to discontinuous functions. The code to reproduce this example is available on the book's GitHub.

Fourier Transform

The Fourier series is defined for periodic functions, so that, outside the domain of definition, the function repeats itself forever. The Fourier transform integral is essentially the limit of a Fourier series as the length of the domain goes to infinity, which allows us to define a function defined on $(-\infty, \infty)$ without repeating, as shown in Fig. 2.6. We will consider the Fourier series on a domain $x \in [-L, L)$, and then let $L \to \infty$. On this domain, the Fourier series is

$$f(x) = \frac{a_0}{2} + \sum_{k=1}^{\infty} \left[a_k \cos\left(\frac{k\pi x}{L}\right) + b_k \sin\left(\frac{k\pi x}{L}\right) \right] = \sum_{k=-\infty}^{\infty} c_k e^{ik\pi x/L} \tag{2.15}$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

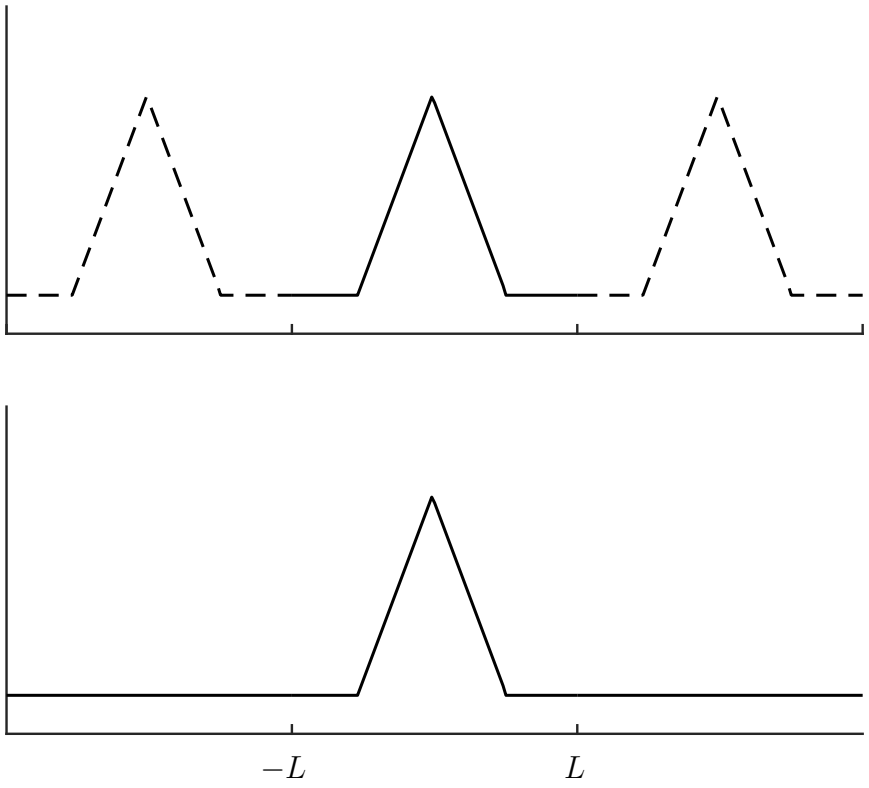


Figure 2.6: (top) Fourier series is only valid for a function that is periodic on the domain $[-L, L)$. (bottom) The Fourier transform is valid for generic non-periodic functions.

with the coefficients given by

$$c_k = \frac{1}{2L}\langle f(x), \psi_k \rangle = \frac{1}{2L}\int_{-L}^{L} f(x) e^{-ik\pi x/L}\,\mathrm{d}x. \tag{2.16}$$

Restating the previous results, $f(x)$ is now represented by a sum of sines and cosines with a discrete set of frequencies given by $\omega_k = k\pi/L$. Taking the limit as $L \to \infty$, these discrete frequencies become a continuous range of frequencies. Define $\omega = k\pi/L$, $\Delta\omega = \pi/L$, and take the limit $L \to \infty$, so that $\Delta\omega \to 0$:

$$f(x) = \lim_{\Delta\omega \to 0} \sum_{k=-\infty}^{\infty} \frac{\Delta\omega}{2\pi} \underbrace{\int_{-\pi/\Delta\omega}^{\pi/\Delta\omega} f(\xi) e^{-ik\Delta\omega\xi}\,\mathrm{d}\xi}_{\langle f(x), \psi_k(x)\rangle} e^{ik\Delta\omega x}. \tag{2.17}$$

When we take the limit, the expression $\langle f(x), \psi_k(x)\rangle$ will become the Fourier transform of $f(x)$, denoted by $\hat{f}(\omega) \triangleq \mathcal{F}(f(x))$. In addition, the summation

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

with weight $\Delta\omega$ becomes a Riemann integral, resulting in the following:

$$\hat{f}(\omega) = \mathcal{F}(f(x)) = \int_{-\infty}^{\infty} f(x)e^{-i\omega x}\,\mathrm{d}x \tag{2.18a}$$

$$f(x) = \mathcal{F}^{-1}(\hat{f}(\omega)) = \frac{1}{2\pi}\int_{-\infty}^{\infty} \hat{f}(\omega)e^{i\omega x}\,\mathrm{d}\omega. \tag{2.18b}$$

These two integrals are known as the *Fourier transform pair*. Both integrals converge as long as $\int_{-\infty}^{\infty} |f(x)|\,\mathrm{d}x < \infty$ and $\int_{-\infty}^{\infty} |\hat{f}(\omega)|\,\mathrm{d}\omega < \infty$; i.e., as long as both functions belong to the space of Lebesgue integrable functions, $f, \hat{f} \in L^1[(-\infty, \infty)]$.

The Fourier transform is particularly useful because of a number of properties, including linearity, and how derivatives of functions behave in the Fourier transform domain. These properties have been used extensively for data analysis and scientific computing (e.g., to solve PDEs accurately and efficiently), as will be explored throughout this chapter.

Derivatives of Functions

The Fourier transform of the derivative of a function is given by

$$\mathcal{F}\left(\frac{\mathrm{d}}{\mathrm{d}x}f(x)\right) = \int_{-\infty}^{\infty} \overbrace{f'(x)}^{dv}\overbrace{e^{-i\omega x}}^{u}\,\mathrm{d}x \tag{2.19a}$$

$$= \Big[\underbrace{f(x)e^{-i\omega x}}_{uv}\Big]_{-\infty}^{\infty} - \int_{-\infty}^{\infty} \underbrace{f(x)}_{v}\Big[\underbrace{-i\omega e^{-i\omega x}}_{du}\Big]\,\mathrm{d}x \tag{2.19b}$$

$$= i\omega\int_{-\infty}^{\infty} f(x)e^{-i\omega x}\,\mathrm{d}x \tag{2.19c}$$

$$= i\omega\mathcal{F}(f(x)). \tag{2.19d}$$

The formula for the Fourier transform of a higher derivative is given by

$$\mathcal{F}\left(\frac{\mathrm{d}^n}{\mathrm{d}x^n}f(x)\right) = i^n\omega^n\mathcal{F}(f(x)). \tag{2.20}$$

This is an extremely important property of the Fourier transform, as it will allow us to turn PDEs into ordinary differential equations (ODEs), closely related to the separation of variables:

$$\underset{\text{(PDE)}}{u_{tt} = cu_{xx}} \quad \xrightarrow{\mathcal{F}} \quad \underset{\text{(ODE)}}{\hat{u}_{tt} = -c\omega^2\hat{u}}. \tag{2.21}$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Linearity of Fourier Transforms

The Fourier transform is a linear operator, so that

$$\mathcal{F}(\alpha f(x) + \beta g(x)) = \alpha \mathcal{F}(f) + \beta \mathcal{F}(g) \tag{2.22}$$

and

$$\mathcal{F}^{-1}(\alpha \hat{f}(\omega) + \beta \hat{g}(\omega)) = \alpha \mathcal{F}^{-1}(\hat{f}) + \beta \mathcal{F}^{-1}(\hat{g}). \tag{2.23}$$

Parseval's Theorem

$$\int_{-\infty}^{\infty} |\hat{f}(\omega)|^2 \, \mathrm{d}\omega = 2\pi \int_{-\infty}^{\infty} |f(x)|^2 \, \mathrm{d}x. \tag{2.24}$$

In other words, the Fourier transform preserves the L_2-norm, up to a constant. This is closely related to unitarity, so that two functions will retain the same inner product before and after the Fourier transform. This property is useful for approximation and truncation, providing the ability to bound error at a given truncation.

Convolution

The convolution of two functions is particularly well behaved in the Fourier domain, being the product of the two Fourier-transformed functions. We define the convolution of two functions $f(x)$ and $g(x)$ as $f * g$:

$$(f * g)(x) = \int_{-\infty}^{\infty} f(x - \xi) g(\xi) \, \mathrm{d}\xi. \tag{2.25}$$

If we let $\hat{f} = \mathcal{F}(f)$ and $\hat{g} = \mathcal{F}(g)$, then

$$\begin{aligned}
\mathcal{F}^{-1}(\hat{f}\hat{g})(x) &= \frac{1}{2\pi} \int_{-\infty}^{\infty} \hat{f}(\omega) \hat{g}(\omega) e^{i\omega x} \, \mathrm{d}\omega && \text{(2.26a)} \\
&= \int_{-\infty}^{\infty} \hat{f}(\omega) e^{i\omega x} \left(\frac{1}{2\pi} \int_{-\infty}^{\infty} g(y) e^{-i\omega y} \, \mathrm{d}y \right) \mathrm{d}\omega && \text{(2.26b)} \\
&= \frac{1}{2\pi} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} g(y) \hat{f}(\omega) e^{i\omega(x-y)} \, \mathrm{d}\omega \, \mathrm{d}y && \text{(2.26c)} \\
&= \int_{-\infty}^{\infty} g(y) \Bigg(\underbrace{\frac{1}{2\pi} \int_{-\infty}^{\infty} \hat{f}(\omega) e^{i\omega(x-y)} \, \mathrm{d}\omega}_{f(x-y)} \Bigg) \mathrm{d}y && \text{(2.26d)} \\
&= \int_{-\infty}^{\infty} g(y) f(x - y) \, \mathrm{d}y = g * f = f * g. && \text{(2.26e)}
\end{aligned}$$

Thus, multiplying functions in the frequency domain is the same as convolving functions in the spatial domain. This will be particularly useful for control systems and transfer functions with the related Laplace transform.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

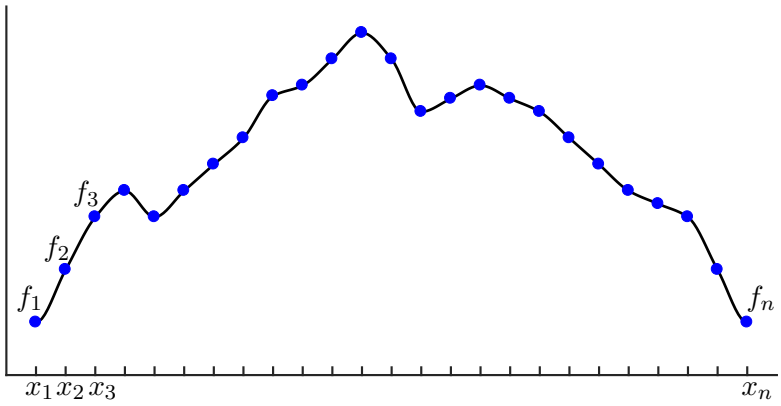


Figure 2.7: Discrete data sampled for the discrete Fourier transform.

2.2 Discrete Fourier Transform (DFT) and Fast Fourier Transform (FFT)

Until now, we have considered the Fourier series and Fourier transform for continuous functions $f(x)$. However, when computing or working with real data, it is necessary to approximate the Fourier transform on discrete vectors of data. The resulting discrete Fourier transform (DFT) is essentially a discretized version of the Fourier series for vectors of data $\mathbf{f} = \begin{bmatrix} f_1 & f_2 & f_3 & \cdots & f_n \end{bmatrix}^T$ obtained by discretizing the function $f(x)$ at a regular spacing, Δx, as in Fig. 2.7.

The DFT is tremendously useful for numerical approximation and computation, but it does not scale well to very large $n \gg 1$, as the simple formulation involves multiplication by a dense $n \times n$ matrix, requiring $\mathcal{O}(n^2)$ operations. In 1965, James W. Cooley (IBM) and John W. Tukey (Princeton) developed the revolutionary *fast* Fourier transform (FFT) algorithm [182, 183] that scales as $\mathcal{O}(n \log(n))$. As n becomes very large, the $\log(n)$ component grows slowly, and the algorithm approaches a linear scaling. Their algorithm was based on a fractal symmetry in the Fourier transform that allows an n-dimensional DFT to be solved with a number of lower-dimensional DFT computations. Although the different computational scaling between the DFT and FFT implementations may seem like a small difference, the fast $\mathcal{O}(n \log(n))$ scaling is what enables the ubiquitous use of the FFT in real-time communication, based on audio and image compression [731].

It is important to note that Cooley and Tukey did not invent the idea of the FFT, as there were decades of prior work developing special cases, although they provided the general formulation that is currently used. Amazingly, the FFT algorithm was formulated by Gauss over 150 years earlier in 1805 to approximate the orbits of the asteroids Pallas and Juno from measurement data, as he required a highly accurate interpolation scheme [319]. As the computations were performed by Gauss in his head and on paper, he required a fast

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

algorithm, and developed the FFT. However, Gauss did not view this as a major breakthrough and his formulation only appeared later in 1866 in his compiled notes [265]. It is interesting to note that Gauss's discovery even pre-dates Fourier's announcement of the Fourier series expansion in 1807, which was later published in 1822 [248].

Discrete Fourier Transform

Although we will always use the FFT for computations, it is illustrative to begin with the simplest formulation of the DFT. The discrete Fourier transform is given by

$$\hat{f}_k = \sum_{j=0}^{n-1} f_j e^{-i2\pi jk/n}, \tag{2.27}$$

and the inverse discrete Fourier transform (iDFT) is given by

$$f_k = \frac{1}{n} \sum_{j=0}^{n-1} \hat{f}_j e^{i2\pi jk/n}. \tag{2.28}$$

Thus, the DFT is a linear operator (i.e., a *matrix*) that maps the data points in $\mathbf{f}$ to the frequency domain $\hat{\mathbf{f}}$:

$$\{f_1, f_2, \ldots, f_n\} \quad \overset{\text{DFT}}{\Longrightarrow} \quad \{\hat{f}_1, \hat{f}_2, \ldots \hat{f}_n\}. \tag{2.29}$$

For a given number of points n, the DFT represents the data using sine and cosine functions with integer multiples of a fundamental frequency, $\omega_n = e^{-2\pi i/n}$. The DFT may be computed by matrix multiplication:

$$\begin{bmatrix} \hat{f}_1 \\ \hat{f}_2 \\ \hat{f}_3 \\ \vdots \\ \hat{f}_n \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n & \omega_n^2 & \cdots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^4 & \cdots & \omega_n^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \cdots & \omega_n^{(n-1)^2} \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_n \end{bmatrix}. \tag{2.30}$$

The output vector $\hat{\mathbf{f}}$ contains the Fourier coefficients for the input vector $\mathbf{f}$, and the DFT matrix $\mathbf{F}$ is a unitary Vandermonde matrix. The matrix $\mathbf{F}$ is complex-valued, so the output $\hat{\mathbf{f}}$ has both a magnitude and a phase, which will both have useful physical interpretations.

The real part of the DFT matrix $\mathbf{F}$ is shown in Fig. 2.8 for $n = 256$. Code 2.2 generates and plots this matrix. It can be seen from this image that there is a hierarchical and highly symmetric multi-scale structure to $\mathbf{F}$. Each row and column is a cosine function with increasing frequency.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

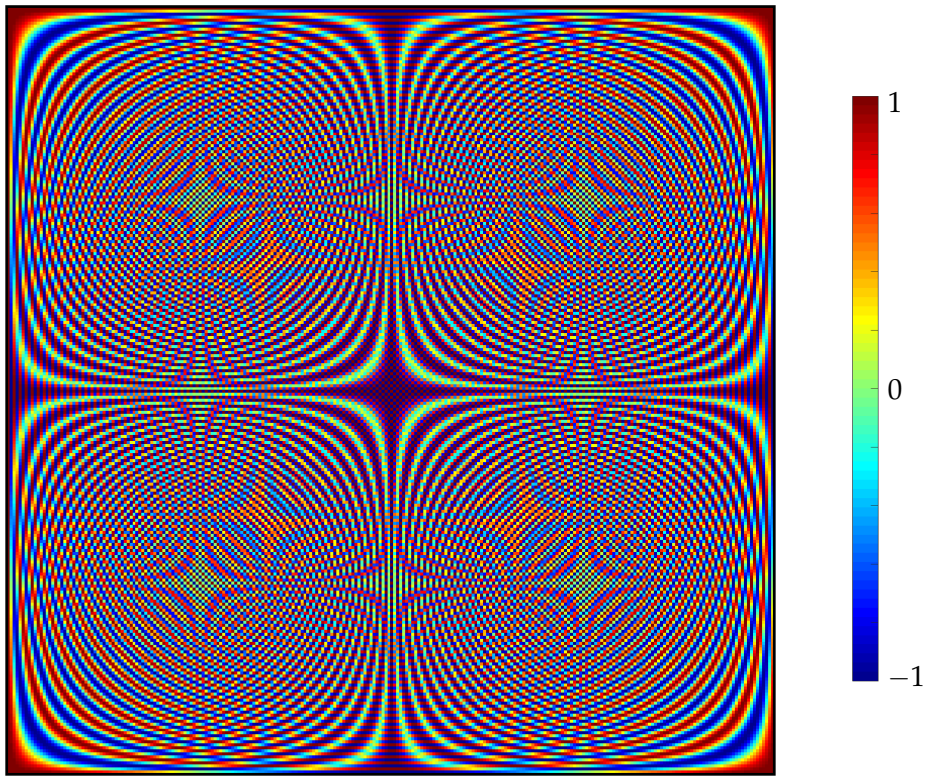


Figure 2.8: Real part of the DFT matrix for $n = 256$.

Code 2.2: [MATLAB] Generate discrete Fourier transform matrix.

```
n = 256;
w = exp(-i*2*pi/n);

% Slow
for i=1:n
    for j=1:n
        DFT(i,j) = w^((i-1)*(j-1));
    end
end

% Fast
[I,J] = meshgrid(1:n,1:n);
DFT = w.^((I-1).*(J-1));
imagesc(real(DFT))
```

Code 2.2: [Python] Generate discrete Fourier transform matrix.

```
n = 256
w = np.exp(-1j * 2 * np.pi / n)
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

```
# Slow
for i in range(n):
    for k in range(n):
        DFT[i,k] = w**(i*k)

DFT = np.real(DFT)

# Fast
J,K = np.meshgrid(np.arange(n),np.arange(n))
DFT = np.power(w,J*K)
DFT = np.real(DFT)
```

Fast Fourier Transform

As mentioned earlier, multiplying by the DFT matrix $\mathbf{F}$ involves $\mathcal{O}(n^2)$ operations. The fast Fourier transform scales as $\mathcal{O}(n\log(n))$, enabling a tremendous range of applications, including audio and image compression in MP3 and JPG formats, streaming video, satellite communications, and the cellular network, to name only a few of the myriad applications. For example, audio is generally sampled at $44.1\,\text{kHz}$, or $44\,100$ samples per second. For $10\,\text{s}$ of audio, the vector $\mathbf{f}$ will have dimension $n = 4.41 \times 10^5$. Computing the DFT using matrix multiplication involves approximately 2×10^{11}, or 200 billion, multiplications. In contrast, the FFT requires approximately 6×10^6, which amounts to a speed-up factor of over $30\,000$. Thus, the FFT has become synonymous with the DFT, and FFT libraries are built in to nearly every device and operating system that performs digital signal processing.

To see the tremendous benefit of the FFT, consider the transmission, storage, and decoding of an audio signal. We will see later that many signals are highly compressible in the Fourier transform domain, meaning that most of the coefficients of $\hat{\mathbf{f}}$ are small and can be discarded. This enables much more efficient storage and transmission of the compressed signal, as only the nonzero Fourier coefficients must be transmitted. However, it is then necessary to rapidly encode and decode the compressed Fourier signal by computing the FFT and inverse FFT (iFFT). This is accomplished with the one-line MATLAB commands

```
>> fhat = fft(f);  % Fast Fourier transform
>> f = ifft(fhat); % Inverse fast Fourier transform
```

and Python commands

```
>>> fhat = np.fft.fft(f);  # Fast Fourier transform
>>> f = np.fft.ifft(fhat); # Inverse fast Fourier transform
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

The basic idea behind the FFT is that the DFT may be implemented much more efficiently if the number of data points n is a power of 2. For example, consider $n = 1024 = 2^{10}$. In this case, the DFT matrix $\mathbf{F}_{1024}$ may be written as

$$\hat{\mathbf{f}} = \mathbf{F}_{1024}\mathbf{f} = \begin{bmatrix} \mathbf{I}_{512} & \mathbf{D}_{512} \\ \mathbf{I}_{512} & -\mathbf{D}_{512} \end{bmatrix} \begin{bmatrix} \mathbf{F}_{512} & \mathbf{0} \\ \mathbf{0} & \mathbf{F}_{512} \end{bmatrix} \begin{bmatrix} \mathbf{f}_{\text{even}} \\ \mathbf{f}_{\text{odd}} \end{bmatrix}, \tag{2.31}$$

where $\mathbf{f}_{\text{even}}$ are the even index elements of $\mathbf{f}$, $\mathbf{f}_{\text{odd}}$ are the odd index elements of $\mathbf{f}$, $\mathbf{I}_{512}$ is the 512×512 identity matrix, and $\mathbf{D}_{512}$ is given by

$$\mathbf{D}_{512} = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & \omega & 0 & \cdots & 0 \\ 0 & 0 & \omega^2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \omega^{511} \end{bmatrix}. \tag{2.32}$$

This expression can be derived from a careful accounting and reorganization of the terms in (2.27) and (2.30). If $n = 2^p$, this process can be repeated, and $\mathbf{F}_{512}$ can be represented by $\mathbf{F}_{256}$, which can then be represented by $\mathbf{F}_{128} \rightarrow \mathbf{F}_{64} \rightarrow \mathbf{F}_{32} \rightarrow \cdots$. If $n \neq 2^p$, the vector can be padded with zeros until it is a power of 2. The FFT then involves an efficient interleaving of even and odd indices of sub-vectors of $\mathbf{f}$, and the computation of several smaller 2×2 DFT computations.

FFT Example: Noise Filtering

To gain familiarity with how to use and interpret the FFT, we will begin with a simple example that uses the FFT to de-noise a signal. We will consider a function of time $f(t)$:

$$f(t) = \sin(2\pi f_1 t) + \sin(2\pi f_2 t), \tag{2.33}$$

with frequencies $f_1 = 50$ and $f_2 = 120$. We then add a large amount of Gaussian white noise to this signal, as shown in the top panel of Fig. 2.9.

It is possible to compute the fast Fourier transform of this noisy signal using the **fft** command. The power spectral density (PSD) is the normalized squared magnitude of $\hat{\mathbf{f}}$, and indicates how much *power* the signal contains in each frequency. In Fig. 2.9 (middle), it is clear that the noisy signal contains two large peaks at 50 Hz and 120 Hz. It is possible to zero-out components that have power below a threshold to remove noise from the signal. After inverse transforming the filtered signal, we find the clean and filtered time series match quite well (Fig. 2.9, bottom). Code 2.3 performs each step and plots the results.

Code 2.3: [MATLAB] Fast Fourier transform to de-noise signal.

```
%% Create a simple signal with two frequencies
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

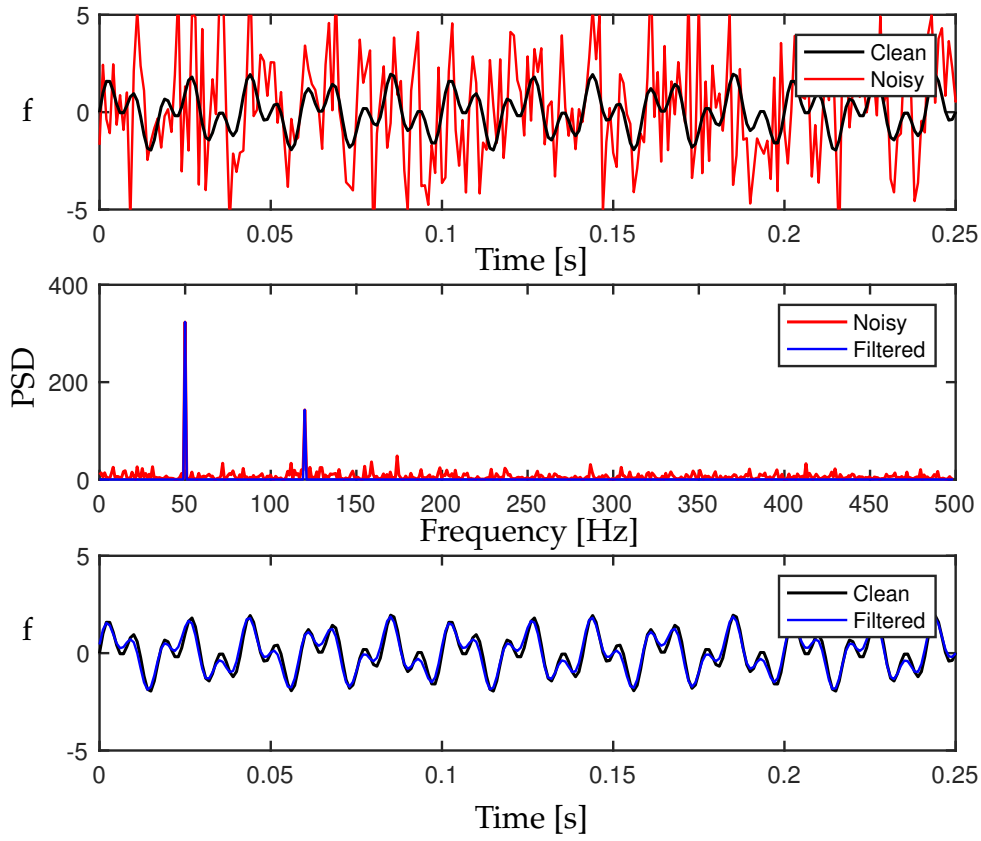


Figure 2.9: De-noising with FFT. (top) Noise is added to a simple signal given by a sum of two sine waves. (middle) In the Fourier domain, dominant peaks may be selected and the noise filtered. (bottom) The de-noised signal is obtained by inverse Fourier transforming the two dominant peaks.

```
dt = .001;
t = 0:dt:1;
f = sin(2*pi*50*t) + sin(2*pi*120*t); % Sum of 2 frequencies
f = f + 2.5*randn(size(t));  %  Add some noise

%% Compute the Fast Fourier Transform FFT
n = length(t);
fhat = fft(f,n);        % Compute the fast Fourier transform
PSD = fhat.*conj(fhat)/n; % Power spectrum (power per freq)
freq = 1/(dt*n)*(0:n); % Create x-axis of frequencies in Hz
L = 1:floor(n/2);    % Only plot the first half of freqs

%% Use the PSD to filter out noise
indices = PSD>100;   % Find all freqs with large power
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

```
PSDclean = PSD.*indices;  % Zero out all others
fhat = indices.*fhat;  % Zero out small Fourier coeffs. in Y
ffilt = ifft(fhat); % Inverse FFT for filtered time signal
```

Code 2.3: [Python] Fast Fourier transform to de-noise signal.

```
# Create a simple signal with two frequencies
dt = 0.001
t = np.arange(0,1,dt)
f = np.sin(2*np.pi*50*t) + np.sin(2*np.pi*120*t)
f_clean = f
f = f + 2.5*np.random.randn(len(t))      # Add some noise

## Compute the Fast Fourier Transform (FFT)
n = len(t)
fhat = np.fft.fft(f,n)                     # Compute the FFT
PSD = fhat * np.conj(fhat) / n             # Power spectrum
   (power per freq)
freq = (1/(dt*n)) * np.arange(n)           # Create x-axis
   of frequencies in Hz
L = np.arange(1,np.floor(n/2),dtype='int') # Only plot the
   first half of freqs

## Use the PSD to filter out noise
indices = PSD > 100       # Find all freqs with large power
PSDclean = PSD * indices  # Zero out all others
fhat = indices * fhat     # Zero out small Fourier coeffs.
   in Y
ffilt = np.fft.ifft(fhat) # Inverse FFT for filtered time
   signal
```

FFT Example: Spectral Derivatives

For the next example, we will demonstrate the use of the FFT for the fast and accurate computation of derivatives. As we saw in (2.19), the continuous Fourier transform has the property that $\mathcal{F}(\mathrm{d}f/\mathrm{d}x) = i\omega\mathcal{F}(f)$. Similarly, the numerical derivative of a vector of discretized data can be well approximated by multiplying each component of the discrete Fourier transform of the vector $\hat{\mathbf{f}}$ by $i\kappa$, where $\kappa = 2\pi k/n$ is the discrete wavenumber associated with that component. The accuracy and efficiency of the spectral derivative make it particularly useful for solving partial differential equations, as explored in the next section.

To demonstrate this so-called *spectral* derivative, we will start with a func-

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

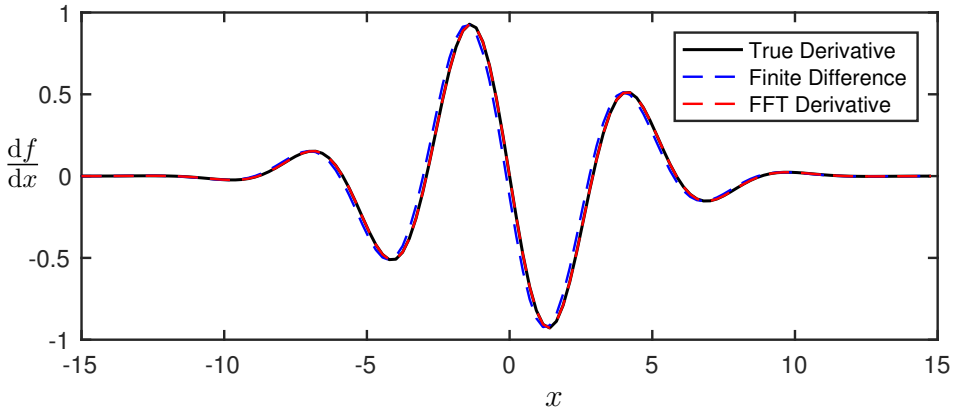


Figure 2.10: Comparison of the spectral derivative, computed using the FFT, with the finite-difference derivative.

tion $f(x)$ where we can compute the analytic derivative for comparison:

$$f(x) = \cos(x)e^{-x^2/25} \quad \Longrightarrow \quad \frac{\mathrm{d}f}{\mathrm{d}x}(x) = -\sin(x)e^{-x^2/25} - \frac{2}{25}xf(x). \tag{2.34}$$

Figure 2.10 compares the spectral derivative with the analytic derivative and the forward Euler finite-difference derivative using $n = 128$ discretization points:

$$\frac{\mathrm{d}f}{\mathrm{d}x}(x_k) \approx \frac{f(x_{k+1}) - f(x_k)}{\Delta x}. \tag{2.35}$$

The error of both differentiation schemes may be reduced by increasing n, which is the same as decreasing Δx. However, the error of the spectral derivative improves more rapidly with increasing n than finite-difference schemes, as shown in Fig. 2.11. The forward Euler differentiation is notoriously inaccurate, with error proportional to $\mathcal{O}(\Delta x)$; however, even increasing the order of a finite-difference scheme will not yield the same accuracy trend as the spectral derivative, which is effectively using information on the whole domain. Code 2.4 computes and compares the two differentiation schemes.

Code 2.4: [MATLAB] Fast Fourier transform to compute derivatives.

```
n = 128; L = 30; dx = L/(n);
x = -L/2:dx:L/2-dx;
f = cos(x).*exp(-x.^2/25);                    % Function
df = -(sin(x).*exp(-x.^2/25) + (2/25)*x.*f);  % Derivative
%% Derivative using FFT (spectral derivative)
fhat = fft(f);
kappa = (2*pi/L)*[-n/2:n/2-1];
kappa = fftshift(kappa);  % Re-order fft frequencies
dfhat = i*kappa.*fhat;
dfFFT = real(ifft(dfhat));
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

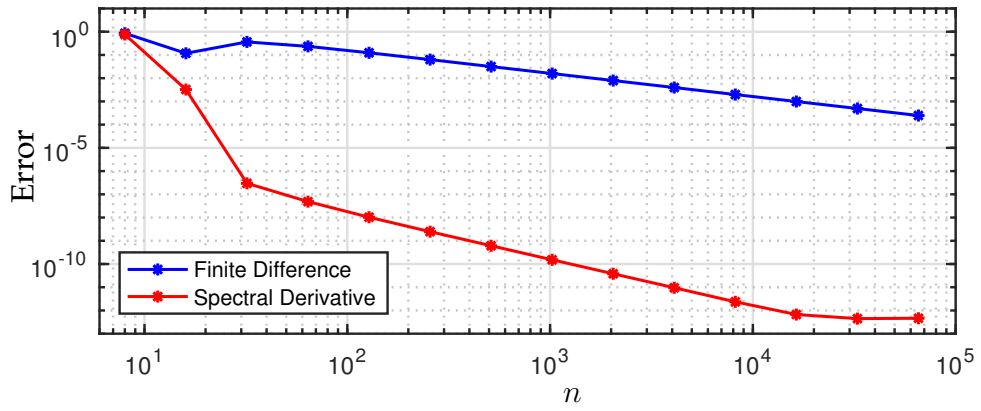


Figure 2.11: Benchmark of spectral derivative for varying data resolution.

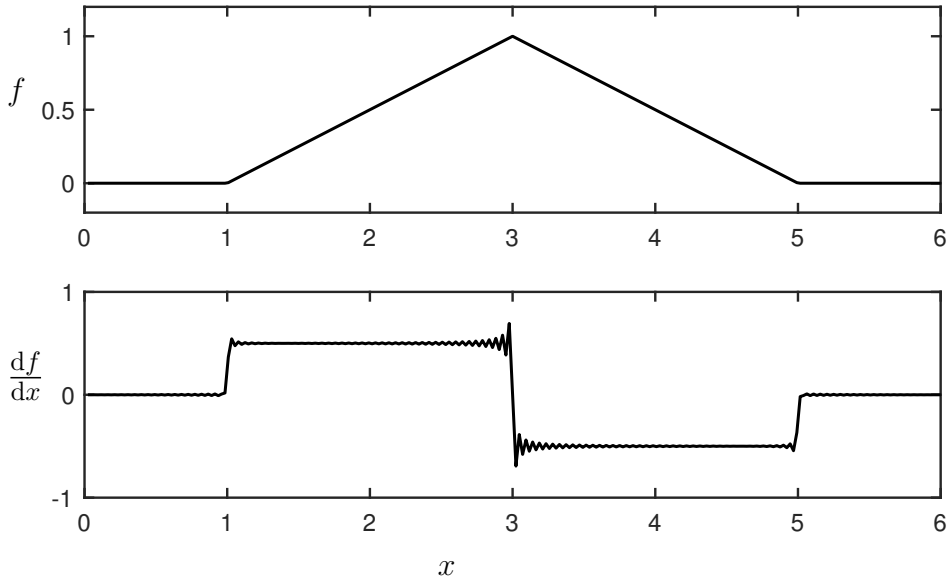


Figure 2.12: Gibbs phenomenon for the spectral derivative of a function with discontinuous derivative.

Code 2.4: [Python] Fast Fourier transform to compute derivatives.

```
## Derivative using FFT (spectral derivative)
fhat = np.fft.fft(f)
kappa = (2*np.pi/L)*np.arange(-n/2,n/2)
kappa = np.fft.fftshift(kappa) # Re-order fft frequencies
dfhat = kappa * fhat * (1j)
dfFFT = np.real(np.fft.ifft(dfhat))
```

If the derivative of a function is discontinuous, then the spectral derivative will exhibit the Gibbs phenomenon, as shown in Fig. 2.12.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

2.3 Transforming Partial Differential Equations

The Fourier transform was originally formulated in the 1800s as a change of coordinates for the heat equation into an eigenfunction coordinate system where the dynamics decouple. More generally, the Fourier transform is useful for transforming partial differential equations (PDEs) into ordinary differential equations (ODEs), as in (2.21). Here, we will demonstrate the utility of the FFT to numerically solve a number of PDEs. For an excellent treatment of spectral methods for PDEs, see Trefethen [710]; extensions also exist for stiff PDEs [377].

Heat Equation

The Fourier transform basis is ideally suited to solve the heat equation. In one spatial dimension, the heat equation is given by

$$u_t = \alpha^2 u_{xx}, \tag{2.36}$$

where $u(t, x)$ is the temperature distribution in time and space. If we Fourier-transform in space, then $\mathcal{F}(u(t, x)) = \hat{u}(t, \omega)$. The PDE in (2.36) becomes

$$\hat{u}_t = -\alpha^2 \omega^2 \hat{u}, \tag{2.37}$$

since the two spatial derivatives contribute $(i\omega)^2 = -\omega^2$ in the Fourier transform domain. Thus, by taking the Fourier transform, the PDE in (2.36) becomes an ODE for each fixed frequency ω. The solution is given by

$$\hat{u}(t, \omega) = e^{-\alpha^2 \omega^2 t} \hat{u}(0, \omega). \tag{2.38}$$

The function $\hat{u}(0, \omega)$ is the Fourier transform of the initial temperature distribution $u(0, x)$. It is now clear that higher frequencies, corresponding to larger values of ω, decay more rapidly as time evolves, so that sharp corners in the temperature distribution rapidly smooth out. We may take the inverse Fourier transform using the convolution property in (2.25), yielding

$$u(t, x) = \mathcal{F}^{-1}(\hat{u}(t, \omega)) = \mathcal{F}^{-1}(e^{-\alpha^2 \omega^2 t}) * u(0, x) = \frac{1}{2\alpha\sqrt{\pi t}} e^{-x^2/(4\alpha^2 t)} * u(0, x). \tag{2.39}$$

To simulate this PDE numerically, it is simpler and more accurate to first transform to the frequency domain using the FFT. In this case (2.37) becomes

$$\hat{u}_t = -\alpha^2 \kappa^2 \hat{u}, \tag{2.40}$$

where κ is the discretized frequency. It is important to use the **fftshift** command to reorder the wavenumbers according to the MATLAB convention.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

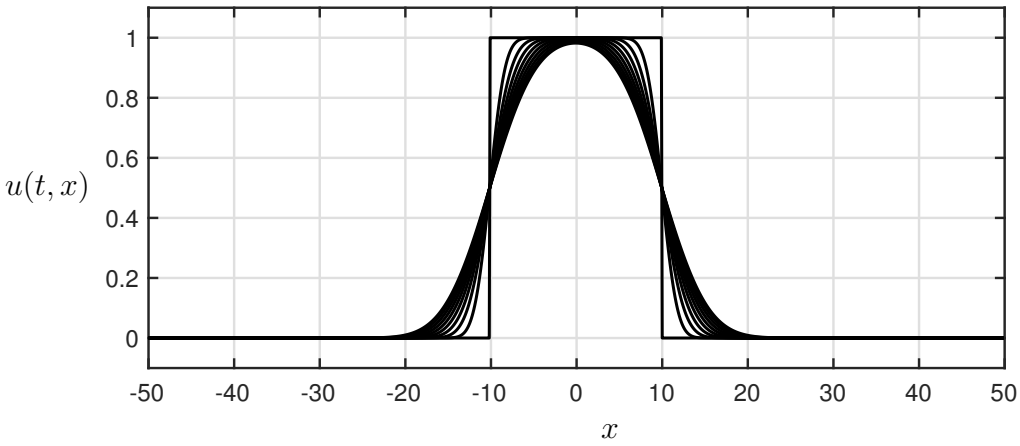


Figure 2.13: Solution of the 1D heat equation in time for an initial condition given by a square hat function. As time evolves, the sharp corners rapidly smooth and the solution approaches a Gaussian function.

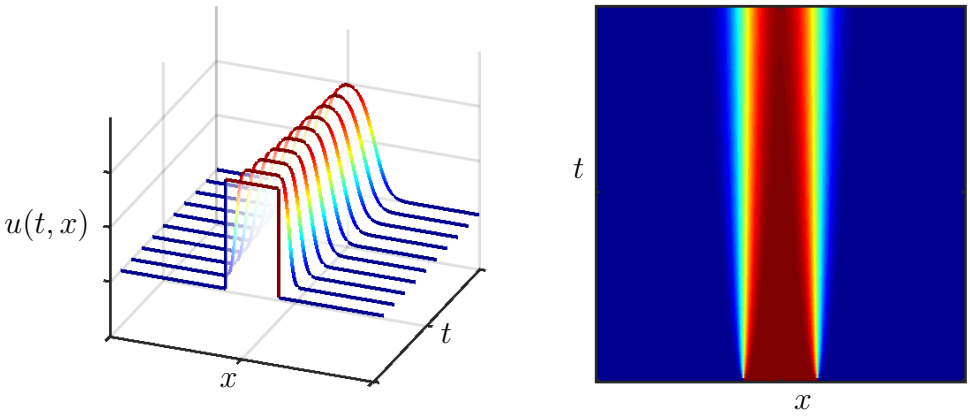


Figure 2.14: Evolution of the 1D heat equation in time, illustrated by a waterfall plot (left) and an x–t diagram (right).

Code 2.5 simulates the one-dimensional (1D) heat equation using the FFT, as shown in Figs. 2.13 and 2.14. In this example, because the PDE is linear, it is possible to advance the system using **ode45** directly in the frequency domain, using the vector field given in Code 2.6.

Figures 2.13 and 2.14 show several different views of the temperature distribution $u(t, x)$ as it evolves in time. Figure 2.13 shows the distribution at several times overlayed, and this same data is visualized in Fig. 2.14 in a waterfall plot (left) and in an x–t diagram (right). In all of the figures, it becomes clear that the sharp corners diffuse rapidly, as these correspond to the highest wavenumbers. Eventually, the lowest wavenumber variations will also decay, until the temperature reaches a constant steady-state distribution, which is a solution of

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Laplace's equation $u_{xx} = 0$. When solving this PDE using the FFT, we are implicitly assuming that the solution domain is periodic, so that the right and left boundaries are identified and the domain forms a ring. However, if the domain is large enough, then the effect of the boundaries is small.

Code 2.5: [MATLAB] Code to simulate the 1D heat equation using the Fourier transform.

```
a = 1;          % Thermal diffusivity constant
L = 100;        % Length of domain
N = 1000;       % Number of discretization points
dx = L/N;
x = -L/2:dx:L/2-dx; % Define x domain

% Define discrete wavenumbers
kappa = (2*pi/L)*[-N/2:N/2-1];
kappa = fftshift(kappa);    % Re-order fft wavenumbers

% Initial condition
u0 = 0*x;
u0((L/2 - L/10)/dx:(L/2 + L/10)/dx) = 1;

% Simulate in Fourier frequency domain
t = 0:0.1:10;
[t,uhat]=ode45(@(t,uhat)rhsHeat(t,uhat,kappa,a),t,fft(u0));

for k = 1:length(t) % iFFT to return to spatial domain
    u(k,:) = ifft(uhat(k,:));
end

% Plot solution in time
figure, waterfall((u(1:10:end,:)));
figure, imagesc(flipud(u));
```

Code 2.5: [Python] Code to simulate the 1D heat equation using the Fourier transform.

```
a = 1    # Thermal diffusivity constant
L = 100  # Length of domain
N = 1000 # Number of discretization points
dx = L/N
x = np.arange(-L/2,L/2,dx) # Define x domain

# Define discrete wavenumbers
kappa = 2*np.pi*np.fft.fftfreq(N, d=dx)

# Initial condition
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

```
u0 = np.zeros_like(x)
u0[int((L/2 - L/10)/dx):int((L/2 + L/10)/dx)] = 1
u0hat = np.fft.fft(u0)

# Simulate in Fourier frequency domain
dt = 0.1
t = np.arange(0,10,dt)
uhat_ri = odeint(rhsHeat, u0hat_ri, t, args=(kappa,a))
uhat = uhat_ri[:,:N] + (1j) * uhat_ri[:,N:]
u = np.zeros_like(uhat)
for k in range(len(t)):
    u[k,:] = np.fft.ifft(uhat[k,:])
u = u.real
```

Code 2.6: [MATLAB] Right-hand side for 1D heat equation in Fourier domain, $\mathrm{d}\hat{u}/\mathrm{d}t$.

```
function duhatdt = rhsHeat(t,uhat,kappa,a)
duhatdt = -a^2*(kappa.^2)'.*uhat;  % Linear and diagonal
```

Code 2.6: [Python] Right-hand side for 1D heat equation in Fourier domain, $\mathrm{d}\hat{u}/\mathrm{d}t$.

```
def rhsHeat(uhat_ri,t,kappa,a):
    uhat = uhat_ri[:N] + (1j) * uhat_ri[N:]
    d_uhat = -a**2 * (np.power(kappa,2)) * uhat
    d_uhat_ri = np.concatenate((d_uhat.real,d_uhat.imag)).
       astype('float64')
    return d_uhat_ri
```

One-Way Wave Equation

As second example is the simple linear PDE for the one-way equation:

$$u_t + cu_x = 0. \tag{2.41}$$

Any initial condition $u(0, x)$ will simply propagate to the right in time with speed c, as $u(t, x) = u(0, x - ct)$ is a solution. The code to simulate this PDE is nearly identical to the above code for the heat equation, and it is available on the book's GitHub. In this example, we simulate this PDE for an initial condition given by a Gaussian pulse. It is possible to integrate this equation in the Fourier transform domain, as before, using the vector field given by Code 2.7. However, it is also possible to integrate this equation in the spatial domain, simply using the FFT to compute derivatives and then transform back. The solution $u(t, x)$ is plotted in Figs. 2.15 and 2.16, as before.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

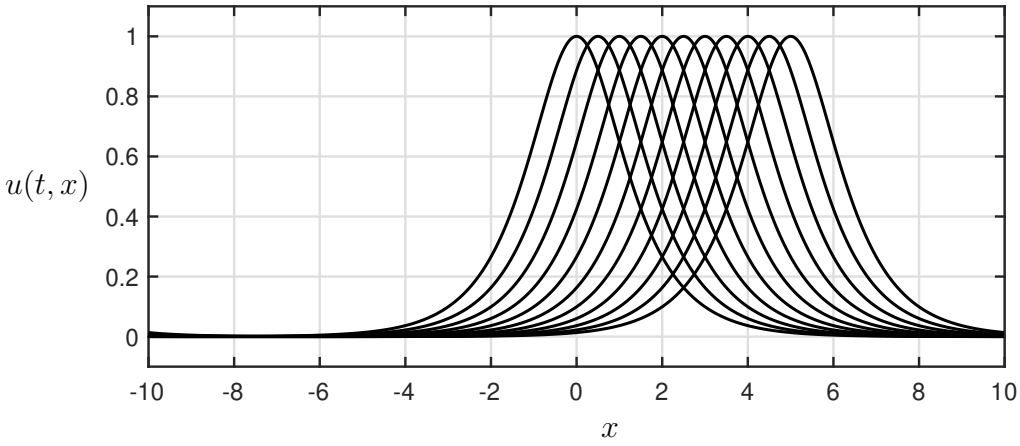


Figure 2.15: Solution of the 1D wave equation in time. As time evolves, the Gaussian initial condition moves from left to right at a constant wave speed.

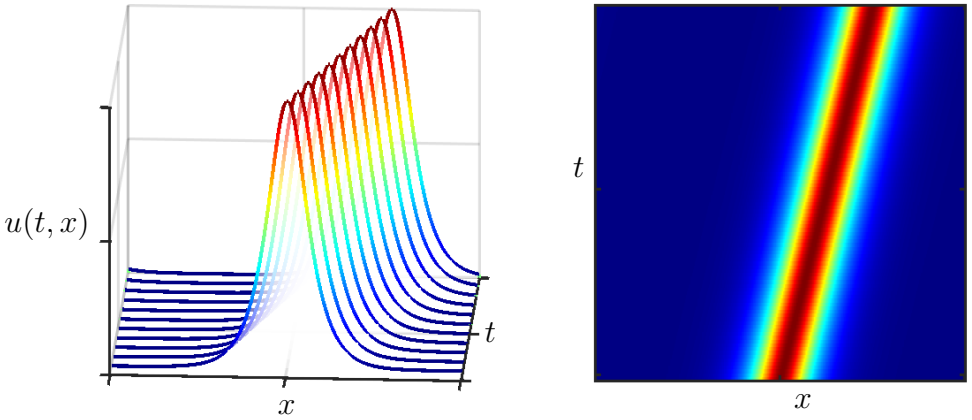


Figure 2.16: Evolution of the 1D wave equation in time, illustrated by a waterfall plot (left) and an x–t diagram (right).

Code 2.7: [MATLAB] Right-hand side for 1D wave equation in Fourier domain.

```
function duhatdt = rhsWave(t,uhat,kappa,c)
duhatdt = -c*i*kappa.*uhat;
```

Code 2.7: [Python]Right-hand side for 1D wave equation in Fourier domain.

```
def rhsWave(uhat_ri,t,kappa,c):
    uhat = uhat_ri[:N] + (1j) * uhat_ri[N:]
    d_uhat = -c*(1j)*kappa*uhat
    d_uhat_ri = np.concatenate((d_uhat.real,d_uhat.imag)).
       astype('float64')
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

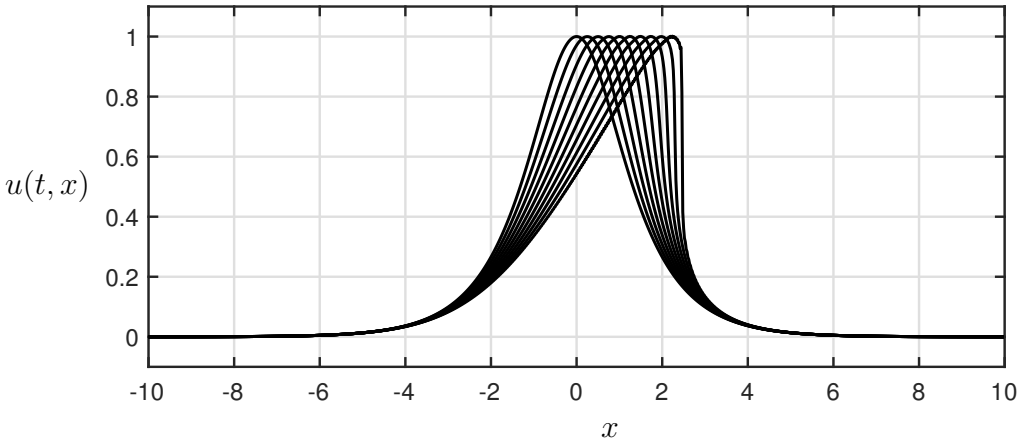


Figure 2.17: Solution of Burgers' equation in time. As time evolves, the leading edge of the Gaussian initial condition steepens, forming a shock front.

```
    return d_uhat_ri
```

Burgers' Equation

For the final example, we consider the nonlinear Burgers' equation,

$$u_t + uu_x = \nu u_{xx}, \tag{2.42}$$

which is a simple 1D example for the nonlinear convection and diffusion that gives rise to shock waves in fluids [336]. The nonlinear convection uu_x essentially gives rise to the behavior of wave steepening, where portions of u with larger amplitude will convect more rapidly, causing a shock front to form.

The code to simulate Burgers' equation is on the book's GitHub, giving rise to Figs. 2.17 and 2.18. Burgers' equation is an interesting example to solve with the FFT, because the nonlinearity requires us to map into and out of the Fourier domain at each time-step, as shown in the vector field in Code 2.8. In this example, we map into the Fourier transform domain to compute u_x and u_{xx}, and then map back to the spatial domain to compute the product uu_x. Figures 2.17 and 2.18 clearly show the wave steepening effect that gives rise to a shock. Without the damping term u_{xx}, this shock would become infinitely steep, but with damping, it maintains a finite width.

Code 2.8: [MATLAB] Right-hand side for Burgers' equation in Fourier transform domain.

```
function dudt = rhsBurgers(t,u,kappa,nu)
uhat = fft(u);
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

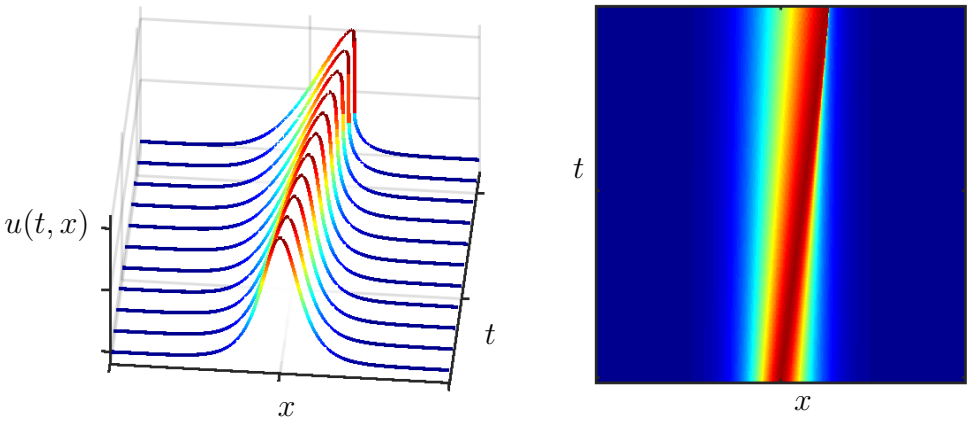


Figure 2.18: Evolution of Burgers' equation in time, illustrated by a waterfall plot (left) and an x–t diagram (right).

```
duhat = i*kappa.*uhat;
dduhat = -(kappa.^2).*uhat;
du = ifft(duhat);
ddu = ifft(dduhat);
dudt = -u.*du + nu*ddu;
```

Code 2.8: [Python] Right-hand side for Burgers' equation in Fourier transform domain.

```
def rhsBurgers(u,t,kappa,nu):
    uhat = np.fft.fft(u)
    d_uhat = (1j)*kappa*uhat
    dd_uhat = -np.power(kappa,2)*uhat
    d_u = np.fft.ifft(d_uhat)
    dd_u = np.fft.ifft(dd_uhat)
    du_dt = -u * d_u + nu*dd_u
    return du_dt.real
```

2.4 Gabor Transform and the Spectrogram

Although the Fourier transform provides detailed information about the frequency content of a given signal, it does not give any information about when in time those frequencies occur. The Fourier transform is only able to characterize truly periodic and stationary signals, as time is stripped out via the integration in (2.18). For a signal with non-stationary frequency content, such as

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

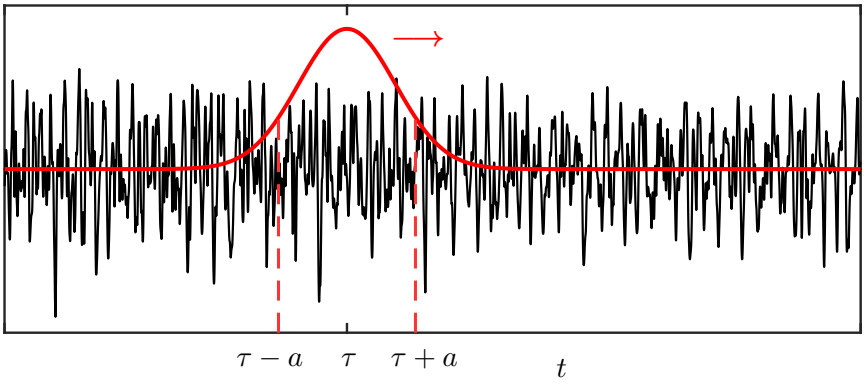


Figure 2.19: Illustration of the Gabor transform with a translating Gaussian window for the short-time Fourier transform.

a musical composition, it is important to simultaneously characterize the frequency content and its evolution in time.

The Gabor transform, also known as the short-time Fourier transform (STFT), computes a windowed FFT in a moving window [346, 572, 648], as shown in Fig. 2.19. This STFT enables the localization of frequency content in time, resulting in the *spectrogram*, which is a plot of frequency versus time, as demonstrated later in Figs. 2.21 and 2.22. The STFT is given by

$$\mathcal{G}(f)(t,\omega) = \hat{f}_g(t,\omega) = \int_{-\infty}^{\infty} f(\tau) e^{-i\omega\tau} \bar{g}(\tau - t)\, \mathrm{d}\tau = \langle f, g_{t,\omega} \rangle, \tag{2.43}$$

where $g_{t,\omega}(\tau)$ is defined as

$$g_{t,\omega}(\tau) = e^{i\omega\tau} g(\tau - t). \tag{2.44}$$

The function $g(t)$ is the kernel, and is often chosen to be a Gaussian:

$$g(t) = e^{-(t-\tau)^2/a^2}. \tag{2.45}$$

The parameter a determines the spread of the short-time window for the Fourier transform, and τ determines the center of the moving window.

The inverse STFT is given by

$$f(t) = \mathcal{G}^{-1}(\hat{f}_g(t,\omega)) = \frac{1}{2\pi \|g\|^2} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \hat{f}_g(\tau,\omega) g(t-\tau) e^{i\omega t}\, \mathrm{d}\omega\, \mathrm{d}t. \tag{2.46}$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

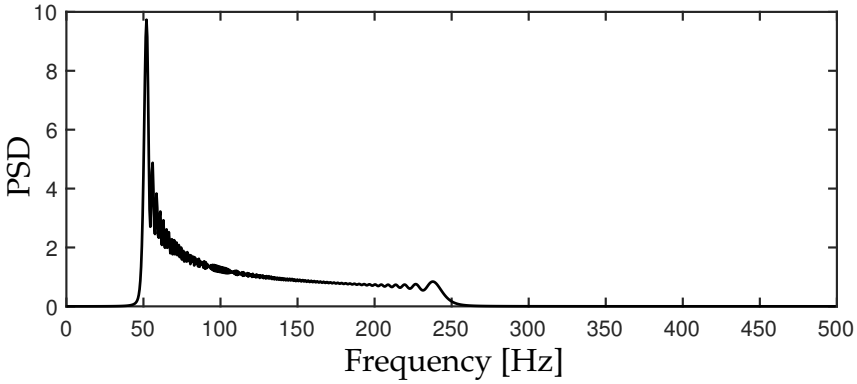


Figure 2.20: Power spectral density of quadratic chirp signal.

Discrete Gabor Transform

Generally, the Gabor transform will be performed on discrete signals, as with the FFT. In this case, it is necessary to discretize both time and frequency:

$$\nu = j\Delta\omega, \tag{2.47}$$

$$\tau = k\Delta t. \tag{2.48}$$

The discretized kernel function becomes

$$g_{j,k} = e^{i2\pi j\Delta\omega t} g(t - k\Delta t) \tag{2.49}$$

and the discrete Gabor transform is

$$\hat{f}_{j,k} = \langle f, g_{j,k} \rangle = \int_{-\infty}^{\infty} f(\tau)\bar{g}_{j,k}(\tau)\,\mathrm{d}\tau. \tag{2.50}$$

This integral can then be approximated using a finite Riemann sum on discretized functions f and $\bar{g}_{j,k}$.

Example: Quadratic Chirp

As a simple example, we construct an oscillating cosine function where the frequency of oscillation increases as a quadratic function of time:

$$f(t) = \cos(2\pi t\omega(t)) \quad \text{where} \quad \omega(t) = \omega_0 + (\omega_1 - \omega_0)t^2/3t_1^2. \tag{2.51}$$

The frequency shifts from ω_0 at $t = 0$ to ω_1 at $t = t_1$.

Figure 2.20 shows the power spectral density (PSD) obtained from the FFT of the quadratic chirp signal. Although there is a clear peak at $50\,\mathrm{Hz}$, there is no information about the progression of the frequency in time. The code to

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

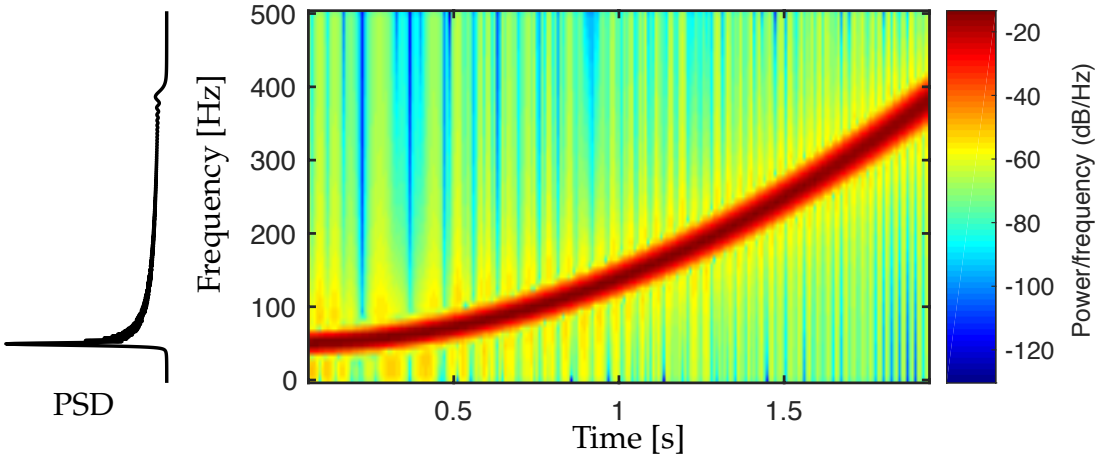


Figure 2.21: Spectrogram of quadratic chirp signal. The PSD is shown on the left, corresponding to the integrated power across rows of the spectrogram.

generate the spectrogram is given in Code 2.9, and the resulting spectrogram is plotted in Fig. 2.21, where it can be seen that the frequency content shifts in time.

Code 2.9: [MATLAB] Spectrogram of quadratic chirp, shown in Fig. 2.21.

```
t = 0:0.001:2;
f0 = 50;
f1 = 250;
t1 = 2;
x = chirp(t,f0,t1,f1,'quadratic');
x = cos(2*pi*t.*(f0 + (f1-f0)*t.^2/(3*t1^2)));
% There is a typo in Matlab documentation...
% ... divide by 3 so derivative amplitude matches frequency
spectrogram(x,128,120,128,1e3,'yaxis')
```

Code 2.9: [Python] Spectrogram of quadratic chirp, shown in Fig. 2.21.

```
dt = 0.001
t = np.arange(0,2,dt)
f0 = 50
f1 = 250
t1 = 2
x = np.cos(2*np.pi*t*(f0 + (f1-f0)*np.power(t,2)/(3*t1**2)))
plt.specgram(x, NFFT=128, Fs=1/dt, noverlap=120,cmap='jet')
```

Example: Beethoven's Sonata Pathétique

It is possible to analyze richer signals with the spectrogram, such as Beethoven's Sonata Pathétique, shown in Fig. 2.22. The spectrogram is widely used to an-

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

alyze music, and has recently been leveraged in the Shazam algorithm, which searches for key point markers in the spectrogram of songs to enable rapid classification from short clips of recorded music [740].

Figure 2.22 shows the first two bars of Beethoven's Sonata Pathétique, along with the spectrogram. In the spectrogram, the various chords and harmonics can be seen clearly. A zoom-in of the frequency shows two octaves, and how cleanly the various notes are excited. Code 2.10 loads the data, computes the spectrogram, and plots the result.

Code 2.10: [MATLAB] Compute spectrogram of Beethoven's Sonata Pathétique (Fig. 2.22).

```
% Download mp3read from http://www.mathworks.com/
   matlabcentral/fileexchange/13852-mp3read-and-mp3write
[Y,FS,NBITS,OPTS] = mp3read('beethoven.mp3');

%% Spectrogram using 'spectrogram' comand
T = 40;             % 40 seconds
y=Y(1:T*FS);        % First 40 seconds
spectrogram(y,5000,400,24000,24000,'yaxis');

%% Spectrogram using short-time Fourier transform 'stft'
wlen = 5000;    % Window length
h=400;         % Overlap is wlen - h
[S,f,t_stft] = stft(y, wlen, h, FS/4, FS); % y axis 0-4000HZ

imagesc(log10(abs(S)));    % Plot spectrogram (log-scaled)
```

To invert the spectrogram and generate the original sound:

```
[x_istft, t_istft] = istft(S, h, FS/4, FS);
sound(x_istft,FS);
```

Artists, such as Aphex Twin, have used the inverse spectrogram of images to generate music. The frequency of a given piano key is also easily computed. For example, the 40th key frequency is given by

```
freq = @(n)(((2^(1/12))^(n-49))*440);
freq(40) % frequency of 40th key = C
```

Uncertainty Principles

In time–frequency analysis, there is a fundamental uncertainty principle that limits the ability to simultaneously attain high resolution in both the time and frequency domains. In the extreme limit, a time series is perfectly resolved in time, but provides no information about frequency content, and the Fourier transform perfectly resolves frequency content, but provides no information

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

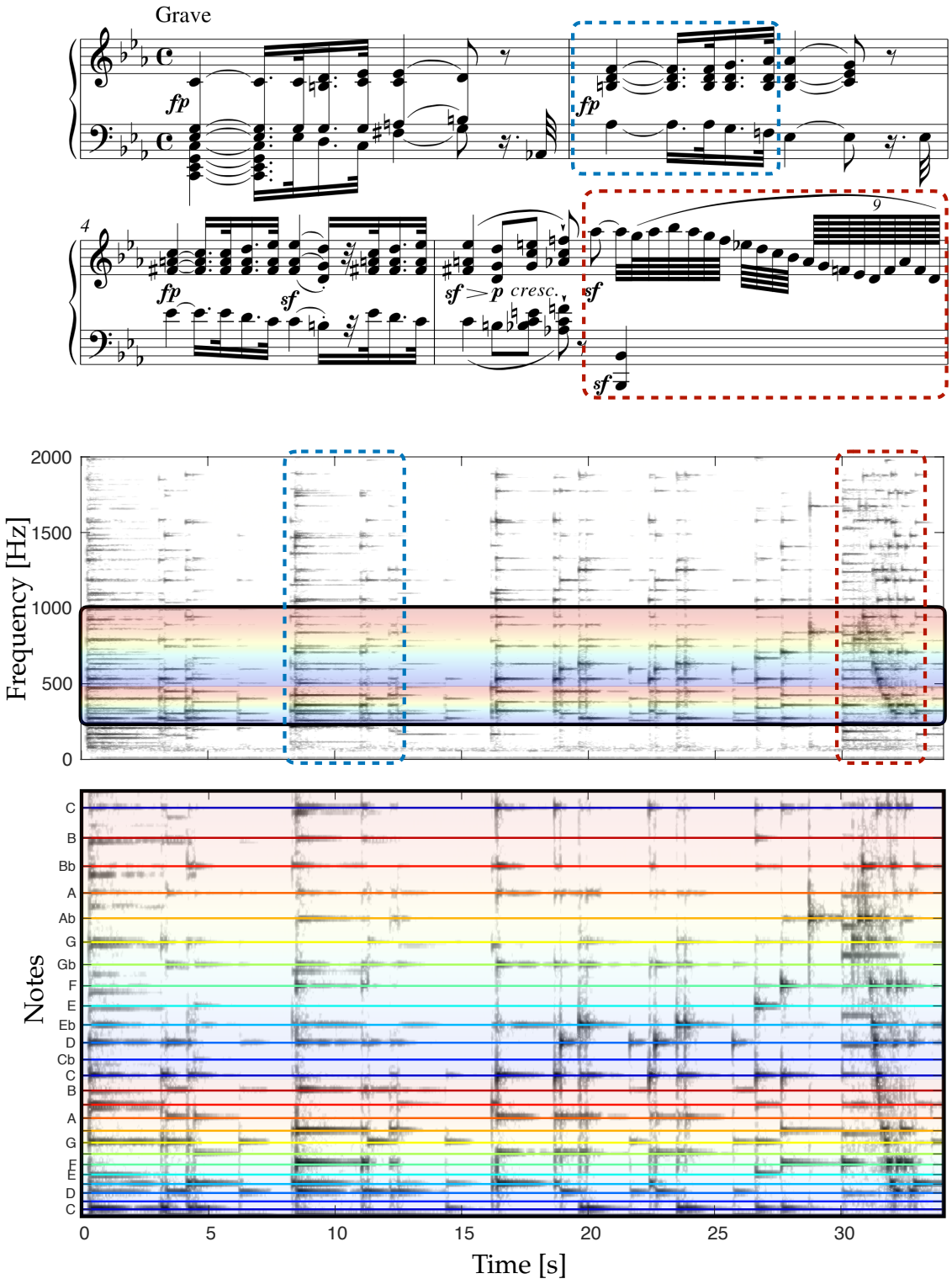


Figure 2.22: First two bars of Beethoven's Sonata Pathétique (No. 8 in C Minor, Op. 13), along with annotated spectrogram.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

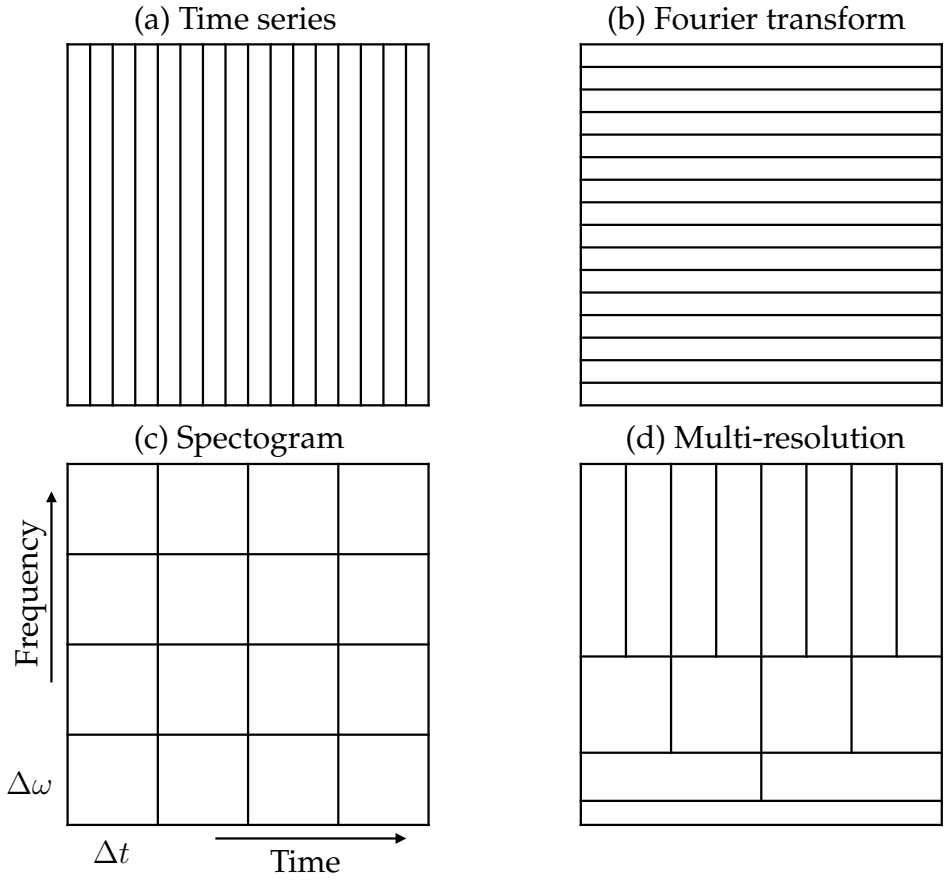


Figure 2.23: Illustration of resolution limitations and uncertainty in time–frequency analysis.

about when in time these frequencies occur. The spectrogram resolves both time and frequency information, but with lower resolution in each domain, as illustrated in Fig. 2.23. An alternative approach, based on a multi-resolution analysis, will be the subject of the next section.

Stated mathematically, the time–frequency uncertainty principle [564] may be written as

$$\left(\int_{-\infty}^{\infty} x^2 |f(x)|^2 \, \mathrm{d}x\right) \left(\int_{-\infty}^{\infty} \omega^2 |\hat{f}(\omega)|^2 \, \mathrm{d}\omega\right) \geq \frac{1}{16\pi^2}. \tag{2.52}$$

This is true if $f(x)$ is absolutely continuous and both $xf(x)$ and $f'(x)$ are square integrable. The function $x^2|f(x)|^2$ is the dispersion about $x = 0$. For real-valued functions, this is the second moment, which measures the variance if $f(x)$ is a Gaussian function. In other words, a function $f(x)$ and its Fourier transform cannot both be arbitrarily localized. If the function f approaches a delta function, then the Fourier transform must become broadband, and vice versa. This has implications for the Heisenberg uncertainty principle [320], as the position

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

and momentum wave functions are Fourier transform pairs.

In time–frequency analysis, the uncertainty principle has implications for the ability to localize the Fourier transform in time. These uncertainty principles are known as the Gabor limit. As the frequency content of a signal is resolved more finely, we lose information about when in time these events occur, and vice versa. Thus, there is a fundamental tradeoff between the simultaneously attainable resolutions in the time and frequency domains. Another implication is that a function f and its Fourier transform cannot both have finite support, meaning that they are localized, as stated in Benedick's theorem [14, 72].

2.5 Laplace Transform

The Laplace[1] transform is closely related to the Fourier transform, and it is used extensively in differential equations and control theory. Like the Fourier transform, the Laplace transform is used to transform PDEs into simpler ODEs, and it is also useful for transforming ODEs into algebraic equations. Here we will derive the Laplace transform as a generalized Fourier transform and demonstrate some of its useful properties.

The Fourier transform is defined for well-behaved functions that decay sufficiently rapidly to zero as the domain goes to infinity, i.e., for Lebesgue integrable functions $f \in L^1[(-\infty, \infty)]$. However, many functions we are interested in, such as exponential functions $e^{\lambda t}$, the well-named Heaviside function

$$H(t) = \begin{cases} 0 & \text{for } t \leq 0, \\ 1 & \text{for } t > 0, \end{cases} \tag{2.53}$$

and the trigonometric functions $\sin(t)$ and $\cos(t)$, do not satisfy this property; see Fig. 2.24 for examples. It is technically possible to Fourier-transform some of these functions, such as trigonometric functions, by multiplying by a window function and then taking the limit as the window becomes infinitely large. However, this approach does not translate to exponential functions, which are unbounded at either $t \to -\infty$ or $t \to \infty$. This limitation rules out using the Fourier transform to analyze a large class of ODEs and PDEs. The Laplace transform is a Fourier-like transform that is valid for the larger class of functions that are not Lebesgue integrable, including exponential functions.

We will consider the Laplace transform as a weighted, one-sided Fourier transform for badly behaved functions. The solution to transforming a function $f(t)$ that is unbounded as $t \to \infty$, such as $f(t) = e^{\lambda t}$, is to first multiply it by a

[1] Pierre-Simon Laplace was born the son of peasant farmers and is now immortalized on the Eiffel tower. He was also an early *data scientist*, realizing that real-world measurement data is noisy and imperfect, and must be viewed through the lens of probability theory.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

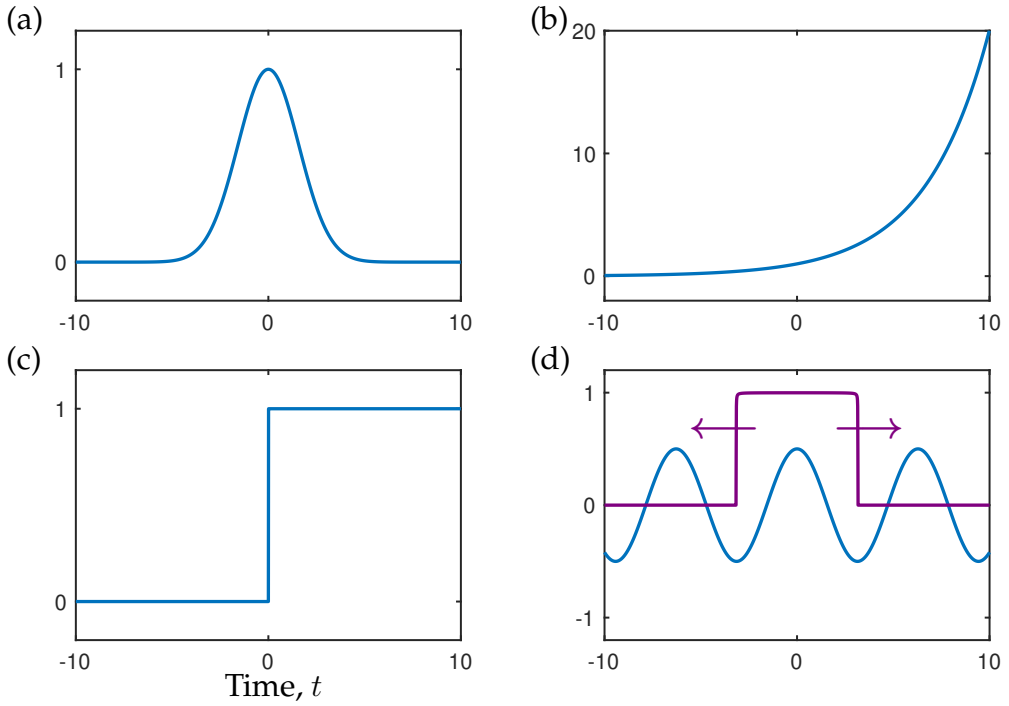


Figure 2.24: The Fourier transform requires that functions are well behaved at $\pm\infty$, as in the Gaussian function (a). Many functions are not well behaved and are difficult or impossible to Fourier-transform, such as the exponential function $e^{\lambda t}$ (b), the Heaviside function $H(t)$ (c), and the cosine function (d). It is possible to Fourier-transform the cosine function by multiplying by a window function and then extending the window size to infinity, but this does not work for unstable functions, like the exponential.

decaying exponential function $e^{-\gamma t}$, where γ is more damped than the growth of $f(t)$; this is the *weighting*. Although this solves the unboundedness of $f(t)$ as $t \to \infty$, now the function $e^{-\gamma t}$ is unbounded for $t \to -\infty$. Thus, we also multiply by the Heaviside function $H(t)$, which forces the function to be zero for $t < 0$; thus the transformation is *one-sided*. Our new weighted, one-sided function $F(t)$ is given by

$$F(t) = f(t)e^{-\gamma t}H(t) = \begin{cases} 0 & \text{for } t \leq 0, \\ f(t)e^{-\gamma t} & \text{for } t > 0. \end{cases} \tag{2.54}$$

Figure 2.25 shows this one-sided weighting procedure for a function that is unbounded as $t \to \infty$. We now take the Fourier transform $\hat{F}(\omega)$ of $F(t)$, which

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

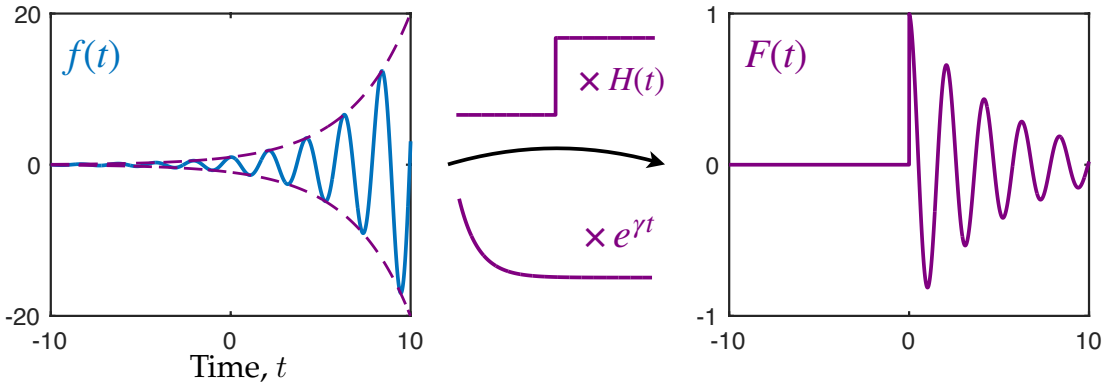


Figure 2.25: The Laplace transform is a weighted, one-sided Fourier transform for badly behaved functions. Given an unstable function $f(t)$, it is possible to multiply by the Heaviside function $H(t)$ and a sufficiently damped exponential function $e^{\gamma t}$, resulting in a function that may be Fourier-transformed.

will be the Laplace transform $\bar{f}(s)$ of $f(t)$:

$$\hat{F}(\omega) = \mathcal{F}(F(t)) = \int_{-\infty}^{\infty} F(t)e^{-i\omega t}\,\mathrm{d}t = \int_{0}^{\infty} f(t)e^{-\gamma t}e^{-i\omega t}\,\mathrm{d}t \tag{2.55a}$$

$$= \int_{0}^{\infty} f(t)e^{-(\gamma+i\omega)t}\,\mathrm{d}t = \int_{0}^{\infty} f(t)e^{-st}\,\mathrm{d}t = \bar{f}(s), \tag{2.55b}$$

where we have introduced the Laplace variable $s = \gamma + i\omega$.

To derive the inverse Laplace transform, we will begin with the inverse Fourier transform of $\hat{F}(\omega)$:

$$F(t) = \mathcal{F}^{-1}(\hat{F}(\omega)) = \frac{1}{2\pi}\int_{-\infty}^{\infty} \hat{F}(\omega)e^{i\omega t}\,\mathrm{d}\omega. \tag{2.56}$$

Multiplying both sides by $e^{\gamma t}$, we recover $f(t)H(t)$:

$$f(t)H(t) = e^{\gamma t}F(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} e^{\gamma t}\hat{F}(\omega)e^{i\omega t}\,\mathrm{d}\omega \tag{2.57a}$$

$$= \frac{1}{2\pi}\int_{-\infty}^{\infty} \hat{F}(\omega)e^{(\gamma+i\omega)t}\,\mathrm{d}\omega. \tag{2.57b}$$

We may express the right-hand side in terms of the Laplace variable $s = \gamma + i\omega$ by noting that $\mathrm{d}s = i\mathrm{d}\omega \implies \mathrm{d}\omega = (1/i)\mathrm{d}s$ and changing the bounds of integration from $-\infty$ to ∞ in $\mathrm{d}\omega$ to $\gamma - i\infty$ to $\gamma + i\infty$ in $\mathrm{d}s$:

$$f(t)H(t) = \frac{1}{2\pi i}\int_{\gamma-i\infty}^{\gamma+i\infty} \bar{f}(s)e^{st}\,\mathrm{d}s. \tag{2.58}$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

This is the expression for the inverse Laplace transform of $\bar{f}(s)$. The coefficient $1/i$ has been incorporated into the $1/(2\pi i)$ term in front of the integral, and $\hat{F}(\omega)$ has been replaced by $\bar{f}(s)$ from (2.55).

Therefore, the Laplace transform pair is given by

$$\bar{f}(s) = \mathcal{L}(f(t)) = \int_0^\infty f(t)e^{-st}\,\mathrm{d}t, \tag{2.59a}$$

$$f(t) = \mathcal{L}^{-1}(\bar{f}(s)) = \frac{1}{2\pi i}\int_{\gamma-i\infty}^{\gamma+i\infty} \bar{f}(s)e^{st}\,\mathrm{d}s. \tag{2.59b}$$

Note that in (2.59b) we have dropped the Heaviside function. This is equivalent to defining the Laplace transform as only being valid for functions $f(t)$ defined on the semi-infinite domain $t > 0$.

To summarize, the Laplace transform is a generalized Fourier transform designed to handle poorly behaved functions, such as exponentials. Even functions that have a Fourier transform often have a simpler Laplace transform. For example, the Dirac delta function, which requires an infinite number of Fourier frequencies to represent, is simply the constant 1 in the Laplace domain; this property makes the Laplace transform particularly useful for studying impulse responses and systems with forcing. A number of properties of the Fourier transform carry over to the Laplace transform (see Exercise 2-10), making them useful for solving ODEs and PDEs, especially in the context of control theory.

Derivatives of Functions

The Laplace transform of the derivative of a function is given by

$$\mathcal{L}\left(\frac{\mathrm{d}}{\mathrm{d}t}f(t)\right) = \int_0^\infty \overbrace{f'(t)}^{dv}\overbrace{e^{-st}}^{u}\,\mathrm{d}t \tag{2.60a}$$

$$= \Big[\underbrace{e^{-st}f(t)}_{uv}\Big]_0^\infty - \int_0^\infty \underbrace{f(t)}_{v}\Big[\underbrace{-se^{-st}}_{du}\Big]\,\mathrm{d}t \tag{2.60b}$$

$$= -f(0) + s\bar{f}(s). \tag{2.60c}$$

The $-f(0)$ term comes from $[e^{-st}f(t)]_0^\infty$ since e^{-st} is 0 at $t = \infty$ and 1 at $t = 0$. The formula for the Laplace transform of a higher derivative is given by

$$\mathcal{L}\left(\frac{\mathrm{d}^n}{\mathrm{d}t^n}f(t)\right) = -f^{(n-1)}(0) - sf^{(n-2)}(0) - \cdots - s^{n-1}f(0) + s^n\bar{f}(s), \tag{2.61}$$

where $f^{(k)}$ denotes the kth derivative. This property is extremely useful, allowing us to convert PDEs into ODEs and ODEs into algebraic expressions. For

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

example, consider the linear second-order damped harmonic oscillator,

$$\ddot{x} + a\dot{x} + bx = 0, \tag{2.62}$$

with initial condition $x(0) = x_0$ and $\dot{x}(0) = v_0$. It is possible to Laplace-transform this equation, noting that $\mathcal{L}(\dot{x}) = -x_0 + s\bar{x}(s)$ and that $\mathcal{L}(\ddot{x}) = -v_0 + s\mathcal{L}(\dot{x}) = -v_0 - sx_0 + s^2\bar{x}(s)$:

$$s^2\bar{x}(s) - sx_0 - v_0 + as\bar{x}(s) - ax_0 + b\bar{x}(s) = 0. \tag{2.63}$$

Rearranging the terms with $\bar{x}(s)$ on one side and the constants on the other, it is possible to obtain a rational function for $\bar{x}(s)$:

$$\underbrace{(s^2 + as + b)}_{\substack{\text{characteristic} \\ \text{polynomial}}}\bar{x}(s) = \underbrace{sx_0 + v_0 + ax_0}_{\substack{\text{initial} \\ \text{conditions}}} \quad \Longrightarrow \quad \bar{x}(s) = \frac{sx_0 + v_0 + ax_0}{s^2 + as + b}. \tag{2.64}$$

It is possible to solve for $x(t)$ by computing the inverse Laplace transform of $\bar{x}(s)$. For simplicity, let $a = 5,\ b = 4,\ x_0 = 2$, and $v_0 = -5$. Then

$$\bar{x}(s) = \frac{2s+5}{s^2+5s+4} = \frac{2s+5}{(s+1)(s+4)} = \frac{1}{s+1} + \frac{1}{s+4} \quad \Longrightarrow \quad x(t) = e^{-t} + e^{-4t}.$$

This uses the identity $\mathcal{L}(e^{\lambda t}) = 1/(s-\lambda)$, which is left as an exercise. The denominator of (2.64) is the characteristic polynomial for the ODE in (2.62), so the roots determine the eigenvalues λ_1 and λ_2. The initial conditions appear exclusively in the numerator, which determines the amplitude of the $e^{\lambda_1 t}$ and $e^{\lambda_2 t}$ solutions. Exercise 2-11 will determine the solution for general $a,\ b,\ x_0$, and v_0.

2.6 Wavelets and Multi-Resolution Analysis

Wavelets [192, 474] extend the concepts in Fourier analysis to more general orthogonal bases, and partially overcome the uncertainty principle discussed above by exploiting a multi-resolution decomposition, as shown in Fig. 2.23(d). This multi-resolution approach enables different time and frequency fidelities in different frequency bands, which is particularly useful for decomposing complex signals that arise from multi-scale processes such as are found in climatology, neuroscience, epidemiology, finance, and turbulence. Images and audio signals are also amenable to wavelet analysis, which is currently the leading method for image compression [23], as will be discussed in subsequent sections and chapters. Moreover, wavelet transforms may be computed using similar fast methods [83], making them scalable to high-dimensional data. There are a number of excellent books on wavelets [475, 523, 707], in addition to the primary references [192, 474].

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

The basic idea in wavelet analysis is to start with a function $\psi(t)$, known as the *mother* wavelet, and generate a family of scaled and translated versions of the function:

$$\psi_{a,b}(t) = \frac{1}{\sqrt{a}}\,\psi\left(\frac{t-b}{a}\right). \tag{2.65}$$

The parameters a and b are responsible for scaling and translating the function ψ, respectively. For example, one can imagine choosing a and b to scale and translate a function to fit in each of the segments in Fig. 2.23(d). If these functions are orthogonal, then the basis may be used for projection, as in the Fourier transform.

The simplest and earliest example of a wavelet is the *Haar* wavelet, developed in 1910 [307]:

$$\psi(t) = \begin{cases} 1 & \text{for } 0 \le t < 1/2, \\ -1 & \text{for } 1/2 \le t < 1, \\ 0 & \text{otherwise.} \end{cases} \tag{2.66}$$

The three Haar wavelets, $\psi_{1,0}$, $\psi_{1/2,0}$, and $\psi_{1/2,1/2}$, are shown in Fig. 2.26, representing the first two layers of the multi-resolution in Fig. 2.23(d). Notice that by choosing each higher frequency layer as a bisection of the next layer down, the resulting Haar wavelets are orthogonal, providing a hierarchical basis for a signal.

The orthogonality property of wavelets described above is critical for the development of the discrete wavelet transform (DWT) below. However, we begin with the continuous wavelet transform (CWT), which is given by

$$\mathcal{W}_\psi(f)(a,b) = \langle f, \psi_{a,b}\rangle = \int_{-\infty}^{\infty} f(t)\bar{\psi}_{a,b}(t)\,\mathrm{d}t, \tag{2.67}$$

where $\bar{\psi}_{a,b}$ denotes the complex conjugate of $\psi_{a,b}$. This is only valid for functions $\psi(t)$ that satisfy the boundedness property that

$$C_\psi = \int_{-\infty}^{\infty} \frac{|\hat{\psi}(\omega)|^2}{|\omega|}\,\mathrm{d}\omega < \infty. \tag{2.68}$$

The inverse continuous wavelet transform (iCWT) is given by

$$f(t) = \frac{1}{C_\psi}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \mathcal{W}_\psi(f)(a,b)\psi_{a,b}(t)\frac{1}{a^2}\,\mathrm{d}a\,\mathrm{d}b. \tag{2.69}$$

New wavelets may also be generated by the convolution $\psi * \phi$ if ψ is a wavelet and ϕ is a bounded and integrable function. There are many other popular mother wavelets ψ beyond the Haar wavelet, designed to have various properties. For example, the Mexican hat wavelet is given by

$$\psi(t) = (1-t^2)e^{-t^2/2}, \tag{2.70a}$$

$$\hat{\psi}(\omega) = \sqrt{2\pi}\,\omega^2 e^{-\omega^2/2}. \tag{2.70b}$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

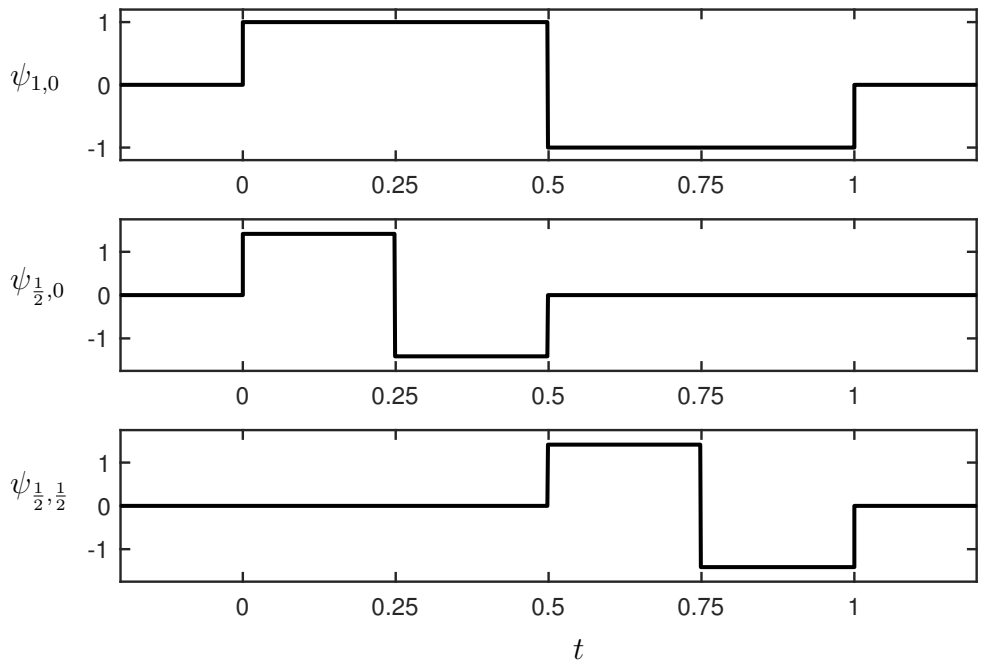


Figure 2.26: Three Haar wavelets for the first two levels of the multi-resolution in Fig. 2.23(d).

Discrete Wavelet Transform

As with the Fourier transform and Gabor transform, when computing the wavelet transform on data, it is necessary to introduce a discretized version. The discrete wavelet transform (DWT) is given by

$$\mathcal{W}_\psi(f)(j,k) = \langle f, \psi_{j,k} \rangle = \int_{-\infty}^{\infty} f(t)\bar{\psi}_{j,k}(t)\,\mathrm{d}t, \tag{2.71}$$

where $\psi_{j,k}(t)$ is a discrete family of wavelets

$$\psi_{j,k}(t) = \frac{1}{a^j}\,\psi\left(\frac{t-kb}{a^j}\right). \tag{2.72}$$

Again, if this family of wavelets is orthogonal, as in the case of the discrete Haar wavelets described above, it is possible to expand a function $f(t)$ uniquely in this basis:

$$f(t) = \sum_{j,k=-\infty}^{\infty} \langle f(t), \psi_{j,k}(t) \rangle \psi_{j,k}(t). \tag{2.73}$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

The explicit computation of a DWT is somewhat involved, and is the subject of several excellent papers and texts [192, 474, 475, 523, 707]. However, the goal here is not to provide computational details, but rather to give a high-level idea of what the wavelet transform accomplishes. By scaling and translating a given shape across a signal, it is possible to efficiently extract multi-scale structures in an efficient hierarchy that provides an optimal tradeoff between time and frequency resolution. This general procedure is widely used in audio and image processing, compression, scientific computing, and machine learning, to name a few examples.

2.7 Two-Dimensional Transforms and Image Processing

Although we analyzed both the Fourier transform and the wavelet transform on one-dimensional signals, both methods readily generalize to higher spatial dimensions, such as two-dimensional and three-dimensional signals. Both the Fourier and wavelet transforms have had tremendous impact on image processing and compression, which provides a compelling example to investigate higher-dimensional transforms.

Two-Dimensional Fourier Transform for Images

The two-dimensional (2D) Fourier transform of a matrix of data $\mathbf{X} \in \mathbb{R}^{n \times m}$ is achieved by first applying the one-dimensional (1D) Fourier transform to every row of the matrix, and then applying the 1D Fourier transform to every column of the intermediate matrix. This sequential row-wise and column-wise Fourier transform is shown in Fig. 2.27. Switching the order of taking the Fourier transform of rows and columns does not change the result.

It is simple to compute the 2D FFT in MATLAB

```
>> fhat = fft2(f);  % 2D FFT
>> f = ifft2(fhat); % 2D Inverse FFT
```

and in Python

```
>>> fhat = np.fft.fft2(f);  # 2D FFT
>>> f = np.fft.ifft2(fhat); # 2D Inverse FFT
```

A code to compute the 2D Fourier transform via 1D row-wise and column-wise FFTs is provided on the book's GitHub.

The two-dimensional FFT is effective for image compression, as many of the Fourier coefficients are small and may be neglected without loss in image quality. Thus, only a few large Fourier coefficients must be stored and transmitted. Code 2.11 and Fig. 2.28 demonstrate the FFT for image compression.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

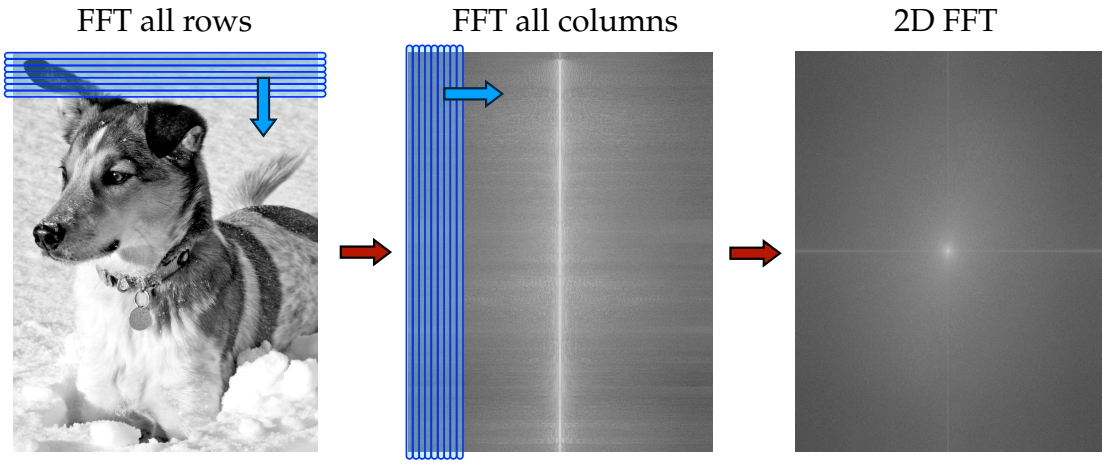


Figure 2.27: Schematic of 2D FFT. First, the FFT is taken of each row, and then the FFT is taken of each column of the resulting transformed matrix.

Code 2.11: [MATLAB] Image compression via the FFT.

```
Bt=fft2(B);     % B is grayscale image from above
Btsort = sort(abs(Bt(:)));  % Sort by magnitude

% Zero out all small coefficients and inverse transform
for keep=[.1 .05 .01 .002];
    thresh = Btsort(floor((1-keep)*length(Btsort)));
    ind = abs(Bt)>thresh;       % Find small indices
    Atlow = Bt.*ind;            % Threshold small indices
    Alow=uint8(ifft2(Atlow));   % Compressed image
    figure, imshow(Alow)        % Plot Reconstruction
end
```

Code 2.11: [Python] Image compression via the FFT.

```
Bt = np.fft.fft2(B)
Btsort = np.sort(np.abs(Bt.reshape(-1))) # sort by magnitude

# Zero out all small coefficients and inverse transform
for keep in (0.1, 0.05, 0.01, 0.002):
    thresh = Btsort[int(np.floor((1-keep)*len(Btsort)))]
    ind = np.abs(Bt)>thresh         # Find small indices
    Atlow = Bt * ind                # Threshold small indices
    Alow = np.fft.ifft2(Atlow).real # Compressed image
    plt.imshow(Alow,cmap='gray')
```

Finally, the FFT is extensively used for de-noising and filtering signals, as it is straightforward to isolate and manipulate particular frequency bands. Code 2.12

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

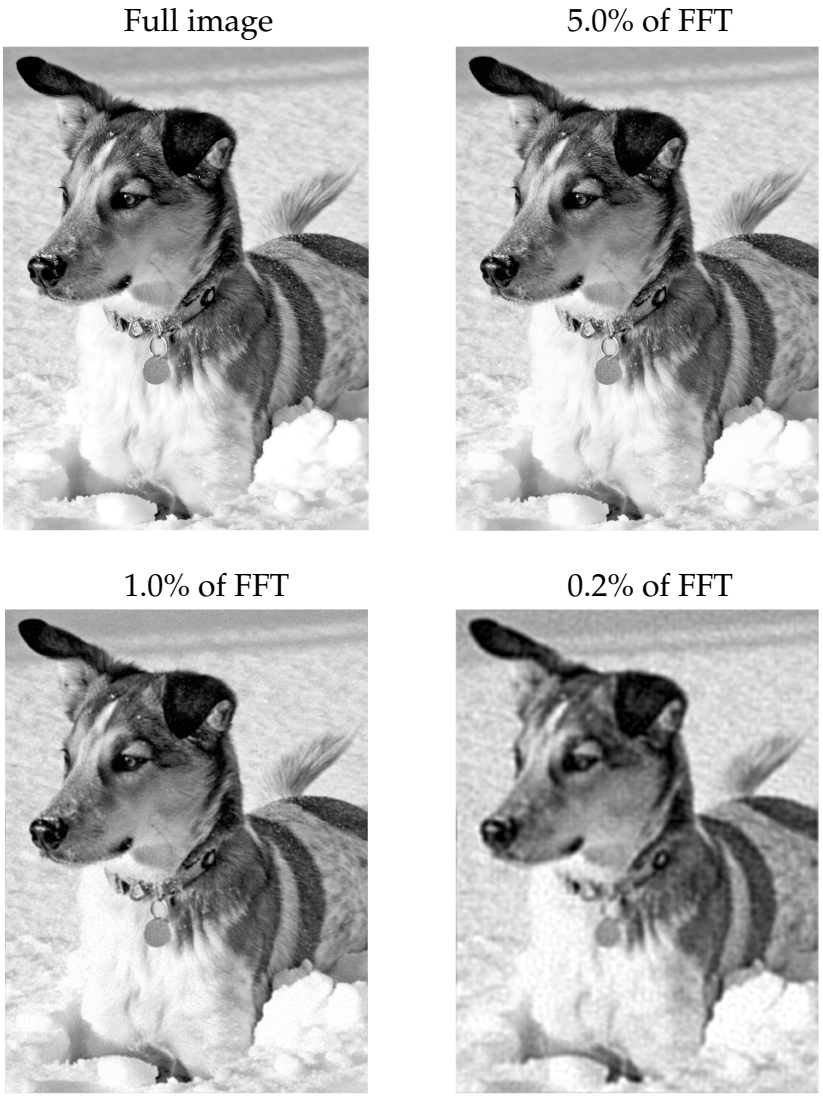


Figure 2.28: Compressed image using various thresholds to keep 5%, 1%, and 0.2% of the largest Fourier coefficients.

and Fig. 2.29 demonstrate the use of an FFT threshold filter to de-noise an image with Gaussian noise added. In this example, it is observed that the noise is especially pronounced in high-frequency modes, and we therefore zero-out any Fourier coefficient outside of a given radius containing low frequencies.

Code 2.12: [MATLAB] Image de-noising via the FFT.

```
Bnoise = B + uint8(200*randn(size(B)));  % Add some noise
Bt=fft2(Bnoise);
F = log(abs(Btshift)+1);          % Put FFT on log-scale

[nx,ny] = size(B);
[X,Y] = meshgrid(-ny/2+1:ny/2,-nx/2+1:nx/2);
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

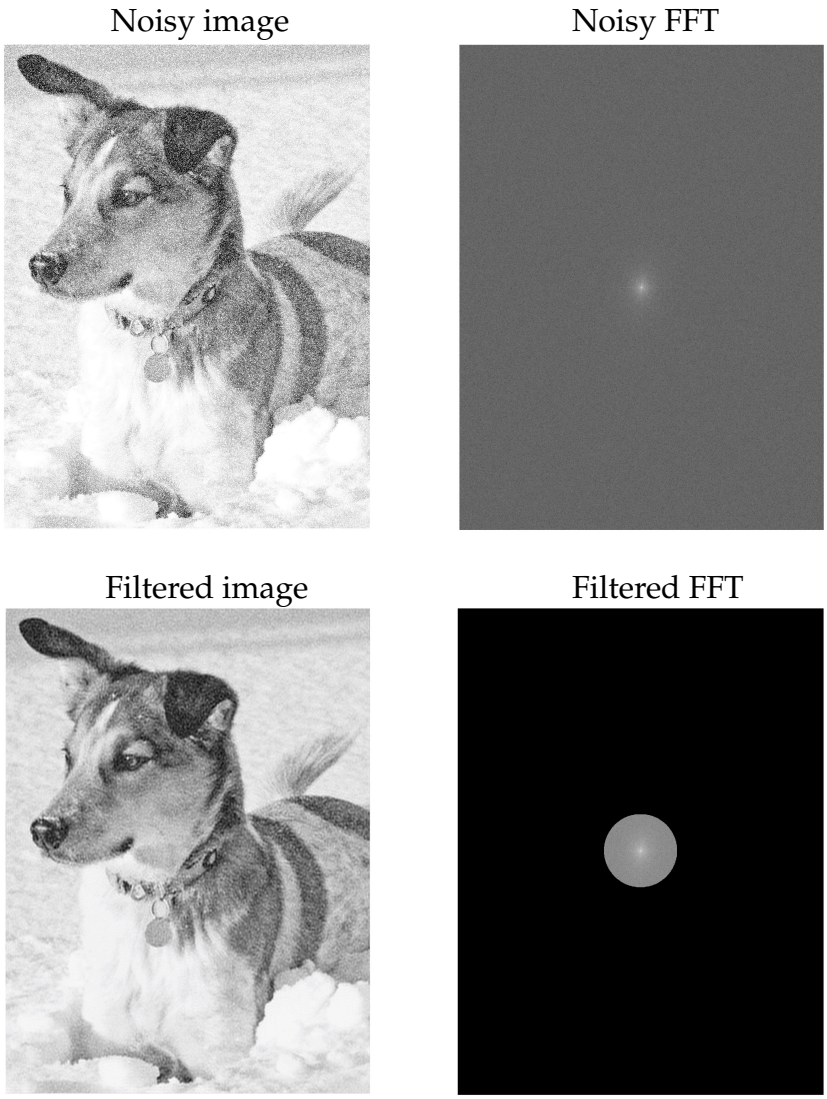


Figure 2.29: De-noising an image by eliminating high-frequency Fourier coefficients outside of a given radius (bottom right).

```
R2 = X.^2+Y.^2;
ind = R2<150^2;
Btshiftfilt = Btshift.*ind;
Ffilt = log(abs(Btshiftfilt)+1);   % Put FFT on log-scale

Btfilt = ifftshift(Btshiftfilt);
Bfilt = ifft2(Btfilt);
```

Code 2.12: [Python] Image de-noising via the FFT.

```
Bnoise = B + 200*np.random.randn(*B.shape).astype('uint8')
Bt = np.fft.fft2(Bnoise)
Btshift = np.fft.fftshift(Bt)
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

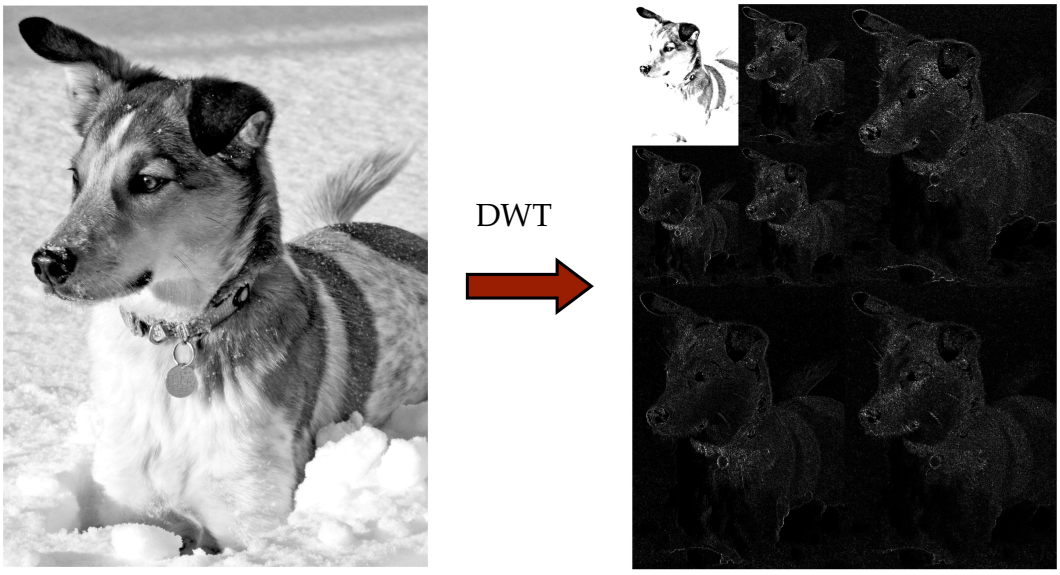


Figure 2.30: Illustration of three-level discrete wavelet transform.

```
F = np.log(np.abs(Btshift)+1) # Put FFT on log scale

nx,ny = B.shape
X,Y = np.meshgrid(np.arange(-ny/2+1,ny/2+1),np.arange(-nx
    /2+1,nx/2+1))
R2 = np.power(X,2) + np.power(Y,2)
ind = R2 < 150**2
Btshiftfilt = Btshift * ind
Ffilt = np.log(np.abs(Btshiftfilt)+1) # Put FFT on log scale
```

Two-Dimensional Wavelet Transform for Images

Similar to the FFT, the discrete wavelet transform is extensively used for image processing and compression. Code 2.13 computes the wavelet transform of an image, and the first three levels are illustrated in Fig. 2.30. In this figure, the hierarchical nature of the wavelet decomposition is seen. The upper left corner of the DWT image is a low-resolution version of the image, and the subsequent features add fine details to the image.

Code 2.13: [MATLAB] Example of a two-level wavelet decomposition.

```
%% Wavelet decomposition (2 level)
n = 2; w = 'db1'; [C,S] = wavedec2(B,n,w);

% LEVEL 1
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

```
A1 = appcoef2(C,S,w,1); % Approximation
[H1 V1 D1] = detcoef2('a',C,S,k); % Details
A1 = wcodemat(A1,128);
H1 = wcodemat(H1,128);
V1 = wcodemat(V1,128);
D1 = wcodemat(D1,128);

% LEVEL 2
A2 = appcoef2(C,S,w,1); % Approximation
[H2 V2 D2] = detcoef2('a',C,S,k); % Details
A2 = wcodemat(A2,128);
H2 = wcodemat(H2,128);
V2 = wcodemat(V2,128);
D2 = wcodemat(D2,128);

dec2 = [A2 H2; V2 D2];
dec1 = [imresize(dec2,size(H1)) H1 ; V1 D1];
image(dec1);
```

Code 2.13: [Python] Example of a two-level wavelet decomposition.

```
import pywt

## Wavelet decomposition (2 level)
n = 2
w = 'db1'
coeffs = pywt.wavedec2(B,wavelet=w,level=n)

# normalize each coefficient array
coeffs[0] /= np.abs(coeffs[0]).max()
for detail_level in range(n):
    coeffs[detail_level + 1] = [d/np.abs(d).max() for d in
       coeffs[detail_level + 1]]

arr, coeff_slices = pywt.coeffs_to_array(coeffs)
plt.imshow(arr,cmap='gray',vmin=-0.25,vmax=0.75)
```

Figure 2.31 shows several versions of the compressed image for various compression ratios, as computed by Code 2.14. The hierarchical representation of data in the wavelet transform is ideal for image compression. Even with an aggressive truncation, retaining only 0.5% of the DWT coefficients, the coarse features of the image are retained. Thus, when transmitting data, even if bandwidth is limited and much of the DWT information is truncated, the most important features of the data are transferred.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.



Figure 2.31: Compressed image using various thresholds to keep 5%, 1%, and 0.5% of the largest wavelet coefficients.

Code 2.14: [MATLAB] Wavelet decomposition for image compression.

```
[C,S] = wavedec2(B,4,'db1');
Csort = sort(abs(C(:))); % Sort by magnitude

for keep =  [.1 .05 .01 .005]
    thresh = Csort(floor((1-keep)*length(Csort)));
    ind = abs(C)>thresh;
    Cfilt = C.*ind;       % Threshold small indices

    % Plot Reconstruction
    Arecon=uint8(waverec2(Cfilt,S,'db1'));
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

```
    figure, imagesc(uint8(Arecon))
end
```

Code 2.14: [Python] Wavelet decomposition for image compression.

```
n = 4
w = 'db1'
coeffs = pywt.wavedec2(B,wavelet=w,level=n)

coeff_arr, coeff_slices = pywt.coeffs_to_array(coeffs)

Csort = np.sort(np.abs(coeff_arr.reshape(-1)))

for keep in (0.1, 0.05, 0.01, 0.005):
    thresh = Csort[int(np.floor((1-keep)*len(Csort)))]
    ind = np.abs(coeff_arr) > thresh
    Cfilt = coeff_arr * ind # Threshold small indices

    coeffs_filt = pywt.array_to_coeffs(Cfilt,coeff_slices,
       output_format='wavedec2')

    # Plot reconstruction
    Arecon = pywt.waverec2(coeffs_filt,wavelet=w)
    plt.imshow(Arecon.astype('uint8'),cmap='gray')
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Suggested Reading

Texts

(1) The analytical theory of heat, by J.-B. J. Fourier, 1978 [249].

(2) A wavelet tour of signal processing, by S. Mallat, 1999 [475].

(3) Spectral methods in MATLAB, by L. N. Trefethen, 2000 [710].

Papers and reviews

(1) An algorithm for the machine calculation of complex Fourier series, by J. W. Cooley and J. W. Tukey, *Mathematics of Computation*, 1965 [182].

(2) The wavelet transform, time–frequency localization and signal analysis, by I. Daubechies, *IEEE Transactions on Information Theory*, 1990 [192].

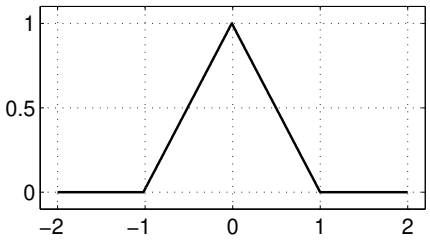
(3) An industrial strength audio search algorithm, by A. Wang et al., *Ismir*, 2003 [740].

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Homework

Exercise 2-1. Load the image dog.jpg and convert to grayscale. Use the FFT to compress the image at different compression ratios. Plot the error between the compressed and actual image as a function of the compression ratio.

Exercise 2-2. Consider the following triangular wave:



$$f(x) = \begin{cases} 0 & \text{for } x < -1, \\ 1 - |x| & \text{for } |x| \leq 1, \\ 0 & \text{for } 1 < x. \end{cases}$$

Compute the *Fourier series* by hand for the domain $-2 \leq x < 2$. Plot the mode coefficients a_n and b_n for the first 100 cosine and sine modes (i.e., for the first $n = 1$ to $n = 100$). Also, plot the approximation using $n = 10$ modes on top of the true triangle wave.

In a few sentences, explain the difference between the Fourier transform and the Fourier series.

Exercise 2-3. Use the FFT to solve the Korteweg–de Vries (KdV) equation,

$$u_t + u_{xxx} - uu_x = 0,$$

on a large domain with an initial condition $u(x, 0) = \mathrm{sech}(x)$. Plot the evolution.

Exercise 2-4. Use the FFT to solve the Kuramoto–Sivashinsky (KS) equation,

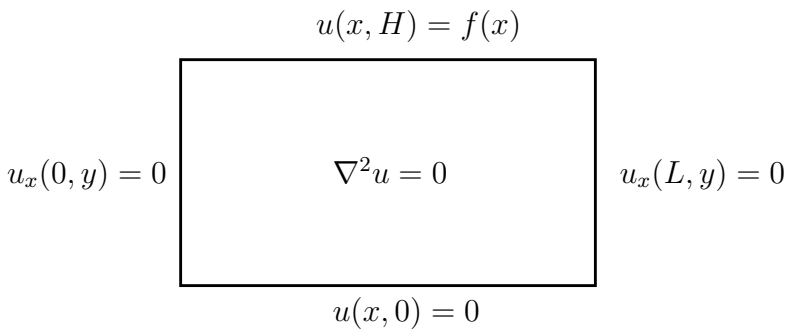
$$u_t + u_{xx} + u_{xxxx} + \tfrac{1}{2}u_x^2 = 0,$$

on a large domain with an initial condition $u(x, 0) = \mathrm{sech}(x)$. Plot the evolution.

Exercise 2-5. Solve for the analytic equilibrium temperature distribution using the 2D Laplace equation on an $L \times H$ sized rectangular domain with the following boundary conditions.

(a) Left: $u_x(0, y) = 0$ (insulating).
(b) Bottom: $u(x, 0) = 0$ (fixed temperature).
(c) Top: $u(x, H) = f(x)$ (zero temperature).
(d) Right: $u_x(L, y) = 0$ (insulating).

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.



Solve for a general boundary temperature $f(x)$. Also solve for a particular temperature distribution $f(x)$; you may choose any non-constant distribution you like.

How would this change if the left and right boundaries were fixed at zero temperature? (You do not have to solve this new problem, just explain in words what would change.)

Exercise 2-6. Now, compute the solution to the 2D heat equation on a circular disk through simulation. Recall that the heat equation is given by

$$u_t = \alpha^2 \nabla^2 u.$$

For this problem, we will solve the heat equation using a finite-difference scheme on a Cartesian grid. We will use a grid of 300×300 with the circular disk in the center. The radius of the circle is $r = 1$, $\alpha = 1$, and the domain is $[-1.5, 1.5]$ in x and $[-1.5, 1.5]$ in y. You can impose the boundary conditions by enforcing the temperature at points that are outside of the disk at the beginning of each new time-step. It should be easy to find points that are outside the disk, because they satisfy $x^2 + y^2 > 1$.

Simulate the unsteady heat equation for the following boundary conditions:

(a) The left half of the boundary of the disk is fixed at a temperature of $u = 1$ and the right half of the boundary is fixed at $u = 2$. Try simulating this with zero initial conditions first. Next, try initial conditions inside the disk where the top half is $u = -1$ and the bottom half is $u = 1$.

(b) The temperature at the boundary of the disk is fixed at $u(\theta) = cos(\theta)$.

Include your code and show some plots of your solutions to the heat equation. Plot the temperature distribution for each case (1) early on in the diffusion process, (2) near steady state, and (3) somewhere in the middle.

Exercise 2-7. Consider the PDE for a vibrating string of finite length L,

$$u_{tt} = c^2 u_{xx}, \quad 0 \leq x \leq L,$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

with the initial conditions

$$u(x,0) = 0, \quad u_t(x,0) = 0,$$

and boundary conditions

$$u(0,t) = 0, \quad u_x(L,t) = f(t).$$

Solve this PDE by using the Laplace transform. You may keep your solution in the frequency domain, since the inverse transform is complicated. Please simplify as much as possible using functions like $\sinh$ and $\cosh$.

This PDE cannot be solved by separation of variables. Why not? (That is, try to solve with separation of variables until you hit a contradiction.)

Exercise 2-8 [MATLAB]. Now, we will use the FFT to simultaneously compress and re-master an audio file. Please download the file `r2112.mat` and load the audio data into the matrix `rush` and the sample rate `FS`.

(a) Listen to the audio signal (`>>sound(rush,FS);`). Compute the FFT of this audio signal.

(b) Compute the power spectral density vector. Plot this to see what the output looks like. Also plot the spectrogram.

(c) Now, download `r2112noisy.mat` and load this file to initialize the variable `rushnoisy`. This signal is corrupted with high-frequency artifacts. Manually zero the last three-quarters of the Fourier components of this noisy signal (if `n=length(rushnoisy)`, then zero-out all Fourier coefficients from `n/4:n`). Use this filtered frequency spectrum to reconstruct the clean audio signal. When reconstructing, be sure to take the real part of the inverse Fourier transform: `cleansignal=real(ifft(filteredcoefs));`.

Because we are only keeping the first quarter of the frequency data, you must multiply the reconstructed signal by 2 so that it has the correct normalized power. Be sure to use the `sound` command to listen to the pre- and post-filtered versions. Plot the power spectral density and spectrograms of the pre- and post-filtered signals.

Exercise 2-9. The convolution integral and the impulse response may be used to simulate how an audio signal would sound under various conditions, such as in a long hallway, in a concert hall, or in a swimming pool.

The basic idea is that you can record the audio response to an impulsive sound in a given location, like a concert hall. For example, imagine that you put a

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

microphone in the most expensive seats in the hall and then record the sound from a shotgun blast up on the stage. (Do not try this!!) Then, if you have a "flat" studio recording of some other audio, you can simulate how it would have sounded in the concert hall by convolving the two signals.

Download and unzip sounds.zip to find various sounds and impulse-response filters. Convolve the various audio files (labeled sound1.wav, ...) with the various filters (labeled FilterXYZ.wav, ...). In MATLAB, use the `wavread` command to load and the `conv` command to convolve. It is best to add 10% of the filtered audio (also known as "wet" audio) to 90% of the original audio (also known as "dry" audio). Listen to the filtered audio, as well as the original audio and the impulse-response filters (note that each sound has a sampling rate of `FS=11,025`). However, you will need to be careful when adding the 10% filtered and 90% unfiltered signals, since the filtered audio will not necessarily have the same length as the filtered audio.

There is a great video explaining how to actually create these impulse responses: `http://www.audioease.com/Pages/Altiverb/sampling.php`

Exercise 2-10. Verify the following properties of the Laplace transform.

(a) Exponential: $\mathcal{L}(e^{\lambda t}) = \dfrac{1}{s-\lambda}$.

(b) Linearity: $\mathcal{L}(af(t) + bf(t)) = a\bar{f}(s) + b\bar{f}(s)$ for all constants $a, b \in \mathbb{C}$.

(c) Convolution: $\mathcal{L}(f(t) * g(t)) = \mathcal{L}(f(t))\mathcal{L}(g(t)) = \bar{f}(s)\bar{g}(s)$.

(d) Constant: $\mathcal{L}(1) = \dfrac{1}{s}$.

(e) Delta function: $\mathcal{L}(\delta(t)) = 1$.

Exercise 2-11. Use the Laplace transform to solve for the general solution to (2.62) for arbitrary $a,\ b,\ x_0$, and v_0.

Show how this solution changes when the system is forced with $u(t)$:

$$\ddot{x} + a\dot{x} + bx = u(t).$$

What if $u(t) = \delta(t)$? What if $u(t) = 1$ for $t > 0$?

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Chapter 3

Sparsity and Compressed Sensing

The inherent structure observed in natural data implies that the data admits a sparse representation in an appropriate coordinate system. In other words, if natural data is expressed in a well-chosen basis, only a few parameters are required to characterize the modes that are active, and in what proportion. All of data compression relies on sparsity, whereby a signal is represented more efficiently in terms of the sparse vector of coefficients in a generic transform basis, such as Fourier or wavelet bases. Recent fundamental advances in mathematics have turned this paradigm upside down. Instead of collecting a high-dimensional measurement and then compressing, it is now possible to acquire *compressed* measurements and solve for the sparsest high-dimensional signal that is consistent with the measurements. This so-called *compressed sensing* is a valuable new perspective that is also relevant for complex systems in engineering, with potential to revolutionize data acquisition and processing. In this chapter, we discuss the fundamental principles of sparsity and compression as well as the mathematical theory that enables compressed sensing, all worked out on motivating examples.

Our discussion on sparsity and compressed sensing will necessarily involve the critically important fields of optimization and statistics. Sparsity is a useful perspective to promote *parsimonious* models that avoid overfitting and remain interpretable because they have the minimal number of terms required to explain the data. This is related to Occam's razor, which states that the simplest explanation is generally the correct one. Sparse optimization is also useful for adding robustness with respect to outliers and missing data, which generally skew the results of least-squares regression, such as the SVD. The topics in this chapter are closely related to randomized linear algebra discussed in Section 1.8, and they will also be used in several subsequent chapters. Sparse regression will be explored further in Chapter 4 and will be used in Section 7.3 to identify interpretable and parsimonious nonlinear dynamical systems models from data.

3.1 Sparsity and Compression

Most natural signals, such as images and audio, are highly compressible. This compressibility means that, when the signal is written in an appropriate basis, only a few modes are active, thus reducing the number of values that must be stored for an accurate representation. Said another way, a compressible signal $\mathbf{x} \in \mathbb{R}^n$ may be written as a sparse vector $\mathbf{s} \in \mathbb{R}^n$ (containing mostly zeros) in a transform basis $\mathbf{\Psi} \in \mathbb{R}^{n \times n}$:

$$\mathbf{x} = \mathbf{\Psi}\mathbf{s}. \tag{3.1}$$

Specifically, the vector $\mathbf{s}$ is called K-sparse in $\mathbf{\Psi}$ if there are exactly K non-zero elements. If the basis $\mathbf{\Psi}$ is generic, such as the Fourier or wavelet basis, then only the few active terms in $\mathbf{s}$ are required to reconstruct the original signal $\mathbf{x}$, reducing the data required to store or transmit the signal.

Images and audio signals are both compressible in Fourier or wavelet bases, so that after taking the Fourier or wavelet transform, most coefficients are small and may be set exactly equal to zero with negligible loss of quality. These few active coefficients may be stored and transmitted, instead of the original high-dimensional signal. Then, to reconstruct the original signal in the ambient space (i.e., in pixel space for an image), one need only take the inverse transform. As discussed in Chapter 2, the fast Fourier transform is the enabling technology that makes it possible to efficiently reconstruct an image $\mathbf{x}$ from the sparse coefficients in $\mathbf{s}$. This is the foundation of JPEG compression for images and MP3 compression for audio.

The Fourier modes and wavelets are *generic* or *universal* bases, in the sense that nearly all natural images or audio signals are sparse in these bases. Therefore, once a signal is compressed, one needs only to store or transmit the sparse vector $\mathbf{s}$ rather than the entire matrix $\mathbf{\Psi}$, since the Fourier and wavelet transforms are already hard-coded on most machines. In Chapter 1 we found that it is also possible to compress signals using the SVD, resulting in a *tailored* basis. In fact, there are two ways that the SVD can be used to compress an image: (1) we may take the SVD of the image directly and only keep the dominant columns of $\mathbf{U}$ and $\mathbf{V}$ (Section 1.2): or (2) we may represent the image as a linear combination of *eigen*-images, as in the eigenfaces example (Section 1.6). The first option is relatively inefficient, as the basis vectors $\mathbf{U}$ and $\mathbf{V}$ must be stored. However, in the second case, a tailored basis $\mathbf{U}$ may be computed and stored once, and then used to compress an entire class of images, such as human faces. This tailored basis has the added advantage that the modes are interpretable as correlation features that may be useful for learning. It is important to note that both the Fourier basis $\mathcal{F}$ and the SVD basis $\mathbf{U}$ are unitary transformations, which will become important in the following sections.

Although the majority of compression theory has been driven by audio, image, and video applications, there are many implications for engineering

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

systems. The solution to a high-dimensional system of differential equations typically evolves on a low-dimensional manifold, indicating the existence of coherent structures that facilitate sparse representation. Even broadband phenomena, such as turbulence, may be instantaneously characterized by a sparse representation. This has a profound impact on how to sense and compute, as will be described throughout this chapter and the remainder of the book.

Example: Image Compression

Compression is relatively simple to implement on images, as described in Section 2.7 and revisited here (see Fig. 3.1 and Code 3.1).

Code 3.1: [MATLAB] Image compression based on the FFT.

```
A=imread('../CODE_DATA/CH03/jelly', 'jpeg'); % Load image
B=rgb2gray(A);              % Convert image to grayscale
imshow(B)                   % Plot image

%% Compute the FFT of image using fft2
Bhat=fft2(B);

%% Zero out all small coefficients and inverse transform
Bhatsort = sort(abs(Bhat(:)));
keep = 0.05;
thresh = Bhatsort(floor((1-keep)*length(Bhatsort)));
ind = abs(Bhat)>thresh;
Bhatcompressed = Bhat.*ind;
Bcompressed=uint8(ifft2(Bhatcompressed));
```

Code 3.1: [Python] Image compression based on the FFT.

```
A = imread(os.path.join('..','DATA','jelly.jpg'))
B = np.mean(A, -1); # Convert RGB to grayscale
plt.imshow(B,cmap='gray')

## Compute FFT of image using fft2
Bhat = np.fft.fft2(B)

## Zero out all small coefficients and inverse transform
Bhatsort = np.sort(np.abs(np.reshape(Bhat,-1)))
keep = 0.05
thresh = Bhatsort[int(np.floor((1-keep)*len(Bhatsort)))]
ind = np.abs(Bhat) > thresh
Bhatcompress = Bhat * ind
Bcompress = np.fft.ifft2(Bhatcompress).astype('uint8')
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

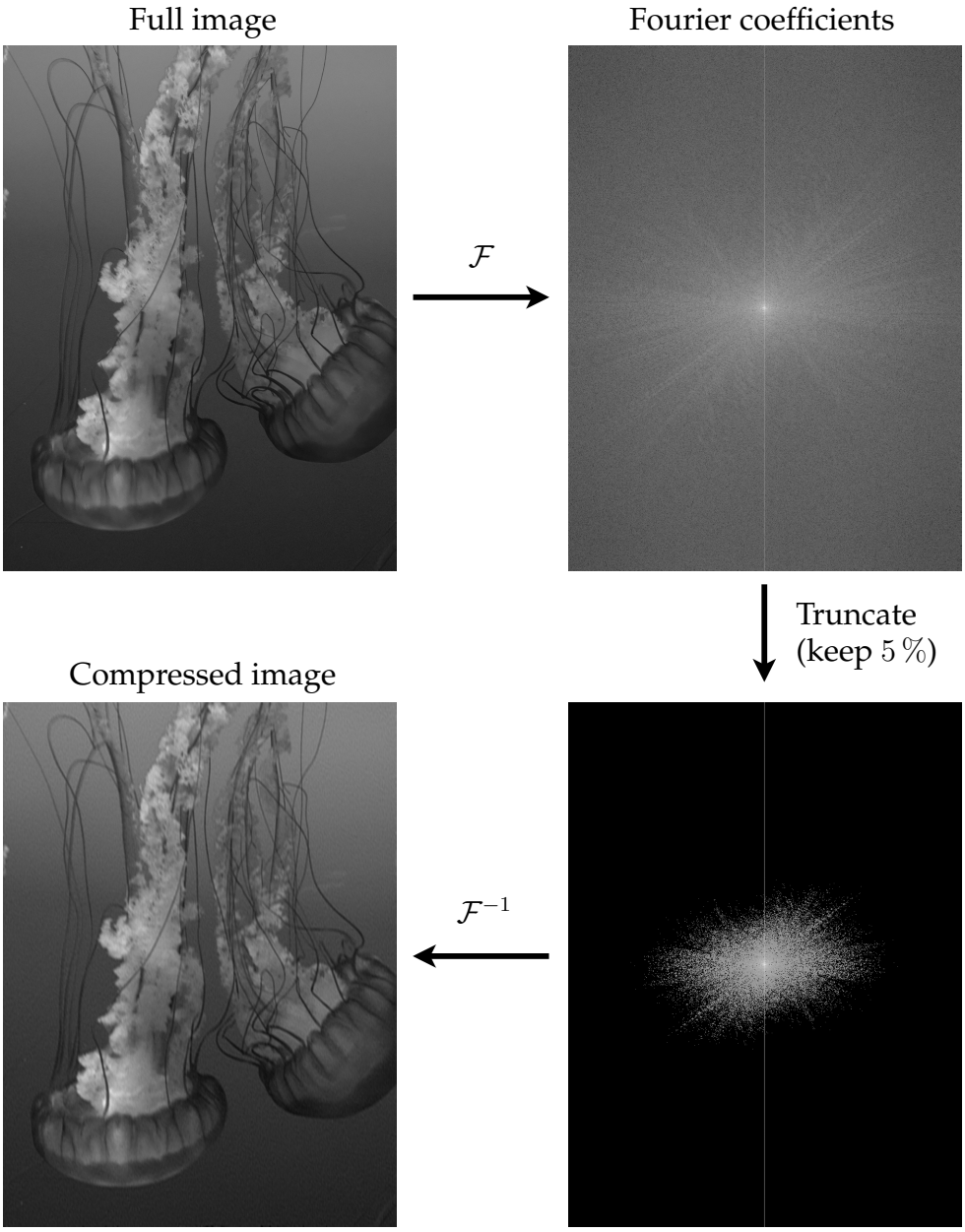


Figure 3.1: Illustration of compression with the fast Fourier transform (FFT) $\mathcal{F}$.

To understand the role of the sparse Fourier coefficients in a compressed image, it helps to view the image as a surface, where the height of a point is given by the brightness of the corresponding pixel. This is shown in Fig. 3.2. Here we see that the surface is relatively simple, and may be represented as a sum of a few spatial Fourier modes.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

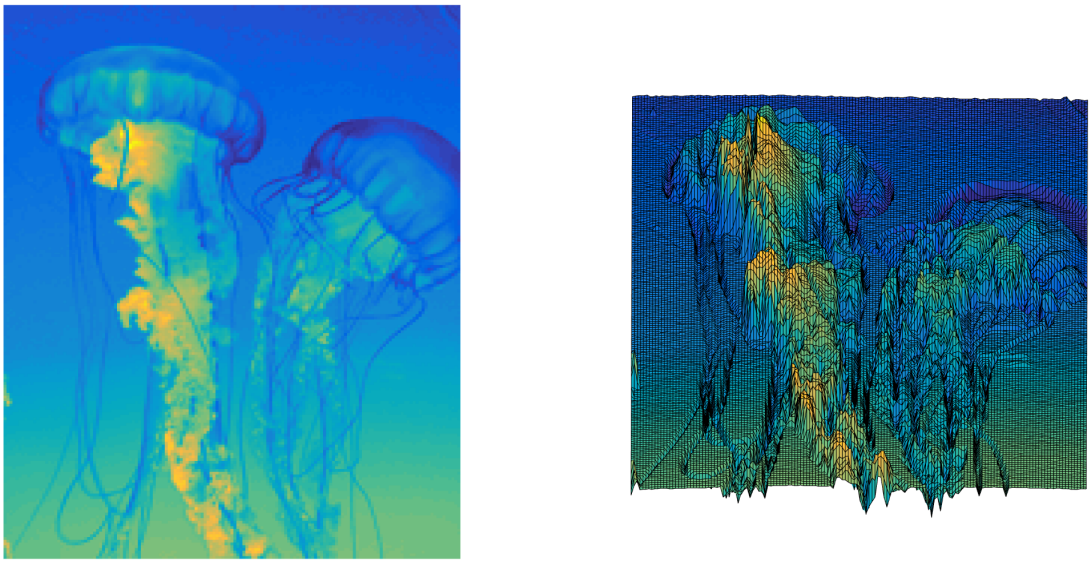


Figure 3.2: Compressed image (left), and viewed as a surface (right).

Why Signals Are Compressible: the Vastness of Image Space

It is important to note that the compressibility of images is related to the overwhelming dimensionality of image space. For even a simple 20×20 pixel black-and-white image, there are 2^{400} distinct possible images, which is larger than the number of nucleons in the known universe. The number of images is considerably more staggering for higher-resolution images with greater color depth.

In the space of one megapixel images (i.e., 1000×1000 pixels), there is an image of us each being born, of me typing this sentence, and of you reading it. However vast the space of these *natural* images, they occupy a tiny, minuscule fraction of the total image space. The majority of the images in image space represent random noise, resembling television static. For simplicity, consider grayscale images, and imagine drawing a random number for the gray value of each of the pixels. With exceedingly high probability, the resulting image will look like noise, with no apparent significance. You could draw these random images for an entire lifetime and never find an image of a mountain, or a person, or anything physically recognizable.[1]

[1]The vastness of signal space was described in Borges's "The Library of Babel" in 1944, where he describes a library containing all possible books that could be written, of which actual coherent books occupy a nearly immeasurably small fraction [97]. In Borges's library, there are millions of copies of this very book, with variations on this single sentence. Another famous variation on this theme considers that, given enough monkeys typing on enough typewriters, one would eventually recreate the works of Shakespeare. One of the oldest related descriptions of these combinatorially large spaces dates back to Aristotle.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

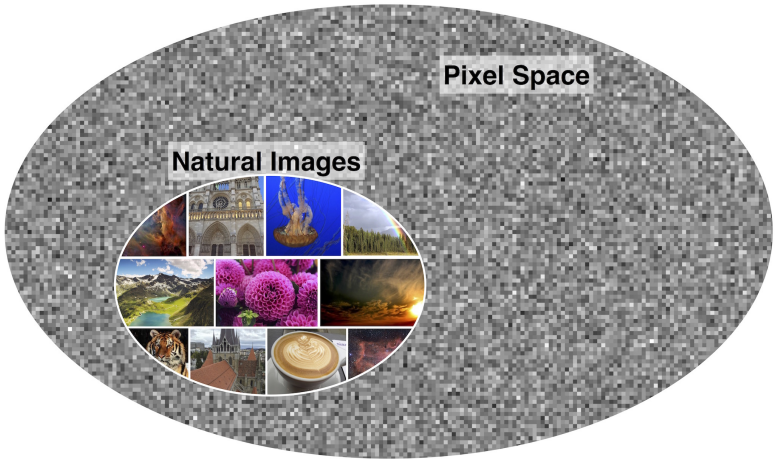


Figure 3.3: Illustration of the vastness of image (pixel) space, with natural images occupying a vanishingly small fraction of the space.

In other words, natural images are extremely rare in the vastness of image space, as illustrated in Fig. 3.3. Because so many images are unstructured or random, most of the dimensions used to encode images are only necessary for these random images. These dimensions are redundant if all we cared about was encoding natural images. An important implication is that the images we care about (i.e., natural images) are highly compressible, if we find a suitable transformed basis where the redundant dimensions are easily identified.

3.2 Compressed Sensing

Despite the considerable success of compression in real-world applications, it still relies on having access to full high-dimensional measurements. The recent advent of compressed sensing [53, 54, 151, 152, 153, 154, 156, 157, 204] turns the compression paradigm upside down: instead of collecting high-dimensional data just to compress and discard most of the information, it is instead possible to collect surprisingly few *compressed* or *random* measurements and then infer what the sparse representation is in the transformed basis. The idea behind compressed sensing is relatively simple to state mathematically, but, until recently, finding the sparsest vector consistent with measurements was a non-polynomial (NP) hard problem. The rapid adoption of compressed sensing throughout the engineering and applied sciences rests on the solid mathematical framework[2] that provides conditions for when it is possible to reconstruct the full signal with high probability using convex algorithms.

[2]Interestingly, the incredibly important collaboration between Emmanuel Candès and Terrance Tao began with them discussing the odd properties of signal reconstruction at their kids' daycare.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Mathematically, compressed sensing exploits the sparsity of a signal in a generic basis to achieve full signal reconstruction from surprisingly few measurements. If a signal $\mathbf{x}$ is K-sparse in $\mathbf{\Psi}$, then instead of measuring $\mathbf{x}$ directly (n measurements) and then compressing, it is possible to collect dramatically fewer randomly chosen or *compressed* measurements and then solve for the non-zero elements of $\mathbf{s}$ in the transformed coordinate system. The measurements $\mathbf{y} \in \mathbb{R}^p$, with $K < p \ll n$ are given by

$$\mathbf{y} = \mathbf{C}\mathbf{x}. \tag{3.2}$$

The measurement matrix[3] $\mathbf{C} \in \mathbb{R}^{p \times n}$ represents a set of p linear measurements on the state $\mathbf{x}$. The choice of measurement matrix $\mathbf{C}$ is of critical importance in compressed sensing, and is discussed in Section 3.4. Typically, measurements may consist of random projections of the state, in which case the entries of $\mathbf{C}$ are Gaussian or Bernoulli distributed random variables. It is also possible to measure individual entries of $\mathbf{x}$ (i.e., single pixels if $\mathbf{x}$ is an image), in which case $\mathbf{C}$ consists of random rows of the identity matrix.

With knowledge of the sparse vector $\mathbf{s}$, it is possible to reconstruct the signal $\mathbf{x}$ from (3.1). Thus, the goal of compressed sensing is to find the sparsest vector $\mathbf{s}$ that is consistent with the measurements $\mathbf{y}$:

$$\mathbf{y} = \mathbf{C}\mathbf{\Psi}\mathbf{s} = \mathbf{\Theta}\mathbf{s}. \tag{3.3}$$

The system of equations in (3.3) is under-determined since there are infinitely many consistent solutions $\mathbf{s}$. The *sparsest* solution $\hat{\mathbf{s}}$ satisfies the following optimization problem:

$$\hat{\mathbf{s}} = \underset{\mathbf{s}}{\operatorname{argmin}} \|\mathbf{s}\|_0 \quad \text{subject to} \quad \mathbf{y} = \mathbf{C}\mathbf{\Psi}\mathbf{s}, \tag{3.4}$$

where $\|\cdot\|_0$ denotes the ℓ_0-pseudo-norm, given by the number of non-zero entries; this is also referred to as the cardinality of $\mathbf{s}$.

The optimization in (3.4) is non-convex, and in general the solution can only be found with a brute-force search that is combinatorial in n and K. In particular, all possible K-sparse vectors in $\mathbb{R}^n$ must be checked; if the exact level of sparsity K is unknown, the search is even broader. Because this search is combinatorial, solving (3.4) is intractable for even moderately large n and K, and the prospect of solving larger problems does not improve with Moore's law of exponentially increasing computational power.

Fortunately, under certain conditions on the measurement matrix $\mathbf{C}$, it is possible to relax the optimization in (3.4) to a convex ℓ_1-minimization [157, 204]:

$$\hat{\mathbf{s}} = \underset{\mathbf{s}}{\operatorname{argmin}} \|\mathbf{s}\|_1 \quad \text{subject to} \quad \mathbf{y} = \mathbf{C}\mathbf{\Psi}\mathbf{s}, \tag{3.5}$$

[3] In the compressed sensing literature, the measurement matrix is often denoted $\mathbf{\Phi}$; instead, we use $\mathbf{C}$ to be consistent with the output equation in control theory. Also, $\mathbf{\Phi}$ is already used to denote DMD modes in Chapter 7.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

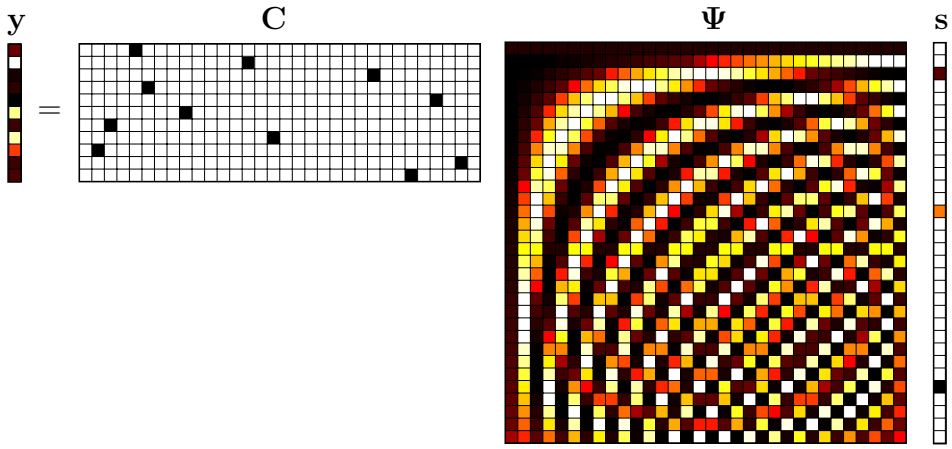


Figure 3.4: Schematic of measurements in the compressed sensing framework.

where $\|\cdot\|_1$ is the ℓ_1-norm, given by

$$\|\mathbf{s}\|_1 = \sum_{k=1}^{n} |s_k|. \tag{3.6}$$

The ℓ_1-norm is also known as the taxicab or Manhattan norm because it represents the distance a taxi would take between two points on a rectangular grid. The overview of compressed sensing is shown schematically in Fig. 3.4. The ℓ_1-minimum-norm solution is sparse, while the ℓ_2-minimum-norm solution is not, as shown in Fig. 3.5.

There are very specific conditions that must be met for the ℓ_1-minimization in (3.5) to converge with high probability to the sparsest solution in (3.4) [53, 151, 156]. These will be discussed in detail in Section 3.4, although they may be summarized as follows.

(a) The measurement matrix $\mathbf{C}$ must be *incoherent* with respect to the sparsifying basis $\mathbf{\Psi}$, meaning that the rows of $\mathbf{C}$ are not correlated with the columns of $\mathbf{\Psi}$.

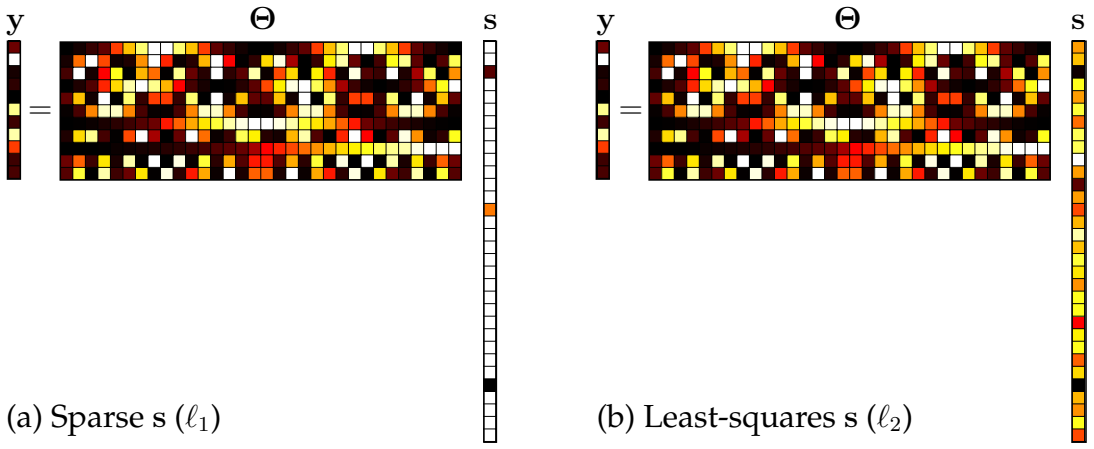
(b) The number of measurements p must be sufficiently large, on the order of

$$p \approx \mathcal{O}(K \log(n/K)) \approx k_1 K \log(n/K). \tag{3.7}$$

The constant multiplier k_1 depends on how incoherent $\mathbf{C}$ and $\mathbf{\Psi}$ are.

Roughly speaking, these two conditions guarantee that the matrix $\mathbf{C\Psi}$ acts as a unitary transformation on K-sparse vectors $\mathbf{s}$, preserving relative distances between vectors and enabling almost certain signal reconstruction with ℓ_1 convex

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.



(a) Sparse s (ℓ_1) (b) Least-squares s (ℓ_2)

Figure 3.5: The ℓ_1- and ℓ_2-minimum-norm solutions to the compressed sensing problem. The difference in solutions for this regression is further considered in Chapter 4.

minimization. This is formulated precisely in terms of the restricted isometry property (RIP) in Section 3.4.

The idea of compressed sensing may be counter-intuitive at first, especially given classical results on sampling requirements for exact signal reconstruction. For instance, the Shannon–Nyquist sampling theorem [536, 653] states that perfect signal recovery requires that it is sampled at twice the rate of the highest frequency present. However, this result only provides a strict bound on the required sampling rate for signals with broadband frequency content. Typically, the only signals that are truly broadband are those that have already been compressed. Since an uncompressed signal will generally be sparse in a transform basis, the Shannon–Nyquist theorem may be relaxed, and the signal may be reconstructed with considerably fewer measurements than given by the Nyquist rate. However, even though the number of measurements may be decreased, compressed sensing does still rely on precise *timing* of the measurements, as we will see. Moreover, the signal recovery via compressed sensing is not strictly speaking guaranteed, but is instead possible with high probability, making it foremost a statistical theory. However, the probability of successful recovery becomes astronomically large for moderate-sized problems.

Disclaimer

A rough schematic of compressed sensing is shown in Fig. 3.6. However, this schematic is a dramatization, and is not actually based on a compressed sensing

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

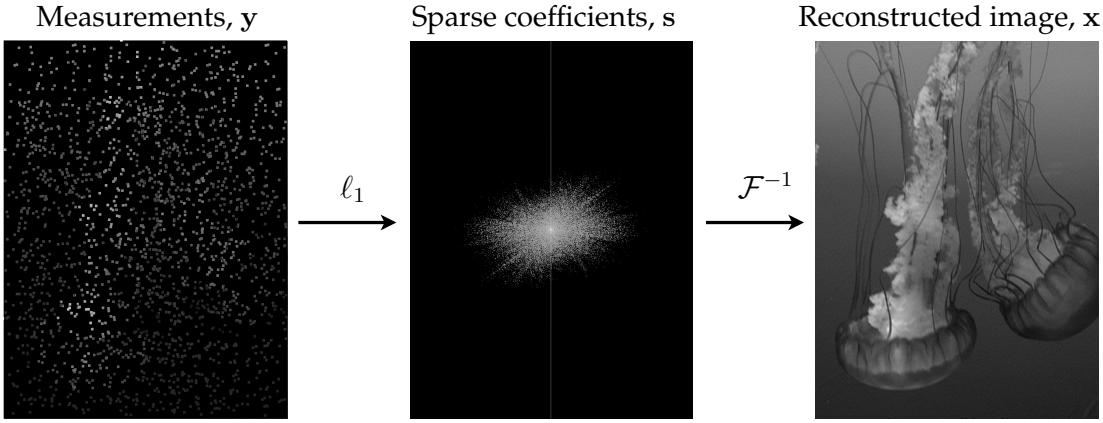


Figure 3.6: Schematic illustration of compressed sensing using ℓ_1-minimization. Note that this is a dramatization, and is not actually based on a compressed sensing calculation. Typically, compressed sensing of images requires a significant number of measurements and is computationally prohibitive.

calculation, since using compressed sensing for image reconstruction is computationally prohibitive. It is important to note that for the majority of applications in imaging, compressed sensing is not practicable. However, images are often still used to motivate and explain compressed sensing because of their ease of manipulation and our intuition for pictures. In fact, we are currently guilty of this exact misdirection.

Upon closer inspection of this image example, we are analyzing an image with 1024×768 pixels, and approximately $5\,\%$ of the Fourier coefficients are required for accurate compression. This puts the sparsity level at $K = 0.05 \times 1024 \times 768 \approx 40\,000$. Thus, a back-of-the-envelope estimate using (3.7), with a constant multiplier of $k_1 = 3$, indicates that we need $p \approx 350\,000$ measurements, which is about $45\,\%$ of the original pixels. Even if we had access to these $45\,\%$ random measurements, inferring the correct sparse vector of Fourier coefficients is computationally prohibitive, much more so than the efficient FFT-based image compression in Section 3.1.

Compressed sensing for images is typically only used in special cases where a reduction of the number of measurements is significant. For example, an early application of compressed sensing technology was for infant MRI (magnetic resonance imaging), where reduction of the time a child must be still could reduce the need for dangerous heavy sedation.

However, it is easy to see that the number of measurements p scales with the sparsity level K, so that if the signal is *more* sparse, then fewer measurements are required. The viewpoint of sparsity is still valuable, and the mathematical innovation of the convex relaxation of combinatorially hard ℓ_0 problems to convex ℓ_1 problems may be used much more broadly than for compressed sensing of images.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Alternative Formulations

In addition to the ℓ_1-minimization in (3.5), there are alternative approaches based on *greedy algorithms* [276, 277, 278, 323, 520, 713, 714, 715, 716, 717, 718, 719] that determine the sparse solution of (3.3) through an iterative matching pursuit problem. For instance, the compressed sensing matching pursuit (CoSaMP) [520] is computationally efficient, easy to implement, and freely available.

When the measurements $\mathbf{y}$ have additive noise, say white noise of magnitude ϵ, there are variants of (3.5) that are more robust:

$$\hat{\mathbf{s}} = \underset{\mathbf{s}}{\operatorname{argmin}} \|\mathbf{s}\|_1 \quad \text{subject to} \quad \|\mathbf{C\Psi s} - \mathbf{y}\|_2 < \epsilon. \tag{3.8}$$

A related convex optimization is the following:

$$\hat{\mathbf{s}} = \underset{\mathbf{s}}{\operatorname{argmin}} \|\mathbf{C\Psi s} - \mathbf{y}\|_2 + \lambda \|\mathbf{s}\|_1, \tag{3.9}$$

where $\lambda \geq 0$ is a parameter that weights the importance of sparsity. Equations (3.8) and (3.9) are closely related [716].

3.3 Compressed Sensing Examples

This section explores concrete examples of compressed sensing for sparse signal recovery. The first example shows that the ℓ_1-norm promotes sparsity when solving a generic under-determined system of equations, and the second example considers the recovery of a sparse two-tone audio signal with compressed sensing.

The ℓ_1-norm and Sparse Solutions to an Under-determined System

To see the sparsity-promoting effects of the ℓ_1-norm, we consider a generic under-determined system of equations. We build a matrix system of equations $\mathbf{y} = \mathbf{\Theta s}$ with $p = 200$ rows (measurements) and $n = 1000$ columns (unknowns). In general, there are infinitely many solutions $\mathbf{s}$ that are consistent with these equations, unless we are very unfortunate and the row equations are linearly dependent while the measurements are inconsistent in these rows. In fact, this is an excellent example of the probabilistic thinking used more generally in compressed sensing: if we generate a linear system of equations at random, that has sufficiently many more unknowns than knowns, then the resulting equations will have infinitely many solutions with *high probability*.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

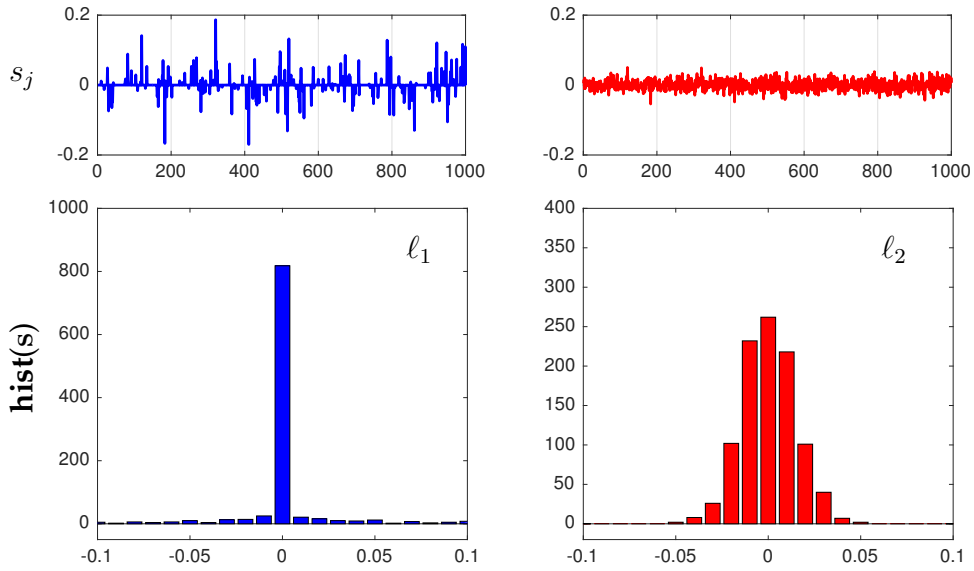


Figure 3.7: Comparison of ℓ_1-minimum-norm (blue, left) and ℓ_2-minimum-norm (red, right) solutions to an under-determined linear system.

In MATLAB, it is straightforward to solve this under-determined linear system for both the minimum ℓ_1-norm and minimum ℓ_2-norm solutions. The minimum ℓ_2-norm solution is obtained using the pseudo-inverse (related to the SVD from Chapters 1 and 4). The minimum ℓ_1-norm solution is obtained via the **cvx** (ConVeX) optimization package [293]. Figure 3.7 shows that the ℓ_1-minimum solution is in fact sparse (with most entries being nearly zero), while the ℓ_2-minimum solution is *dense*, with a bit of energy in each vector coefficient.

Code 3.2: [MATLAB] Solutions to under-determined linear system $\mathbf{y} = \mathbf{\Theta s}$.

```
% Solve y = Theta * s for "s"
n = 1000;  % dimension of s
p = 200;   % number of measurements, dim(y)
Theta = randn(p,n);
y = randn(p,1);

% L1 minimum norm solution s_L1
cvx_begin;
    variable s_L1(n);
    minimize( norm(s_L1,1) );
    subject to
        Theta*s_L1 == y;
cvx_end;

s_L2 = pinv(Theta)*y;  % L2 minimum norm solution s_L2
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Code 3.2: [Python] Solutions to under-determined linear system $\mathbf{y} = \boldsymbol{\Theta}\mathbf{s}$.

```
from scipy.optimize import minimize
# Solve y = Theta * s for "s"
n = 1000 # dimension of s
p = 200  # number of measurements, dim(y)
Theta = np.random.randn(p,n)
y = np.random.randn(p)

# L1 Minimum norm solution s_L1
def L1_norm(x):
    return np.linalg.norm(x,ord=1)

constr = ({'type': 'eq', 'fun': lambda x:  Theta @ x - y})
x0 = np.linalg.pinv(Theta) @ y # initialize with L2 solution
res = minimize(L1_norm, x0, method='SLSQP',constraints=
   constr)
s_L1 = res.x
```

Recovering an Audio Signal from Sparse Measurements

To illustrate the use of compressed sensing to reconstruct a high-dimensional signal from a sparse set of random measurements, we consider a signal consisting of a two-tone audio signal:

$$x(t) = \cos(2\pi \times 97t) + \cos(2\pi \times 777t). \tag{3.10}$$

This signal is clearly sparse in the frequency domain, as it is defined by a sum of exactly two cosine waves. The highest frequency present is $777\,\text{Hz}$, so that the Nyquist sampling rate is $1554\,\text{Hz}$. However, leveraging the sparsity of the signal in the frequency domain, we can accurately reconstruct the signal with random samples that are spaced at an average sampling rate of $128\,\text{Hz}$, which is well below the Nyquist sampling rate. Figure 3.8 shows the result of compressed sensing, as implemented in Code 3.3. In this example, the full signal is generated from $t = 0$ to $t = 1$ with a resolution of $n = 4096$ and is then randomly sampled at $p = 128$ locations in time. The sparse vector of coefficients in the discrete cosine transform (DCT) basis is solved for using matching pursuit.

Code 3.3: [MATLAB] Compressed sensing of two-tone cosine signal.

```
%% Generate signal, DCT of signal
n = 4096;    % points in high resolution signal
t = linspace(0, 1, n);
x = cos(2* 97 * pi * t) + cos(2* 777 * pi * t);
xt = fft(x); % Fourier transformed signal
PSD = xt.*conj(xt)/n;  % Power spectral density
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

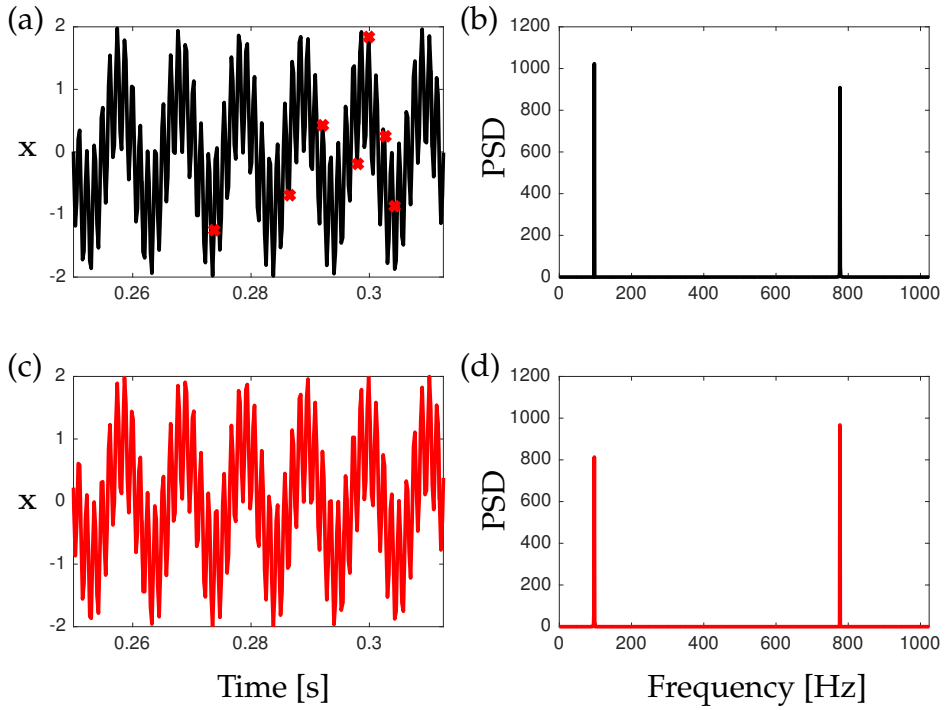


Figure 3.8: Compressed sensing reconstruction of a two-tone audio signal given by $x(t) = \cos(2\pi \times 97t) + \cos(2\pi \times 777t)$. The full signal and power spectral density are shown in panels (a) and (b), respectively. The signal is measured at random sparse locations in time, demarcated by red points in (a), and these measurements are used to build the compressed sensing estimate in (c) and (d). The time series shown in (a) and (c) are a zoom-in of the entire time range, which is from $t = 0$ to $t = 1$.

```
%% Randomly sample signal
p = 128; % num. random samples, p=n/32
perm = round(rand(p, 1) * n);
y = x(perm); % compressed measurement

%% Solve compressed sensing problem
Psi = dct(eye(n, n));  % build Psi
Theta = Psi(perm, :);  % Measure rows of Psi

s = cosamp(Theta,y',10,1.e-10,10); % CS via matching pursuit
xrecon = idct(s);      % reconstruct full signal
```

Code 3.3: [Python] Compressed sensing of two-tone cosine signal.

```
from cosamp_fn import cosamp
## Generate signal, DCT of signal
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

```
n = 4096 # points in high resolution signal
t = np.linspace(0,1,n)
x = np.cos(2 * 97 * np.pi * t) + np.cos(2 * 777 * np.pi * t)
xt = np.fft.fft(x) # Fourier transformed signal
PSD = xt * np.conj(xt) / n # Power spectral density

## Randomly sample signal
p = 128 # num. random samples, p = n/32
perm = np.floor(np.random.rand(p) * n).astype(int)
y = x[perm]

## Solve compressed sensing problem
Psi = dct(np.identity(n)) # Build Psi
Theta = Psi[perm,:]       # Measure rows of Psi
# CS via matching pursuit
s = cosamp(Theta,y,10,epsilon=1.e-10,max_iter=10)
xrecon = idct(s) # reconstruct full signal
```

It is important to note that the $p = 128$ measurements are randomly chosen from the 4096 resolution signal. Thus, we know the precise timing of the sparse measurements at a much higher resolution than our sampling rate. If we chose $p = 128$ measurements uniformly in time, the compressed sensing algorithm fails. Specifically, if we compute the PSD directly from these uniform measurements, the high-frequency signal will be aliased, resulting in erroneous frequency peaks.

In the compressed sensing matching pursuit (CoSaMP) code, the desired level of sparsity K must be specified, and this quantity may not be known ahead of time. The alternative ℓ_1-minimization routine does not require knowledge of the desired sparsity level *a priori*, although convergence to the sparsest solution relies on having sufficiently many measurements p, which indirectly depends on K.

3.4 The Geometry of Compression

Compressed sensing can be summarized in a relatively simple statement: A given signal, if it is sufficiently sparse in a known basis, may be recovered (with high probability) using significantly fewer measurements than the signal length, if there are sufficiently many measurements and these measurements are sufficiently random. Each part of this statement can be made precise and mathematically rigorous in an overarching framework that describes the geometry of sparse vectors, and how these vectors are transformed through random measurements. Specifically, enough good measurements will result in a matrix

$$\mathbf{\Theta} = \mathbf{C}\mathbf{\Psi} \tag{3.11}$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

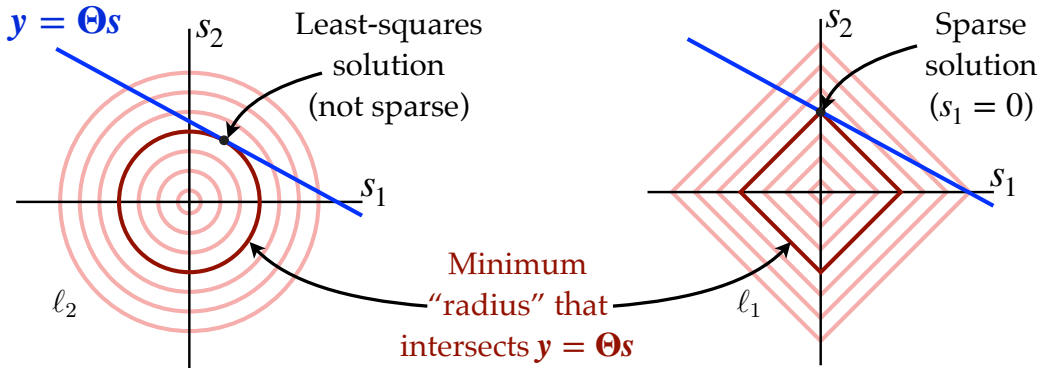


Figure 3.9: Different ℓ_p-norms establish different shapes for level sets of constant "radius" or distance from the origin. In the ℓ_2-norm, these correspond to circles, whereas in the ℓ_1 case, these are diamonds. The blue line represents the solution set of an under-determined system of equations $\mathbf{y} = \mathbf{\Theta s}$, and the red curves represent the minimum-norm level sets that intersect this blue line for different norms. In the ℓ_2-norm, the minimum-norm solution is not sparse, as it has components of s_1 and s_2, while in the ℓ_1-norm, the solution is sparse, as $s_1 = 0$.

that preserves the distance and inner product structure of sparse vectors $\mathbf{s}$. In other words, we seek a measurement matrix $\mathbf{C}$ so that $\mathbf{\Theta}$ acts as a near-isometry map on sparse vectors. Isometry literally means *same distance*, and is closely related to unitarity, which preserves not only distance, but also angles between vectors. When $\mathbf{\Theta}$ acts as a near-isometry, it is possible to solve the following equation for the sparsest vector $\mathbf{s}$ using convex ℓ_1-minimization:

$$\mathbf{y} = \mathbf{\Theta s}. \tag{3.12}$$

The remainder of this section describes the conditions on the measurement matrix $\mathbf{C}$ that are required for $\mathbf{\Theta}$ to act as a near-isometry map with high probability. The geometric properties of various norms are shown in Figs. 3.9 and 3.10.

Determining how many measurements to take is relatively simple. If the signal is K-sparse in a basis $\mathbf{\Psi}$, meaning that all but K coefficients are zero, then the number of measurements scales as $p \sim \mathcal{O}(K \log(n/K)) = k_1 K \log(n/K)$, as in (3.7). The constant multiplier k_1, which defines *exactly* how many measurements are needed, depends on the quality of the measurements. Roughly speaking, measurements are good if they are *incoherent* with respect to the columns of the sparsifying basis, meaning that the rows of $\mathbf{C}$ have small inner product with the columns of $\mathbf{\Psi}$. If the measurements are coherent with columns of the sparsifying basis, then a measurement will provide little information unless that basis mode happens to be non-zero in $\mathbf{s}$. In contrast, incoherent measurements are excited by nearly any active mode, making it possible to infer the active modes. Delta functions are incoherent with respect to Fourier

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

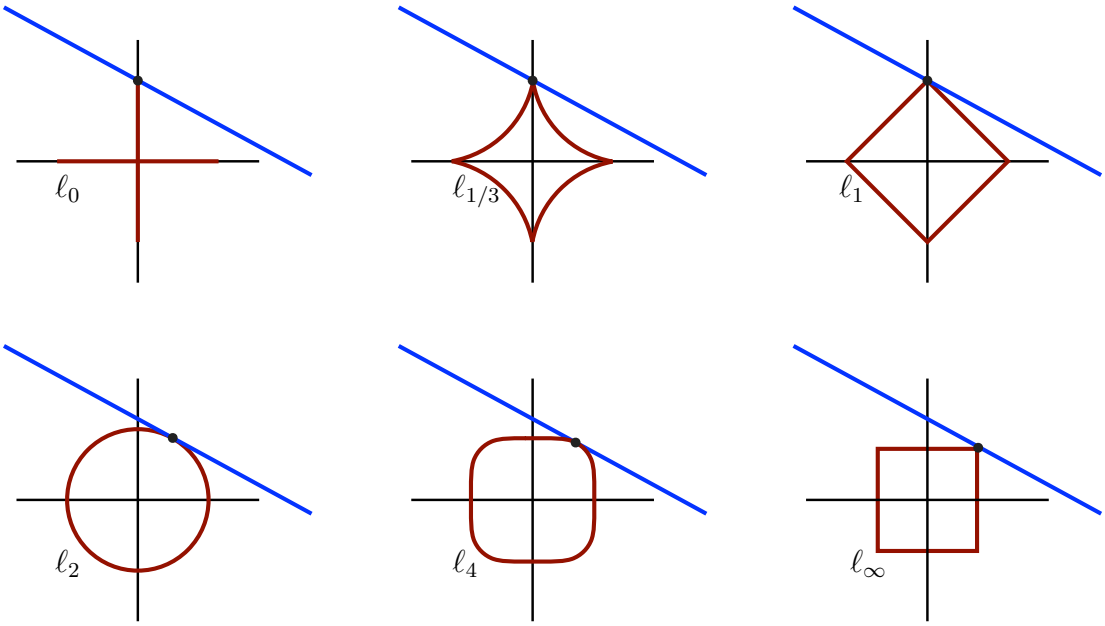


Figure 3.10: The minimum-norm point on a line in different ℓ_p-norms. The blue line represents the solution set of an under-determined system of equations, and the red curves represent the minimum-norm level sets that intersect this blue line for different norms. In the norms between ℓ_0 and ℓ_1, the minimum-norm solution also corresponds to the sparsest solution, with only one coordinate active. In the ℓ_2 and higher norms, the minimum-norm solution is not sparse, but has all coordinates active.

modes, as they excite a broadband frequency response. The more *incoherent* the measurements, the smaller the required number of measurements p.

The incoherence of measurements $\mathbf{C}$ and the basis $\mathbf{\Psi}$ is given by $\mu(\mathbf{C}, \mathbf{\Psi})$:

$$\mu(\mathbf{C}, \mathbf{\Psi}) = \sqrt{n} \max_{j,k} |\langle \mathbf{c}_k, \boldsymbol{\psi}_j \rangle|, \tag{3.13}$$

where $\mathbf{c}_k$ is the kth row of the matrix $\mathbf{C}$ and $\boldsymbol{\psi}_j$ is the jth column of the matrix $\mathbf{\Psi}$. The incoherence μ will range between 1 and $\sqrt{n}$. The formula for incoherence above only makes sense when the rows of $\mathbf{C}$ and columns of $\mathbf{\Psi}$ are normalized to have unit length.

The Restricted Isometry Property (RIP)

When measurements are incoherent, the matrix $\mathbf{C\Psi}$ satisfies a *restricted isometry property* (RIP) for sparse vectors $\mathbf{s}$,

$$(1 - \delta_K)\|\mathbf{s}\|_2^2 \leq \|\mathbf{C\Psi s}\|_2^2 \leq (1 + \delta_K)\|\mathbf{s}\|_2^2,$$

with restricted isometry constant δ_K [154]. The constant δ_K is defined as the smallest number that satisfies the above inequality for *all* K-sparse vectors $\mathbf{s}$.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

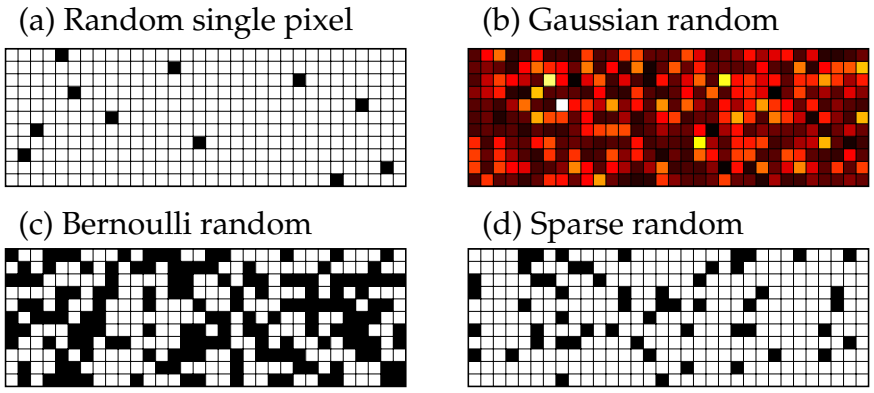


Figure 3.11: Examples of good random measurement matrices $\mathbf{C}$.

When δ_K is small, then $\mathbf{C\Psi}$ acts as a near-isometry on K-sparse vectors $\mathbf{s}$. In practice, it is difficult to compute δ_K directly; moreover, the measurement matrix $\mathbf{C}$ may be chosen to be random, so that it is more desirable to derive statistical properties about the bounds on δ_K for a family of measurement matrices $\mathbf{C}$, rather than to compute δ_K for a specific $\mathbf{C}$. Generally, increasing the number of measurements will decrease the constant δ_K, improving the property of $\mathbf{C\Psi}$ to act isometrically on sparse vectors. When there are sufficiently many incoherent measurements, as described above, it is possible to accurately determine the K non-zero elements of the n-length vector $\mathbf{s}$. In this case, there are bounds on the constant δ_K that guarantee exact signal reconstruction for noiseless data. An in-depth discussion of incoherence and the RIP can be found in [53, 154].

Incoherence and Measurement Matrices

Another significant result of compressed sensing is that there are generic sampling matrices $\mathbf{C}$ that are sufficiently incoherent with respect to nearly all transform bases. Specifically, Bernoulli and Gaussian random measurement matrices satisfy the RIP for a generic basis $\mathbf{\Psi}$ with high probability [153]. There are additional results generalizing the RIP and investigating incoherence of sparse matrices [276].

In many engineering applications, it is advantageous to represent the signal $\mathbf{x}$ in a generic basis, such as Fourier or wavelets. One key advantage is that single-point measurements are incoherent with respect to these bases, exciting a broadband frequency response. Sampling at random point locations is appealing in applications where individual measurements are expensive, such as in ocean monitoring. Examples of random measurement matrices, including single-pixel, Gaussian, Bernoulli, and sparse random, are shown in Fig. 3.11.

A particularly useful transform basis for compressed sensing is obtained by the SVD,[4] resulting in a tailored basis in which the data is optimally sparse

[4]The SVD provides an optimal low-rank matrix approximation, and it is used in principal

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

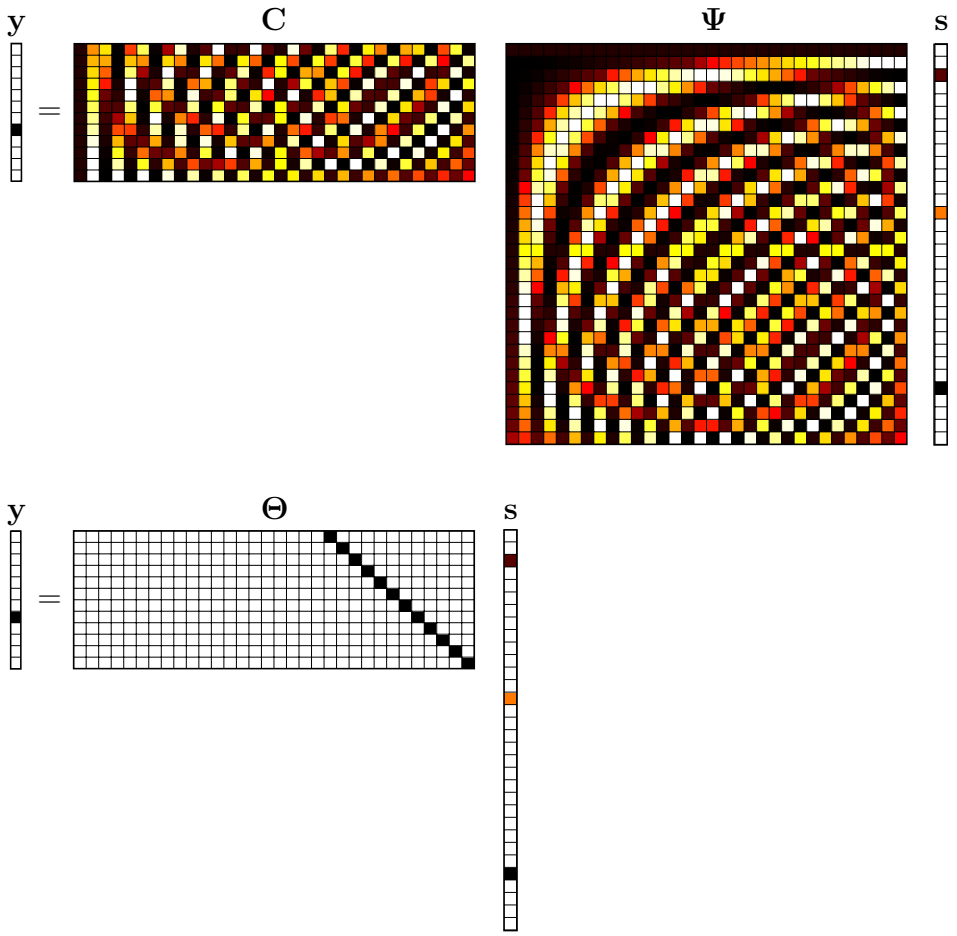


Figure 3.12: Examples of a bad measurement matrix $\mathbf{C}$.

[42, 112, 113, 136, 420]. A truncated SVD basis may result in a more efficient signal recovery from fewer measurements. Progress has been made developing a compressed SVD and PCA based on the Johnson–Lindenstrauss (JL) lemma [250, 277, 351, 571]. The JL lemma is closely related to the RIP, indicating when it is possible to embed high-dimensional vectors in a low-dimensional space while preserving spectral properties.

Bad Measurements

So far we have described how to take *good* compressed measurements. Figure 3.12 shows a particularly poor choice of measurements $\mathbf{C}$, corresponding to the last p columns of the sparsifying basis $\boldsymbol{\Psi}$. In this case, the product $\boldsymbol{\Theta} = \mathbf{C}\boldsymbol{\Psi}$ is a $p \times p$ identity matrix padded with zeros on the left. In this case, any signal $\mathbf{s}$ that is not active in the last p columns of $\boldsymbol{\Psi}$ is in the null space of $\boldsymbol{\Theta}$, and is completely invisible to the measurements $\mathbf{y}$. In this case, these measurements

component analysis (PCA) and proper orthogonal decomposition (POD).

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

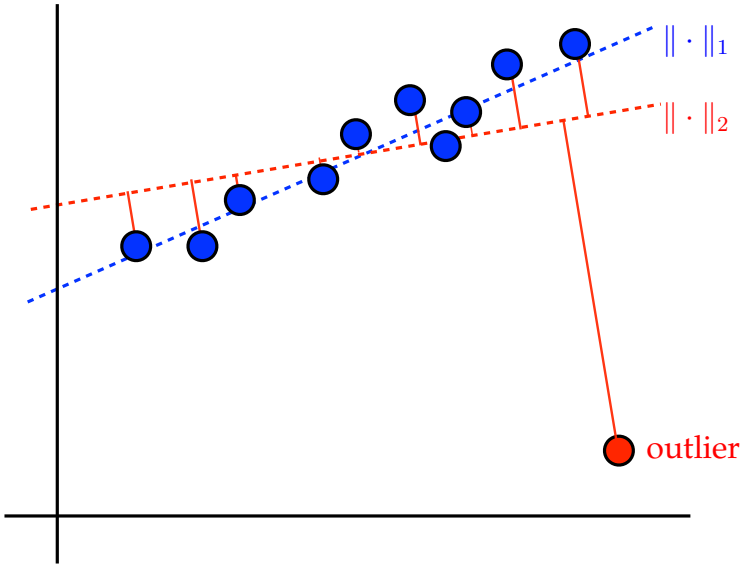


Figure 3.13: Least-squares regression is sensitive to outliers (red), while minimum ℓ_1-norm regression is robust to outliers (blue).

incur significant information loss for many sparse vectors.

3.5 Sparse Regression

The use of the ℓ_1-norm to promote sparsity significantly pre-dates compressed sensing. In fact, many benefits of the ℓ_1-norm were well known and oft used in statistics decades earlier. In this section, we show that the ℓ_1-norm may be used to *regularize* statistical regression, both to penalize statistical outliers and also to promote *parsimonious* statistical models with as few factors as possible. The role of ℓ_2 versus ℓ_1 in regression is further detailed in Chapter 4.

Outlier Rejection and Robustness

Least-squares regression is perhaps the most common statistical model used for data fitting. However, it is well known that the regression fit may be arbitrarily corrupted by a single large outlier in the data; outliers are weighted more heavily in least-squares regression because their distance from the fit line is squared. This is shown schematically in Fig. 3.13.

In contrast, ℓ_1-minimum solutions give equal weight to all data points, making it potentially more robust to outliers and corrupt data. This procedure is also known as least absolute deviations (LAD) regression, among other names. A script demonstrating the use of least-squares (ℓ_2) and LAD (ℓ_1) regression for a data set with an outlier is given in Code 3.4.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Code 3.4: [MATLAB] Use of ℓ_1-norm for robust statistical regression.

```
x = sort(4*(rand(25,1)-.5)); % Random data from [-2,2]
b = .9*x + .1*randn(size(x)); % Line y=.9x with noise
atrue = x\b;           % Least-squares slope (no outliers)

b(end) = -5.5;                  % Introduce outlier
acorrupt = x\b;                 % New slope

cvx_begin;        % L1 optimization to reject outlier
    variable aL1;     % aL1 is slope to be optimized
    minimize( norm(aL1*x-b,1) );     % aL1 is robust
cvx_end;
```

Code 3.4: [Python] Use of ℓ_1-norm for robust statistical regression.

```
x = np.sort(4*(np.random.rand(25,1)-0.5),axis=0) # data
b = 0.9*x + 0.1*np.random.randn(len(x),1) # Noisy line y=ax
atrue = np.linalg.lstsq(x,b,rcond=None)[0] # Least-squares a

b[-1] = -5.5  # Introduce outlier
acorrupt = np.linalg.lstsq(x,b,rcond=None)[0] # New slope

## L1 optimization to reject outlier
def L1_norm(a):
    return np.linalg.norm(a*x-b,ord=1)

a0 = acorrupt   # initialize to L2 solution
res = minimize(L1_norm, a0)
aL1 = res.x[0]  # aL1 is robust
```

Feature Selection and LASSO Regression

Interpretability is important in statistical models, as these models are often communicated to a non-technical audience, including business leaders and policy makers. Generally, a regression model is more interpretable if it has fewer terms that bear on the outcome, motivating yet another perspective on sparsity.

The least absolute shrinkage and selection operator (LASSO) is an ℓ_1 penalized regression technique that balances model complexity with descriptive capability [702]. This principle of *parsimony* in a model is also a reflection of Occam's razor, stating that, among all possible descriptions, the simplest correct model is probably the true one. Since its inception by Tibshirani in 1996 [702], the LASSO has become a cornerstone of statistical modeling, with many modern variants and related techniques [315, 348, 760]. The LASSO is closely related to the earlier non-negative garrote of Breimen [107], and is also related to earlier work on soft thresholding by Donoho and Johnstone [207, 208]. LASSO may be

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

thought of as a sparsity-promoting regression that benefits from the stability of the ℓ_2 regularized ridge regression [332], also known as Tikhonov regularization. The elastic net is a frequently used regression technique that combines the ℓ_1 and ℓ_2 penalty terms from LASSO and ridge regression [777]. Sparse regression will be explored in more detail in Chapter 4.

Given a number of observations of the predictors and outcomes of a system, arranged as rows of a matrix $\mathbf{A}$ and a vector $\mathbf{b}$, respectively, regression seeks to find the relationship between the columns of $\mathbf{A}$ that is most consistent with the outcomes in $\mathbf{b}$. Mathematically, this may be written as

$$\mathbf{A}\mathbf{x} = \mathbf{b}. \tag{3.14}$$

Least-squares regression will tend to result in a vector $\mathbf{x}$ that has non-zero coefficients for all entries, indicating that *all* columns of $\mathbf{A}$ must be used to predict $\mathbf{b}$. However, we often believe that the statistical model should be *simpler*, indicating that $\mathbf{x}$ may be sparse. The LASSO adds an ℓ_1 penalty term to *regularize* the least-squares regression problem, i.e., to prevent overfitting:

$$\mathbf{x} = \underset{\mathbf{x}'}{\operatorname{argmin}} \|\mathbf{A}\mathbf{x}' - \mathbf{b}\|_2 + \lambda\|\mathbf{x}\|_1. \tag{3.15}$$

Typically, the parameter λ is varied through a range of values and the fit is *validated* against a test set of holdout data. If there is not enough data to have a sufficiently large training set and test set, it is common to repeatedly train and test the model on random selection of the data (often $80\,\%$ for training and $20\,\%$ for testing), resulting in a *cross-validated* performance. This cross-validation procedure enables the selection of a parsimonious model that has relatively few terms and avoids overfitting.

Many statistical systems are over-determined, as there are more observations than candidate predictors. Thus, it is not possible to use standard compressed sensing, as measurement noise will guarantee that no exact sparse solution exists that minimizes $\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2$. However, the LASSO regression works well with over-determined problems, making it a general regression method. Note that an early version of the geometric picture in Fig. 3.10 to explain the sparsity-promoting nature of the ℓ_1-norm was presented in Tibshirani's 1996 paper [702].

LASSO regression is frequently used to build statistical models for disease, such as cancer and heart failure, since there are many possible predictors, including demographics, lifestyle, biometrics, and genetic information. Thus, LASSO represents a clever version of the *kitchen-sink* approach, whereby nearly all possible predictive information is thrown into the mix, and afterwards these are then sifted and sieved through for the truly relevant predictors.

As a simple example, we consider an artificial data set consisting of 100 observations of an outcome, arranged in a vector $\mathbf{b} \in \mathbb{R}^{100}$. Each outcome in $\mathbf{b}$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

is given by a combination of exactly two out of 10 candidate predictors, whose observations are arranged in the rows of a matrix $\mathbf{A} \in \mathbb{R}^{100\times 10}$:

```
A = randn(100,10);      % Matrix of possible predictors
x = [0; 0; 1; 0; 0; 0; -1; 0; 0; 0]; % 2 nonzero predictors
b = A*x + 2*randn(100,1);      % Observations (with noise)
```

The vector x is sparse by construction, with only two non-zero entries, and we also add noise to the observations in b. The least-squares regression is:

```
>>xL2 = pinv(A)*b

  xL2 = -0.0232
        -0.3395
         0.9591
        -0.1777
         0.2912
        -0.0525
        -1.2720
        -0.0411
         0.0413
        -0.0500
```

Note that all coefficients are non-zero. Implementing the LASSO, with 10-fold cross-validation, is a single straightforward command in MATLAB:

```
[XL1 FitInfo] = lasso(A,b,'CV',10);
```

The **lasso** command sweeps through a range of values for λ, and the resulting x are each stored as columns of the matrix in **XL1**. To select the most parsimonious model that describes the data while avoiding overfitting, we may plot the cross-validated error as a function of λ, as in Fig. 3.14:

```
lassoPlot(XL1,FitInfo,'PlotType','CV')
```

The green point is at the value of λ that minimizes the cross-validated mean-square error, and the blue point is at the minimum cross-validated error plus one standard deviation. The resulting model is found via **FitInfo.Index1SE**:

```
>> xL1 = XL1(:,FitInfo.Index1SE)

  xL1 =    0
           0
      0.7037
           0
           0
           0
     -0.4929
           0
           0
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

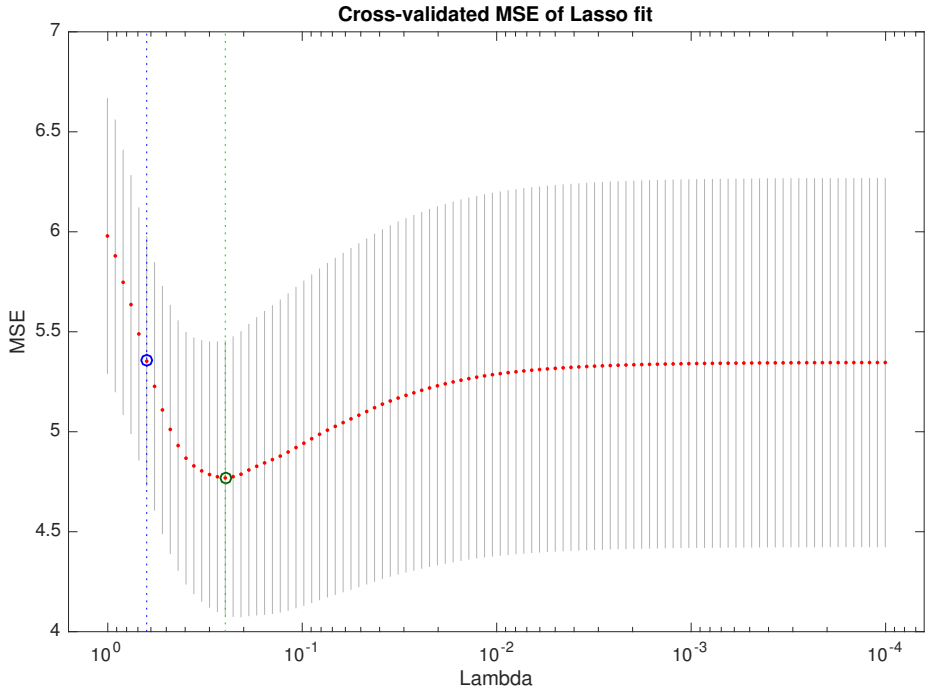


Figure 3.14: Output of **lassoPlot** command to visualize cross-validated mean-squared error (MSE) as a function of λ.

```
              0
```

Note that the resulting model is sparse and the correct terms are active. However, the regression values for these terms are not accurate, and so it may be necessary to *de-bias* the LASSO by applying a final least-squares regression to the non-zero coefficients identified:

```
>>xL1DeBiased = pinv(A(:,abs(xL1)>0))*b
  xL1DeBiased =  1.0980
                -1.0671
```

In Python, the LASSO and associated analysis functions are also simple:

```
from sklearn import linear_model
from sklearn import model_selection
reg = linear_model.LassoCV(cv=10).fit(A, b)
lasso = linear_model.Lasso(random_state=0, max_iter=10000)
alphas = np.logspace(-4, -0.5, 30)

tuned_parameters = [{'alpha': alphas}]

clf = model_selection.GridSearchCV(lasso, tuned_parameters,
   cv=10, refit=False)
clf.fit(A, b)
```

Related plotting commands to reproduce this example in Python are provided on the book's GitHub.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

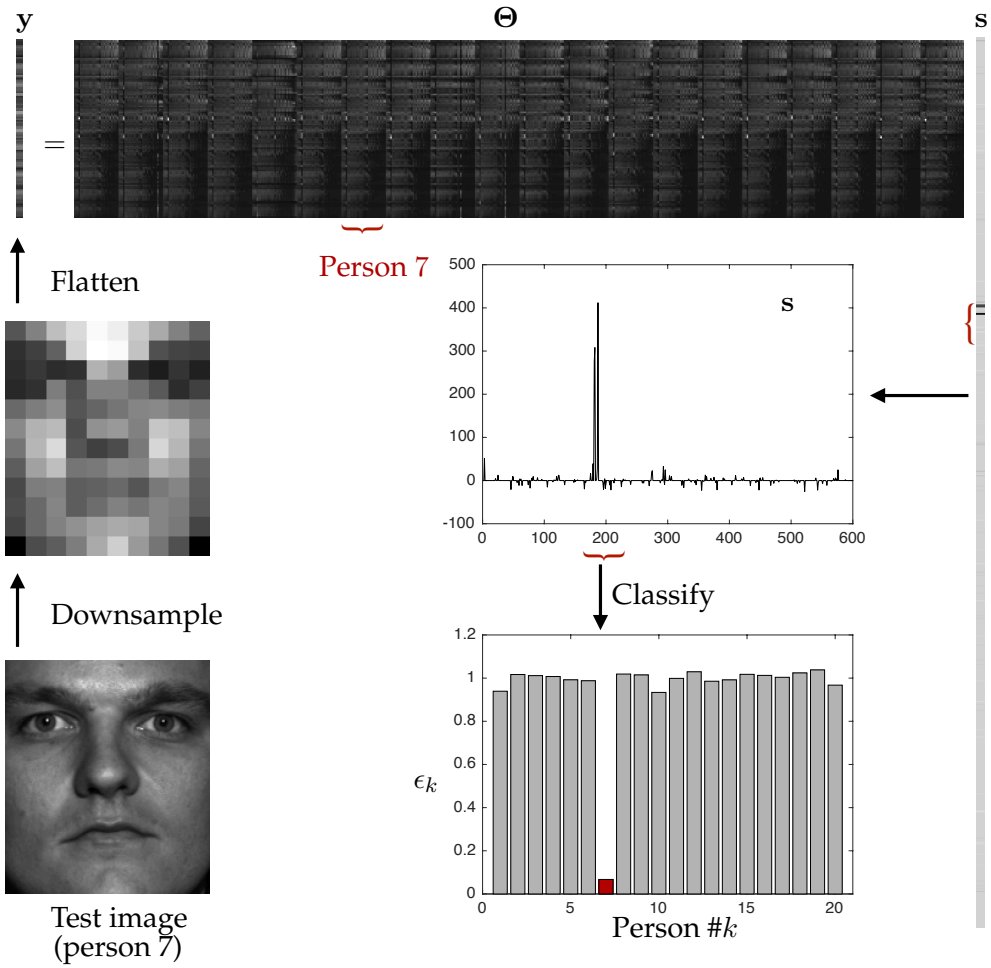


Figure 3.15: Schematic overview of sparse representation for classification.

3.6 Sparse Representation

Implicit in our discussion on sparsity is the fact that, when high-dimensional signals exhibit low-dimensional structure, they admit a *sparse representation* in an appropriate basis or dictionary. In addition to a signal being sparse in an SVD or Fourier basis, it may also be sparse in an overcomplete dictionary whose columns consist of the training data itself. In essence, in addition to a test signal being sparse in generic feature library $\mathbf{U}$ from the SVD, $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^*$, it may also have a sparse representation in the dictionary $\mathbf{X}$.

Wright et al. [762] demonstrated the power of sparse representation in a dictionary of test signals for robust classification of human faces, despite significant noise and occlusions. The so-called sparse representation for classification (SRC) has been widely used in image processing, and more recently to classify dynamical regimes in nonlinear differential equations [136, 147, 256, 411, 569].

The basic schematic of SRC is shown in Fig. 3.15, where a library of images of faces is used to build an overcomplete library $\mathbf{\Theta}$. In this example, 30 images

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

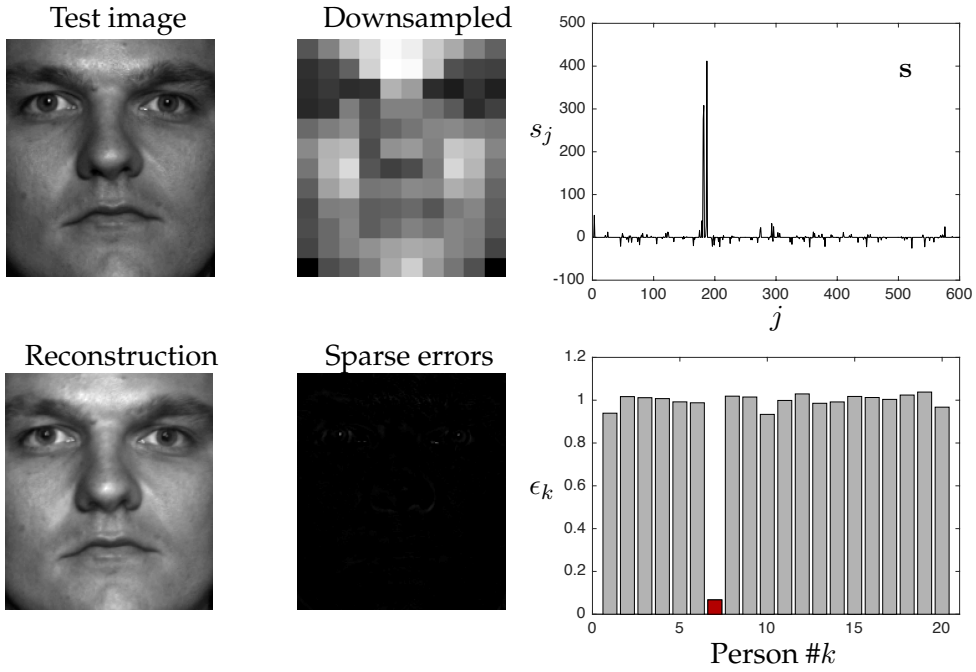


Figure 3.16: Sparse representation for classification demonstrated using a library of faces. A clean test image is correctly identified as the person 7 in the library.

are used for each of 20 different people in the Yale B database, resulting in 600 columns in $\boldsymbol{\Theta}$. To use compressed sensing, i.e., ℓ_1-minimization, we need $\boldsymbol{\Theta}$ to be under-determined, and so we downsample each image from 192×168 to 12×10, so that the flattened images are 120-component vectors. The algorithm used to downsample the images has an impact on the classification accuracy. A new test image $\mathbf{y}$ corresponding to class c, appropriately downsampled to match the columns of $\boldsymbol{\Theta}$, is then sparsely represented as a sum of the columns of $\boldsymbol{\Theta}$ using the compressed sensing algorithm. The resulting vector of coefficients $\mathbf{s}$ should be sparse, and ideally will have large coefficients primarily in the regions of the library corresponding to the correct person in class c. The final classification stage in the algorithm is achieved by computing the ℓ_2 reconstruction error using the coefficients in the $\mathbf{s}$ vector corresponding to each of the categories separately. The category that minimizes the ℓ_2 reconstruction error is chosen for the test image. Figures 3.16–3.19 show the use of SRC to correctly identify the correct person from images with different noise and corruption.

The entire code to reproduce this example in MATLAB and Python is available on the book's GitHub.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

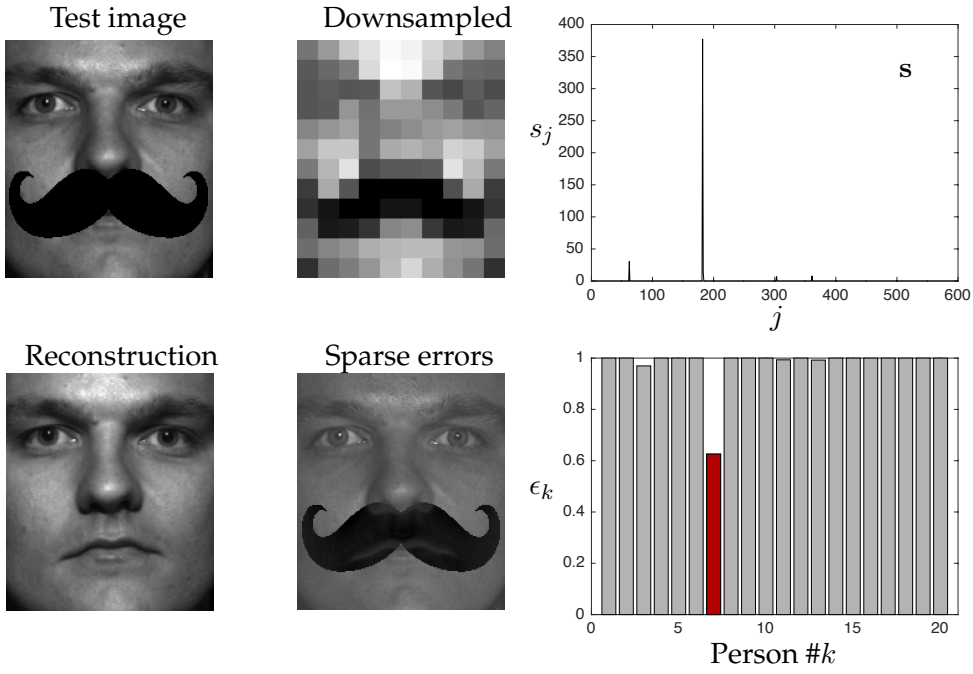


Figure 3.17: Sparse representation for classification demonstrated on example face from person 7 occluded by a fake mustache.

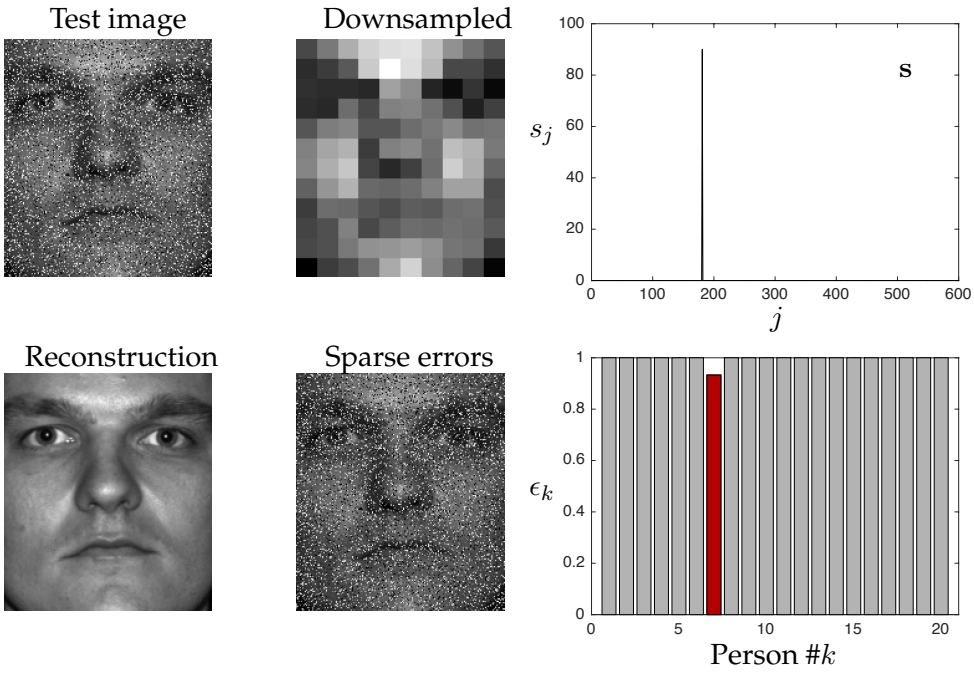


Figure 3.18: Sparse representation for classification demonstrated on example image with 30% occluded pixels (randomly chosen and uniformly distributed).

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

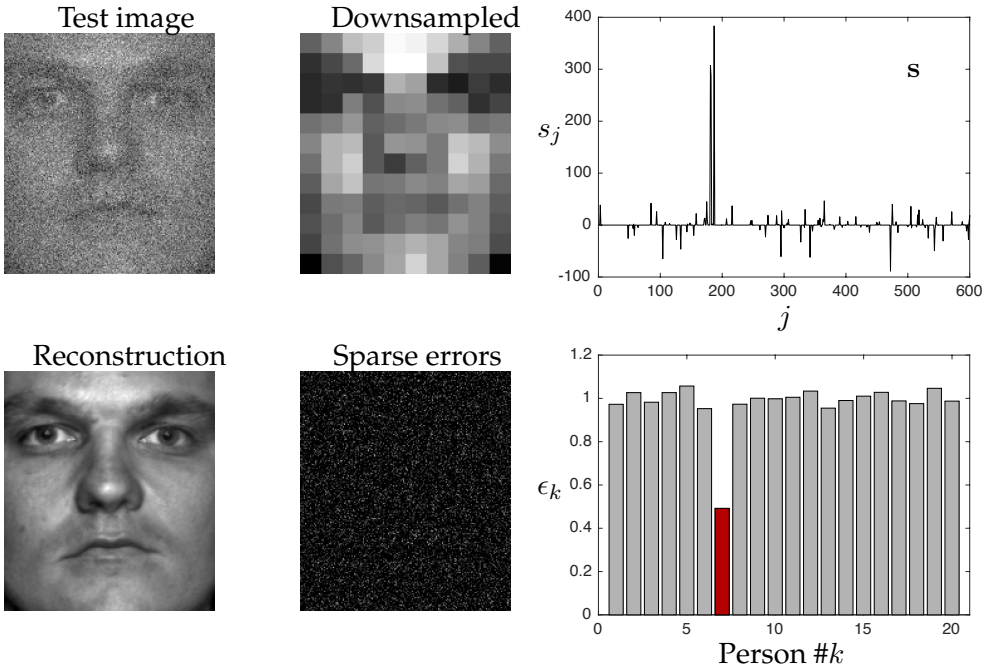


Figure 3.19: Sparse representation for classification demonstrated on example with white noise added to image.

3.7 Robust Principal Component Analysis (RPCA)

As mentioned earlier in Section 3.5, least-squares regression models are highly susceptible to outliers and corrupted data. Principal component analysis (PCA) suffers from the same weakness, making it *fragile* with respect to outliers. To ameliorate this sensitivity, Candès et al. [155] have developed a robust principal component analysis (RPCA) that seeks to decompose a data matrix $\mathbf{X}$ into a structured low-rank matrix $\mathbf{L}$ and a sparse matrix $\mathbf{S}$ containing outliers and corrupt data:

$$\mathbf{X} = \mathbf{L} + \mathbf{S}. \tag{3.16}$$

The principal components of $\mathbf{L}$ are *robust* to the outliers and corrupt data in $\mathbf{S}$. This decomposition has profound implications for many modern problems of interest, including video surveillance (where the background objects appear in $\mathbf{L}$ and foreground objects appear in $\mathbf{S}$), face recognition (eigenfaces are in $\mathbf{L}$ and shadows, occlusions, etc. are in $\mathbf{S}$), natural language processing and latent semantic indexing, and ranking problems.[5]

[5]The ranking problem may be thought of in terms of the Netflix prize for matrix completion. In the Netflix prize, a large matrix of preferences is constructed, with rows corresponding to users and columns corresponding to movies. This matrix is sparse, as most users only rate a handful of movies. The Netflix prize seeks to accurately fill in the missing entries of the matrix, revealing the likely user rating for movies the user has not seen.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Mathematically, the goal is to find $\mathbf{L}$ and $\mathbf{S}$ that satisfy the following:

$$\min_{\mathbf{L},\mathbf{S}} \text{rank}(\mathbf{L}) + \|\mathbf{S}\|_0 \quad \text{subject to} \quad \mathbf{L}+\mathbf{S}=\mathbf{X}. \tag{3.17}$$

However, neither the $\text{rank}(\mathbf{L})$ nor the $\|\mathbf{S}\|_0$ terms are convex, and this is not a tractable optimization problem. Similar to the compressed sensing problem, it is possible to solve for the optimal $\mathbf{L}$ and $\mathbf{S}$ with *high probability* using a convex relaxation of (3.17):

$$\min_{\mathbf{L},\mathbf{S}} \|\mathbf{L}\|_* + \lambda\|\mathbf{S}\|_1 \quad \text{subject to} \quad \mathbf{L}+\mathbf{S}=\mathbf{X}. \tag{3.18}$$

Here, $\|\cdot\|_*$ denotes the nuclear norm, given by the sum of singular values, which is a proxy for rank. Remarkably, the solution to (3.18) converges to the solution of (3.17) with high probability if $\lambda = 1/\sqrt{\max(n,m)}$, where n and m are the dimensions of $\mathbf{X}$, given that $\mathbf{L}$ and $\mathbf{S}$ satisfy the following conditions:

(a) $\mathbf{L}$ is not sparse; and

(b) $\mathbf{S}$ is not low-rank; we assume that the entries are randomly distributed so that they do not have low-dimensional column space.

The convex problem in (3.17) is known as *principal component pursuit* (PCP), and may be solved using the augmented Lagrange multiplier (ALM) algorithm. Specifically, an augmented Lagrangian may be constructed:

$$\mathcal{L}(\mathbf{L},\mathbf{S},\mathbf{Y}) = \|\mathbf{L}\|_* + \lambda\|\mathbf{S}\|_1 + \langle \mathbf{Y},\ \mathbf{X}-\mathbf{L}-\mathbf{S}\rangle + \frac{\mu}{2}\|\mathbf{X}-\mathbf{L}-\mathbf{S}\|_F^2. \tag{3.19}$$

A general solution would solve for the $\mathbf{L}_k$ and $\mathbf{S}_k$ that minimize $\mathcal{L}$, update the Lagrange multipliers $\mathbf{Y}_{k+1} = \mathbf{Y}_k + \mu(\mathbf{X}-\mathbf{L}_k-\mathbf{S}_k)$, and iterate until the solution converges. However, for this specific system, the alternating directions method (ADM) [447, 768] provides a simple procedure to find $\mathbf{L}$ and $\mathbf{S}$.

First, a shrinkage operator $\mathcal{S}_\tau(x) = \text{sign}(x)\max(|x|-\tau, 0)$ is constructed. In MATLAB, the function **shrink** is defined as

```
function out = shrink(X,tau)
    out = sign(X).*max(abs(X)-tau,0);
end
```

In Python, the function **shrink** is defined as

```
def shrink(X,tau):
    Y = np.abs(X)-tau
    return np.sign(X) * np.maximum(Y,np.zeros_like(Y))
```

Next, the singular value threshold operator $\text{SVT}_\tau(\mathbf{X}) = \mathbf{U}\mathcal{S}_\tau(\mathbf{\Sigma})\mathbf{V}^*$ is constructed. In MATLAB, the function **SVT** is defined as

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

```
function out = SVT(X,tau)
    [U,S,V] = svd(X,'econ');
    out = U*shrink(S,tau)*V';
end
```

In Python, the function **SVT** is defined as

```
def SVT(X,tau):
    U,S,VT = np.linalg.svd(X,full_matrices=0)
    out = U @ np.diag(shrink(S,tau)) @ VT
    return out
```

Finally, it is possible to use the $\mathcal{S}_\tau$ and SVT operators iteratively to solve for $\mathbf{L}$ and $\mathbf{S}$ as in Code 3.5.

Code 3.5: [MATLAB] RPCA using alternating directions method (ADM).

```
function [L,S] = RPCA(X)
[n1,n2] = size(X);
mu = n1*n2/(4*sum(abs(X(:))));
lambda = 1/sqrt(max(n1,n2));
thresh = 1e-7*norm(X,'fro');

L = zeros(size(X));
S = zeros(size(X));
Y = zeros(size(X));
count = 0;
while((norm(X-L-S,'fro')>thresh)&&(count<1000))
    L = SVT(X-S+(1/mu)*Y,1/mu);
    S = shrink(X-L+(1/mu)*Y,lambda/mu);
    Y = Y + mu*(X-L-S);
    count = count + 1
end
```

Code 3.5: [Python] RPCA using alternating directions method (ADM).

```
def RPCA(X):
    n1,n2 = X.shape
    mu = n1*n2/(4*np.sum(np.abs(X.reshape(-1))))
    lambd = 1/np.sqrt(np.maximum(n1,n2))
    thresh = 10**(-7) * np.linalg.norm(X)

    S = np.zeros_like(X)
    Y = np.zeros_like(X)
    L = np.zeros_like(X)
    count = 0
    while (np.linalg.norm(X-L-S) > thresh) and (count <
       1000):
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

```
        L = SVT(X-S+(1/mu)*Y,1/mu)
        S = shrink(X-L+(1/mu)*Y,lambd/mu)
        Y = Y + mu*(X-L-S)
        count += 1
    return L,S
```

RPCA is demonstrated on the eigenfaces data with the following code in MATLAB

```
load allFaces.mat
X = faces(:,1:nfaces(1));
[L,S] = RPCA(X);
```

and in Python

```
X = faces[:,:nfaces[0]]
L,S = RPCA(X)
```

In this example, the original columns of $\mathbf{X}$, along with the low-rank and sparse components, are shown in Fig. 3.20. Notice that in this example, RPCA effectively fills in occluded regions of the image, corresponding to shadows. In the low-rank component $\mathbf{L}$, shadows are removed and filled in with the most consistent low-rank features from the eigenfaces. This technique can also be used to remove other occlusions such as fake mustaches, sunglasses, or noise.

3.8 Sparse Sensor Placement

Until now, we have investigated signal reconstruction in a generic basis, such as Fourier or wavelets, with random measurements. This provides considerable flexibility, as no prior structure is assumed, except that the signal is sparse in a known basis. For example, compressed sensing works equally well for reconstructing an image of a mountain, a face, or a cup of coffee. However, if we know that we will be reconstructing a human face, we can dramatically reduce the number of sensors required for reconstruction or classification by optimizing sensors for a particular feature library $\mathbf{\Psi}_r = \tilde{\mathbf{U}}$ built from the SVD.

Thus, it is possible to design *tailored* sensors for a particular library, in contrast to the previous approach of random sensors in a generic library. Near-optimal sensor locations may be obtained using fast greedy procedures that scale well with large signal dimension, such as the matrix QR factorization. The following discussion will closely follow Manohar et al. [481] and B. Brunton et al. [122], and the reader is encouraged to find more details there. Similar approaches will be used for efficient sampling of reduced-order models in Chapter 13, where they are termed *hyper-reduction*. There are also extensions of the following for sensor and actuator placement in control [484], based on the balancing transformations discussed in Chapter 9.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

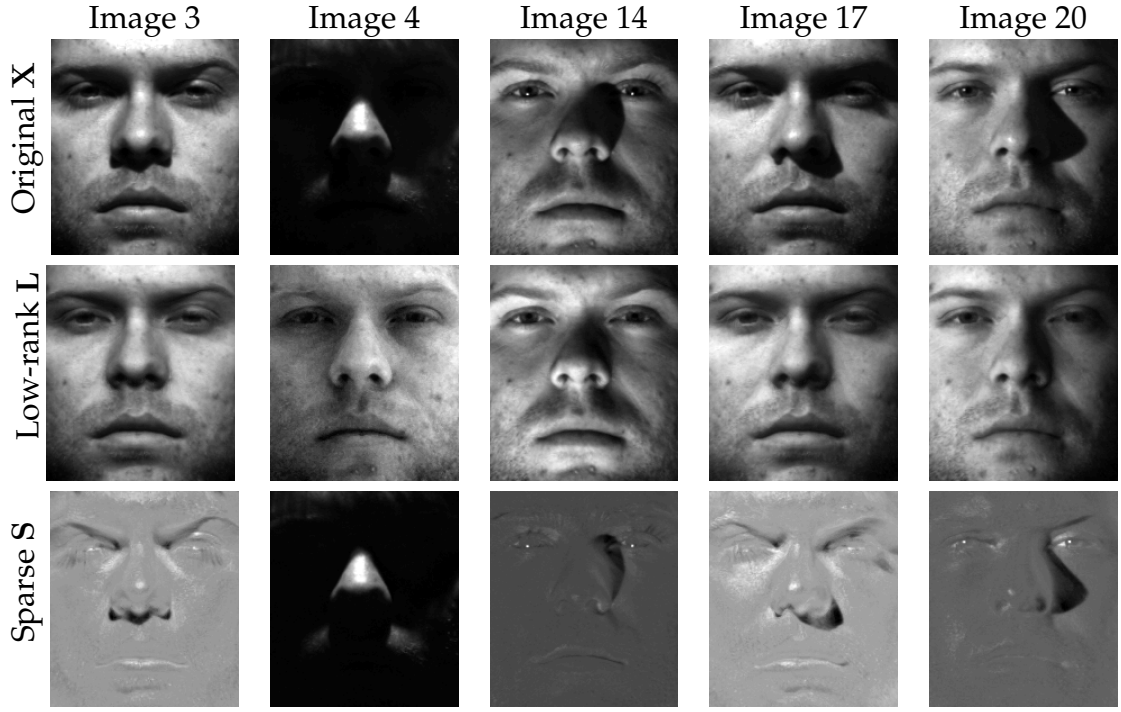


Figure 3.20: Output of RPCA for images in the Yale B database.

Optimizing sensor locations is important for nearly all downstream tasks, including classification, prediction, estimation, modeling, and control. However, identifying optimal locations involves a brute-force search through the combinatorial choices of p sensors out of n possible locations in space. Recent greedy and sparse methods are making this search tractable and scalable to large problems. Reducing the number of sensors through principled selection may be critically enabling when sensors are costly, and may also enable faster state estimation for low-latency, high-bandwidth control.

The examples in this section are in MATLAB. However, extensive Python code for sparse sensing is available at the following:
`https://github.com/dynamicslab/pysensors`.

Sparse Sensor Placement for Reconstruction

The goal of optimized sensor placement in a tailored library $\mathbf{\Psi}_r \in \mathbb{R}^{n\times r}$ is to design a sparse measurement matrix $\mathbf{C} \in \mathbb{R}^{p\times n}$, so that inversion of the linear system of equations

$$\mathbf{y} = \mathbf{C}\mathbf{\Psi}_r\mathbf{a} = \boldsymbol{\theta}\mathbf{a} \tag{3.20}$$

is as well conditioned as possible. In other words, we will design $\mathbf{C}$ to minimize the condition number of $\mathbf{C}\mathbf{\Psi}_r = \boldsymbol{\theta}$, so that it may be inverted to identify the

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

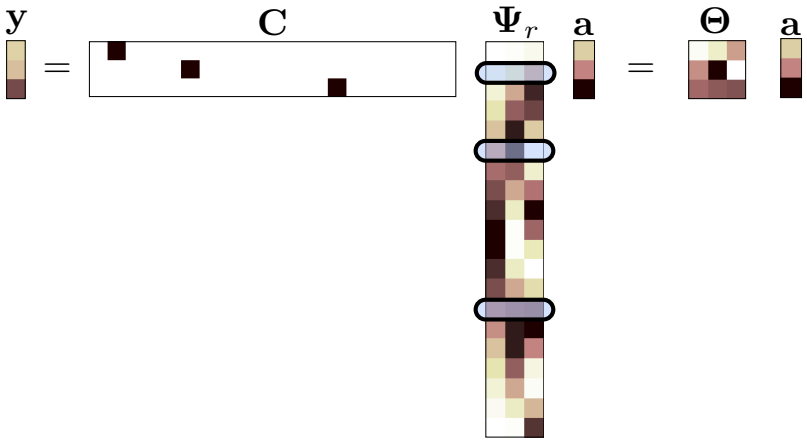


Figure 3.21: Least-squares with r sparse sensors provides a unique solution to $\mathbf{a}$, hence $\mathbf{x}$. Reproduced with permission from Manohar et al. [481].

low-rank coefficients $\mathbf{a}$ given noisy measurements $\mathbf{y}$. The condition number of a matrix $\boldsymbol{\theta}$ is the ratio of its maximum and minimum singular values, indicating how sensitive matrix multiplication or inversion is to errors in the input. Larger condition numbers indicate worse performance inverting a noisy signal. The condition number is a measure of the worst-case error when the signal $\mathbf{a}$ is in the singular vector direction associated with the minimum singular value of $\boldsymbol{\theta}$, and noise is added that is aligned with the maximum singular vector:

$$\boldsymbol{\theta}(\mathbf{a} + \epsilon_{\mathbf{a}}) = \sigma_{\min}\mathbf{a} + \sigma_{\max}\epsilon_{\mathbf{a}}. \tag{3.21}$$

Thus, the signal-to-noise ratio decreases by the condition number after mapping through $\boldsymbol{\theta}$. We therefore seek to minimize the condition number through a principled choice of $\mathbf{C}$. This is shown schematically in Fig. 3.21 for $p = r$.

When the number of sensors is equal to the rank of the library, i.e., $p = r$, then $\boldsymbol{\theta}$ is a square matrix, and we are choosing $\mathbf{C}$ to make this matrix as well conditioned for inversion as possible. When $p > r$, we seek to improve the condition of $\mathbf{M} = \boldsymbol{\theta}^T\boldsymbol{\theta}$, which is involved in the pseudo-inverse. It is possible to develop optimization criteria that optimize the minimum singular value, the trace, or the determinant of $\boldsymbol{\theta}$ (respectively $\mathbf{M}$). However, each of these optimization problems is NP-hard, requiring a combinatorial search over the possible sensor configurations. Iterative methods exist to solve this problem, such as convex optimization and semi-definite programming [101, 353], although these methods may be expensive, requiring iterative $n \times n$ matrix factorizations. Instead, greedy algorithms are generally used to approximately optimize the sensor placement. These *gappy POD* [239] methods originally relied on random subsampling. However, significant performance advances were demonstrated by using principled sampling strategies for reduced-order models (ROMs) [75] in fluid dynamics [754] and ocean modeling [767]. More recently, variants of the

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

so-called *empirical interpolation method* (EIM, DEIM and Q-DEIM) [55, 171, 215] have provided near-optimal sampling for interpolative reconstruction of nonlinear terms in ROMs.

Random Sensors. In general, randomly placed sensors may be used to estimate mode coefficients $\mathbf{a}$. However, when $p = r$ and the number of sensors is equal to the number of modes, the condition number is typically very large. In fact, the matrix $\mathbf{\Theta}$ is often numerically singular and the condition number is near 10^{16}. Oversampling, as in Section 1.8, rapidly improves the condition number, and even $p = r + 10$ usually has much better reconstruction performance.

QR Pivoting for Sparse Sensors. The greedy matrix QR factorization with column pivoting of $\mathbf{\Psi}_r^T$, explored by Drmac and Gugercin [215] for reduced-order modeling, provides a particularly simple and effective sensor optimization. The QR pivoting method is fast, simple to implement, and provides nearly optimal sensors tailored to a specific SVD/POD basis. QR factorization is optimized for most scientific computing libraries, including MATLAB, LAPACK, and NumPy. In addition, QR can be sped-up by ending the procedure after the first p pivots are obtained.

The reduced matrix QR factorization with column pivoting decomposes a matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ into a unitary matrix $\mathbf{Q}$, an upper triangular matrix $\mathbf{R}$ and a column permutation matrix $\mathbf{C}^T$ such that $\mathbf{A}\mathbf{C}^T = \mathbf{Q}\mathbf{R}$. The pivoting procedure provides an approximate greedy solution method to minimize the matrix volume, which is the absolute value of the determinant. QR column pivoting increments the volume of the submatrix constructed from the pivoted columns by selecting a new pivot column with maximal 2-norm, then subtracting from every other column its orthogonal projection onto the pivot column.

Thus QR factorization with column pivoting yields r point sensors (pivots) that best sample the r basis modes $\mathbf{\Psi}_r$:

$$\mathbf{\Psi}_r^T \mathbf{C}^T = \mathbf{Q}\mathbf{R}. \tag{3.22}$$

Sensor selection based on the pivoted QR algorithm is quite simple for $p = r$. In MATLAB, the code is

```
[Q,R,pivot] = qr(Psi_r','vector'); % QR sensor selection
C = zeros(p,n);
for j=1:p
    C(j,pivot(j))=1;
end
```

In Python the code is

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

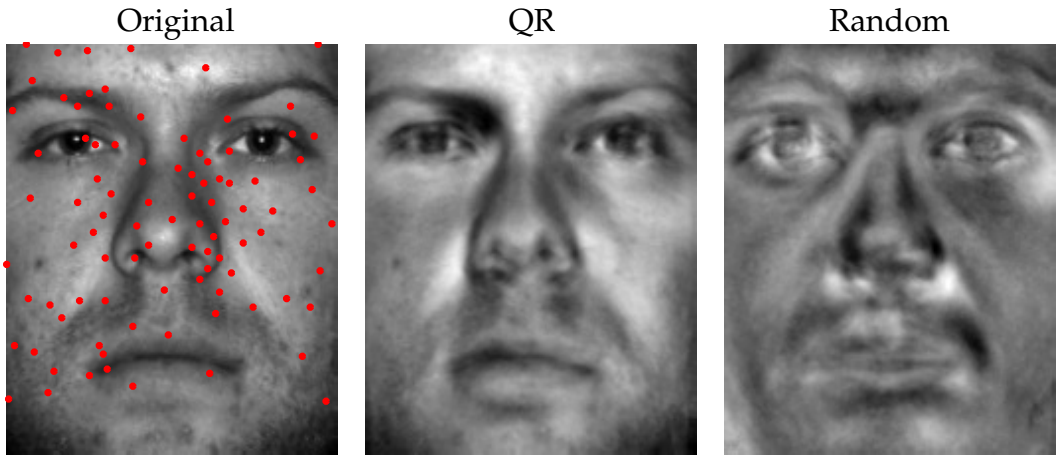


Figure 3.22: (left) Original image and locations of $p = 100$ QR sensors in a $r = 100$ mode library. (middle) Reconstruction with QR sensors. (right) Reconstruction with random sensors.

```
from scipy import linalg
Q,R,pivot = linalg.qr(Psi_r.T,pivoting=True)
C = np.zeros_like(Psi_r.T)
C[pivot[:r]] = 1
for k in range(r):
    C[k,pivot[k]] = 1
```

It may also be advantageous to use oversampling [557], choosing more sensors than modes, so $p > r$. In this case, there are several strategies, and random oversampling is a robust choice.

Example: Reconstructing a Face with Sparse Sensors

To demonstrate the concept of signal reconstruction in a tailored basis, we will design optimized sparse sensors in the library of eigenfaces from Section 1.6. Figure 3.22 shows the QR sensor placement and reconstruction, along with the reconstruction using random sensors. We use $p = 100$ sensors in a $r = 100$ mode library. This code assumes that the faces have been loaded and the singular vectors are in a matrix $\mathbf{U}$. Optimized QR sensors result in a more accurate reconstruction, with about three times less reconstruction error. In addition, the condition number is orders of magnitude smaller than with random sensors. Both QR and random sensors may be improved by oversampling. From the QR sensors $\mathbf{C}$ based on $\mathbf{\Psi}_r$, it is possible to reconstruct an approximate image from these sensors. In MATLAB, the reconstruction is given by

```
Theta = C*Psi_r;
y = faces(pivot(1:p),1); % Measure at pivot locations
a = Theta\y;             % Estimate coefficients
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

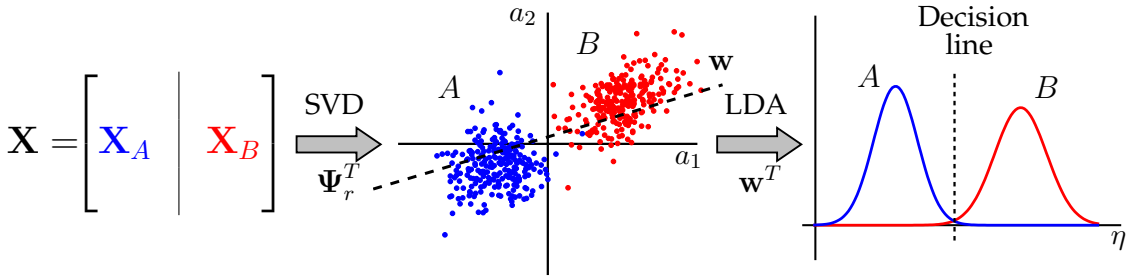


Figure 3.23: Schematic illustrating SVD for feature extraction, followed by linear discriminant analysis (LDA) for the automatic classification of data into two categories A and B. Reproduced with permission from Bai et al. [40].

```
faceRecon = Psi_r * a;    % Reconstruct face
```

In Python, the reconstruction is given by

```
Theta = np.dot(C , Psi_r)
y = faces[pivot[:r]]              # Measure at pivot locations
a = np.dot(np.linalg.pinv(Theta),y) # Estimate coefficients
faceRecon = np.dot(Psi_r , a)       # Reconstruct face
```

Sparse Classification

For image classification, even fewer sensors may be required than for reconstruction. For example, sparse sensors may be selected that contain the most discriminating information to characterize two categories of data [122]. Given a library of r SVD modes $\boldsymbol{\Psi}_r$, it is often possible to identify a vector $\mathbf{w} \in \mathbb{R}^r$ in this subspace that maximally distinguishes between two categories of data, as described in Section 5.6 and shown in Fig. 3.23. Sparse sensors $\mathbf{s}$ that map into this discriminating direction, projecting out all other information, are found by

$$\mathbf{s} = \underset{\mathbf{s}'}{\operatorname{argmin}} \|\mathbf{s}'\|_1 \quad \text{subject to} \quad \boldsymbol{\Psi}_r^T \mathbf{s}' = \mathbf{w}. \tag{3.23}$$

This sparse sensor placement optimization for classification (SSPOC) is shown in Fig. 3.24 for an example classifying dogs versus cats. The library $\boldsymbol{\Psi}_r$ contains the first r *eigen-pets* and the vector $\mathbf{w}$ identifies the key differences between dogs and cats. Note that this vector does not care about the degrees of freedom that characterize the various features within the dog or cat clusters, but rather only the differences between the two categories. Optimized sensors are aligned with regions of interest, such as the eyes, nose, mouth, and ears.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

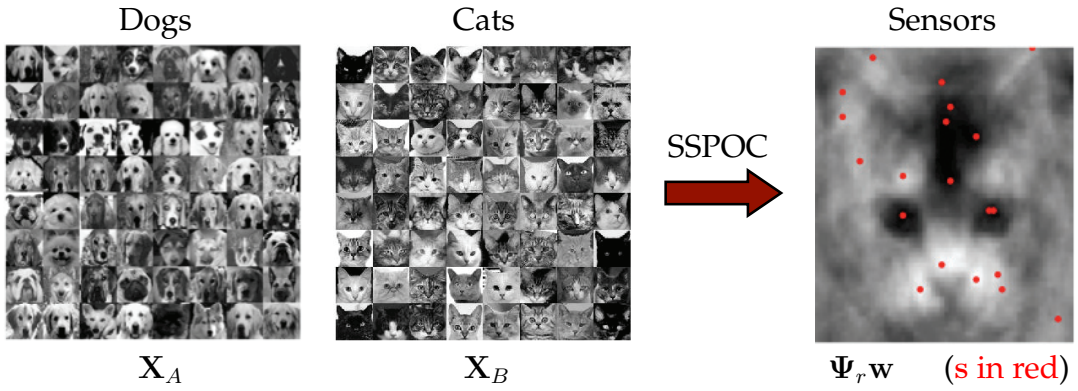


Figure 3.24: Sparse sensor placement optimization for classification (SSPOC) illustrated for optimizing sensors to classify dogs and cats. Reproduced with permission from B. Brunton et al. [122].

Suggested Reading

Papers and reviews

(1) Regression shrinkage and selection via the lasso, by R. Tibshirani, *Journal of the Royal Statistical Society B*, 1996 [702].

(2) Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information, by E. J. Candès, J. Romberg, and T. Tao, *IEEE Transactions on Automatic Control*, 2006 [156].

(3) Compressed sensing, by D. L. Donoho, *IEEE Transactions on Information Theory*, 2006 [204].

(4) Compressive sensing, by R. G. Baraniuk, *IEEE Signal Processing Magazine*, 2007 [53].

(5) Robust face recognition via sparse representation, by J. Wright, A. Yang, A. Ganesh, S. Sastry, and Y. Ma, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2009 [762].

(6) Robust principal component analysis?, by E. J. Candès, X. Li, Y. Ma, and J. Wright, *Journal of the ACM*, 2011 [155].

(7) Signal recovery from random measurements via orthogonal matching pursuit, by J. A. Tropp and A. C. Gilbert, *IEEE Transactions on Information Theory*, 2007 [717].

(8) Data-driven sparse sensor placement, by K. Manohar, B. W. Brunton, J. N. Kutz, and S. L. Brunton, *IEEE Control Systems Magazine*, 2018 [481].

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Homework

Exercise 3-1. Load the image dog.jpg and convert to grayscale. We will repeat Exercise 2-1, using the FFT to compress the image at different compression ratios. However, now, we will compare the error versus compression ratio for the image downsampled at different resolutions. Compare the original image (2000×1500) and downsampled copies of the following resolutions: 1000×750, 500×375, 200×150, and 100×75. Plot the error versus compression ratio for each image resolution on the same plot. Explain the observed trends.

Exercise 3-2. This example will explore geometry and sampling probabilities in high-dimensional spaces. Consider a two-dimensional square dart board with length $L = 2$ on both sides and a circle of radius $R = 1$ in the middle. Write a program to throw $10\,000$ darts by generating a uniform random x and y position on the square. Compute the radius for each point and compute what fraction land inside the circle (i.e., how many have radius < 1). Is this consistent with your expectation based on the area of the circle and the square?

Repeat this experiment, throwing $10\,000$ darts randomly (sampled from a uniform distribution) on an N-dimensional cube (length $L = 2$) with an N-dimensional sphere inside (radius $R = 1$), for $N = 2$ through $N = 10$. For a given N, what fraction of the points land inside the sphere. Plot this fraction versus N. Also compute the histogram of the radii of the randomly sampled points for each N and plot these. What trends do you notice in the data?

Exercise 3-3. This exercise will explore the relationship between the sparsity K, the signal size n, and the number of samples p in compressed sensing.

(a) For $n = 1000$ and $K = 5$, create a K-sparse vector $\mathbf{s}$ of Fourier coefficients in a Fourier basis $\boldsymbol{\Psi}$. For each p from 1 to 100, create a Gaussian random sampling matrix $\mathbf{C} \in \mathbb{R}^{p \times n}$ to create a measurement vector $\mathbf{y} = \mathbf{C}\boldsymbol{\Psi}\mathbf{s}$. Use compressed sensing based on this measurement to estimate $\hat{\mathbf{s}}$. For each p, repeat this with at least 10 realizations of the random measurement matrix $\mathbf{C}$. Plot the average relative error of $\|\hat{\mathbf{s}} - \mathbf{s}\|_2 / \|\mathbf{s}\|$ versus p; it may be helpful to visualize the errors with a box-and-whisker plot. Explain the trends. Also plot the average ℓ_1 and ℓ_0 error versus p.

(b) Repeat the above experiment for $K = 1$ through $K = 20$. What changes?

(c) Now repeat the above experiment for $K = 5$, varying the signal size using $n = 100$, $n = 500$, $n = 1000$, $n = 2000$, and $n = 5000$.

Exercise 3-4. Repeat the above exercise with a uniformly sampled random sample matrix. Also repeat with a Bernoulli random matrix and a matrix that comprises random single pixels. Plot the average relative errors for these different

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

sampling matrices on the same plot (including the plot for Gaussian random sampling). Discuss the trends.

Exercise 3-5. Generate a DFT matrix $\mathbf{\Psi}$ for $n = 512$. We will use this as a basis for compressed sensing, and we will compute the incoherence of this basis and different measurement matrices. For $p = 16$, create different random measurement matrices $\mathbf{C}$ given by Gaussian random measurements, Bernoulli random measurements, and random single-pixel measurements. For each matrix, normalize the length of each row to 1. Now, for each measurement matrix type, compute the incoherence $\mu(\mathbf{C}, \mathbf{\Psi})$. Repeat this for many random instances of each $\mathbf{C}$ matrix type and compare the histogram of incoherence values for each matrix type. Further, compare the histogram of each inner product $\sqrt{n}\langle \mathbf{c}_k, \boldsymbol{\psi}_j \rangle$ for each matrix type. Discuss any trends and the implications for compressed sensing with these measurement matrices. Are there other factors that are relevant for the sensing matrix?

Exercise 3-6. This exercise will explore sparse representation from Section 3.6 to estimate a fluid flow field, following Callaham et al. [147]. Load the cylinder flow data set. Coarsen each flow field by a factor of 20 in each direction using `imresize`, and build a library of these coarsened measurements (i.e., a matrix, where each column contains these downsampled measurements). Plot a movie of the flow field in these new coordinates. Now, pick a column of the full flow field matrix and add Gaussian random noise to this field. Downsample the noisy field by a factor of 20 and use SRC to find the closest downsampled library element. Then use this column of the full flow field library as your reconstructed estimate.

Try this approach with different levels of noise added to the original flow field. See how much noise is required before the method breaks. Try different approaches to creating a low-dimensional representation of the image (i.e., instead of downsampling, you can measure the flow field in a small 5×10 patch and use this as the low-dimensional feature for SRC).

Exercise 3-7. This exercise will explore RPCA from Section 3.7 for robust flow field analysis, following Scherl et al. [631].

(a) Load the cylinder flow data set. Compute the SVD as in Exercise 1-7 and plot the movie of the flow. Also plot the singular values and first 10 singular vectors.

(b) Now, contaminate a random 10% of the entries of the data matrix with salt-and-pepper noise. The contaminated points should not be the same for each column, and the salt-and-pepper noise should be $\pm 5\eta$, where η is the standard deviation of the entire data set. Compute the SVD of

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

the contaminated matrix and plot the movie of the flow along with the singular values and first 10 singular vectors.

(c) Clean the contaminated data set by applying RPCA and keeping the low-rank portion $\mathbf{L}$. Again, compute the SVD of the decontaminated matrix $\mathbf{L}$ and plot the movie of the flow along with the singular values and first 10 singular vectors. Compare these with the results from the original clean and contaminated data sets.

(d) Try to clean the data by applying the Gavish–Donoho threshold to the data matrix contaminated with salt-and-pepper noise. Does this work? Explain why or why not.

Exercise 3-8. This exercise will explore the sparse sensor selection approach based on QR from Section 3.8.

(a) Load the Yale B faces data set. Randomly choose one person to omit from the data matrix and compute the SVD of the remaining data. Compute the QR sensor locations for $p = 100$ using the first $r = 100$ modes of this SVD basis $\tilde{\mathbf{U}}$. Use these sensor locations to reconstruct the images of the person that was left out of the matrix for the SVD. Compare the reconstruction error using these QR sensor locations with reconstruction using $p = 100$ randomly chosen sensors, as in Fig. 3.22.

(b) Now, repeat this experiment 36 times, each time omitting a different person from the data before computing the SVD, and use the sensor locations to reconstruct the images of the omitted person. This will provide enough reconstruction errors on which to perform statistics. For each experiment, also compute the reconstruction error using 36 different configurations of $p = 100$ random sensors. Plot the histograms of the error for the QR and random sensors, and discuss.

(c) Finally, repeat the above experiments for different sensor number $p = 10$ through $p = 200$ in increments of 10. Plot the error distributions versus p for QR and random sensor configurations. Because each value of p corresponds to many reconstruction errors, it would be best to plot this as a box-and-whisker plot or as a violin plot.

Exercise 3-9. In the exercise above, for $p = 100$, compare the reconstruction results using the $p = 100$ QR sensors to reconstruct in the first $r = 100$ modes, versus using the same sensors to reconstruct in the first $r = 90$ SVD modes. Is one more accurate than the other? Compare the condition number of the 100×100 and 100×90 matrices obtained by sampling the p rows of the $r = 100$ and $r = 90$ columns of $\tilde{\mathbf{U}}$ from the SVD.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Part II

Machine Learning and Data Analysis

Chapter 4

Regression and Model Selection

All of machine learning revolves around optimization. This includes regression and model selection frameworks that aim to provide parsimonious and interpretable models for data [350]. Curve fitting is the most basic of regression techniques, with polynomial and exponential fitting resulting in solutions that come from solving the linear system

$$\mathbf{A}\mathbf{x} = \mathbf{b}. \tag{4.1}$$

When the model is not prescribed, then optimization methods are used to select the best model. This changes the underlying mathematics for function fitting to either an over-determined or an under-determined optimization problem for linear systems given by

$$\underset{\mathbf{x}}{\operatorname{argmin}}(\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2 + \lambda g(\mathbf{x})) \quad \text{or} \tag{4.2a}$$

$$\underset{\mathbf{x}}{\operatorname{argmin}}\, g(\mathbf{x}) \quad \text{subject to} \quad \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2 \leq \epsilon, \tag{4.2b}$$

where $g(\mathbf{x})$ is a given penalization (with penalty parameter λ for over-determined systems).

For over- and under-determined linear systems of equations, which result in either no solutions or an infinite number of solutions of (4.1), a choice of constraint or penalty, which is also known as *regularization*, must be made in order to produce a solution. For instance, one can enforce a solution minimizing the smallest ℓ_2-norm in an under-determined system so that $\min g(\mathbf{x}) = \min \|\mathbf{x}\|_2$. More generally, when considering regression to nonlinear models, then the overall mathematical framework takes the more general form

$$\underset{\mathbf{x}}{\operatorname{argmin}}(f(\mathbf{A}, \mathbf{x}, \mathbf{b}) + \lambda g(\mathbf{x})) \quad \text{or} \tag{4.3a}$$

$$\underset{\mathbf{x}}{\operatorname{argmin}}\, g(\mathbf{x}) \quad \text{subject to} \quad f(\mathbf{A}, \mathbf{x}, \mathbf{b}) \leq \epsilon, \tag{4.3b}$$

which are often solved using gradient descent algorithms. Indeed, this general framework is also at the center of deep learning algorithms.

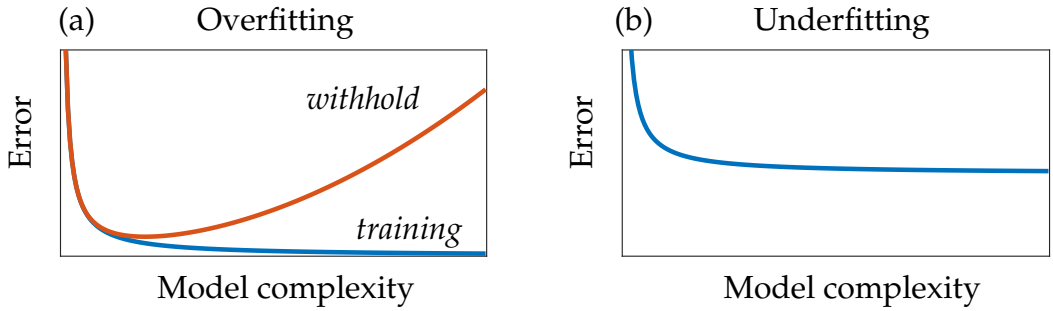


Figure 4.1: Prototypical behavior of over- and underfitting of data. (a) For overfitting, increasing the model complexity or training epochs (iterations) leads to improved reduction of error on training data while increasing the error on the withheld data. (b) For underfitting, the error performance is limited due to restrictions on model complexity. These canonical graphs are ubiquitous in data science and of paramount importance when evaluating a model.

In addition to optimization strategies, a central concern in data science is understanding if a proposed model has overfit or underfit the data. Thus *cross-validation* strategies are critical for evaluating any proposed model. Cross-validation will be discussed in detail in what follows, but the main concepts can be understood from Fig. 4.1. A given data set must be partitioned into training, validation, and withhold sets. A model is constructed from the training and validation data and finally tested on the withhold set. For overfitting, increasing the model complexity or training epochs (iterations) improves the error on the training set while leading to increased error on the withhold set. Figure 4.1(a) shows the canonical behavior of data overfitting, suggesting that the model complexity and/or training epochs be limited in order to avoid the overfitting. In contrast, underfitting limits the ability to achieve a good model as shown in Fig. 4.1(b). However, it is not always clear if you are underfitting or if the model can be improved. Cross-validation is of such paramount importance that it is automatically included in most machine learning algorithms in MATLAB. Importantly, the following mantra holds: *If you do not cross-validate, you is dumb.*

The next few chapters will outline how optimization and cross-validation arise in practice, and will highlight the choices that need to be made in applying meaningful constraints and structure to $g(\mathbf{x})$ so as to achieve interpretable solutions. Indeed, the objective (loss) function $f(\cdot)$ and regularization $g(\cdot)$ are equally important in determining computationally tractable optimization strategies. Often times, proxy loss and regularization functions are chosen in order to achieve approximations to the true objective of the optimization. Such choices depend strongly upon the application area and data under consideration.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

4.1 Classic Curve Fitting

Curve fitting is one of the most basic and foundational tools in data science. From our earliest educational experiences in the engineering and physical sciences, least-squares polynomial fitting was advocated for understanding the dominant trends in real data. Adrien-Marie Legendre used least-squares as early as 1805 to fit astronomical data [436], with Gauss more fully developing the theory of least-squares as an optimization problem in a seminal contribution of 1821 [264]. Curve fitting in such astronomical applications was highly effective given the simple elliptical orbits (quadratic polynomial functions) manifest by planets and comets. Thus one can argue that data science has long been a cornerstone of our scientific efforts. Indeed, it was through Kepler's access to Tycho Brahe's state-of-the-art astronomical data that he was able, after 11 years of research, to produce the foundations for the laws of planetary motion, positing the elliptical nature of planetary orbits, which were clearly best-fit solutions to the available data [380].

A broader mathematical viewpoint of curve fitting, which we will advocate throughout this text, is *regression*. Like curve fitting, regression attempts to estimate the relationship among variables using a variety of statistical tools. Specifically, one can consider the general relationship between independent variables $\mathbf{X}$, dependent variables $\mathbf{Y}$, and some unknown parameters $\boldsymbol{\beta}$:

$$\mathbf{Y} = f(\mathbf{X}, \boldsymbol{\beta}), \tag{4.4}$$

where the regression function $f(\cdot)$ is typically prescribed and the parameters $\boldsymbol{\beta}$ are found by optimizing the *goodness-of-fit* of this function to data. In what follows, we will consider curve fitting as a special case of regression. Importantly, regression and curve fitting discover relationships among variables by optimization. Broadly speaking, machine learning is framed around regression techniques, which are themselves framed around optimization based on data. Thus, at its absolute mathematical core, machine learning and data science revolve around positing an optimization problem. Of course, the success of optimization itself depends critically on defining an *objective function* to be optimized.

Least-Squares Fitting Methods

To illustrate the concepts of regression, we will consider classic least-squares polynomial fitting for characterizing trends in data. The concept is straightforward and simple: use a simple function to describe a trend by minimizing the sum-square error between the selected function $f(\cdot)$ and its fit to the data. As we show here, classical curve fitting is formulated as a simple solution of $\mathbf{Ax} = \mathbf{b}$.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Consider a set of n data points:

$$(x_1, y_1),\ (x_2, y_2),\ (x_3, y_3),\ \ldots,\ (x_n, y_n). \tag{4.5}$$

Further, assume that we would like to find a best-fit line through these points. We can approximate the line by the function

$$f(x) = \beta_1 x + \beta_2, \tag{4.6}$$

where the constants β_1 and β_2, which are the parameters of the vector $\boldsymbol{\beta}$ in (4.4), are chosen to minimize some error associated with the fit. The line fit gives the *linear regression* model $\mathbf{Y} = f(\mathbf{A}, \boldsymbol{\beta}) = \beta_1 \mathbf{X} + \beta_2$. Thus the function gives a linear model which approximates the data, with the approximation error at each point given by

$$f(x_k) = y_k + E_k, \tag{4.7}$$

where y_k is the true value of the data and E_k is the error of the fit from this value.

Various error metrics can be minimized when approximating with a given function $f(x)$. The choice of error metric, or norm, used to compute a goodness-of-fit will be critical in this chapter. Three standard possibilities are often considered, which are associated with the ℓ_2- (least-squares), ℓ_1-, and ℓ_∞-norms. These are defined as follows:

$$E_\infty(f) = \max_{1<k<n} |f(x_k) - y_k| \qquad \text{maximum error } (\ell_\infty), \tag{4.8a}$$

$$E_1(f) = \frac{1}{n}\sum_{k=1}^{n} |f(x_k) - y_k| \qquad \text{mean absolute error } (\ell_1), \tag{4.8b}$$

$$E_2(f) = \left(\frac{1}{n}\sum_{k=1}^{n} |f(x_k) - y_k|^2\right)^{1/2} \qquad \text{least-squares error } (\ell_2). \tag{4.8c}$$

Such regression error metrics have been previously considered in Chapter 1, but they will be considered once again here in the framework of model selection.

In addition to the above norms, one can more broadly consider the error based on the ℓ_p-norm:

$$E_p(f) = \left(\frac{1}{n}\sum_{k=1}^{n} |f(x_k) - y_k|^p\right)^{1/p}. \tag{4.9}$$

For different values of p, the best-fit line will be different. In most cases, the differences are small. However, when there are outliers in the data, the choice of norm can have a significant impact.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

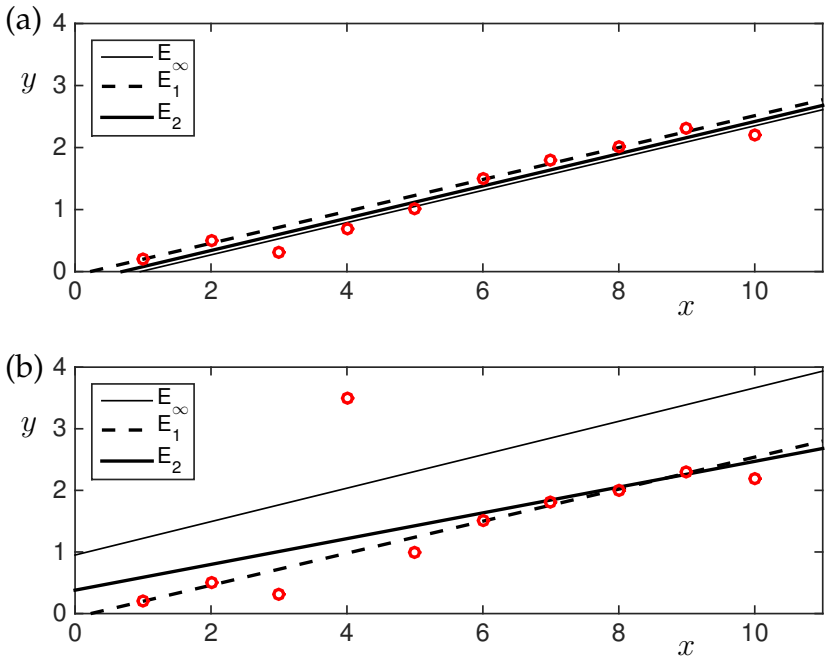


Figure 4.2: Line fits for the three different error metrics E_∞, E_1, and E_2. In (a), the data has no outliers and the three linear models, although different, produce approximately the same model. With outliers, (b) shows that the predictions are significantly different.

When fitting a curve to a set of data, the root-mean-square error (4.8c) is often chosen to be minimized. This is called a *least-squares fit*. Figure 4.2 depicts three line fits that minimize the errors E_∞, E_1, and E_2 listed above. The E_∞ error line fit is strongly influenced by the one data point that does not fit the trend. The E_1 and E_2 lines fit nicely through the bulk of the data, although their slopes are quite different in comparison to when the data has no outliers. The linear models for these three error metrics are constructed using MATLAB's **fminsearch** command. The code for all three is given as follows.

Code 4.1: [MATLAB] Regression for linear fit.

```
p1=fminsearch('fit1',[1 1],[],x,y);
p2=fminsearch('fit2',[1 1],[],x,y);
p3=fminsearch('fit3',[1 1],[],x,y);

xf=0:0.1:11
y1=polyval(p1,xf); y2=polyval(p2,xf); y3=polyval(p3,xf);
```

Code 4.1: [Python] Regression for linear fit.

```
p1 = scipy.optimize.fmin(fit1,x0,args=(t,));
p2 = scipy.optimize.fmin(fit2,x0,args=(t,));
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

```
p3 = scipy.optimize.fmin(fit3,x0,args=(t,));

xf = np.arange(0,11,0.1)
y1 = np.polyval(p1,xf); y2 = np.polyval(p2,xf); y3 = np.
   polyval(p3,xf)
```

For each error metric, the computation of the error metrics (4.8) must be computed. The **fminsearch** command requires that the objective function for minimization be given. For the three error metrics considered, this results in the following set of functions for **fminsearch**.

Code 4.2: [MATLAB] Maximum error ℓ_∞.

```
function E=fit1(x0,x,y)
E=max(abs( x0(1)*x+x0(2)-y ));
```

Code 4.2: [MATLAB] Sum of absolute error ℓ_1.

```
function E=fit2(x0,x,y)
E=sum(abs( x0(1)*x+x0(2)-y ));
```

Code 4.2: [MATLAB] Least-squares error ℓ_2.

```
function E=fit3(x0,x,y)
E=sum(abs( x0(1)*x+x0(2)-y ).^2 );
```

Code 4.2: [Python] Fitting errors.

```
def fit1(x0,t):
    x,y=t
    return np.max(np.abs(x0[0]*x + x0[1]-y))
def fit2(x0,t):
    x,y=t
    return np.sum(np.abs(x0[0]*x + x0[1]-y))
def fit3(x0,t):
    x,y=t
    return np.sum(np.power(np.abs(x0[0]*x + x0[1]-y),2))
```

Finally, for the outlier data, an additional point is added to the data in order to help illustrate the influence of the error metrics on producing a linear regression model.

Least-Squares Line

Least-squares fitting to linear models has critical advantages over other norms and metrics. Specifically, the optimization is inexpensive, since the error can be computed analytically. To show this explicitly, consider applying the least-squares fit criteria to the data points (x_k, y_k), where $k = 1, 2, 3, \ldots, n$. To fit the

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

curve

$$f(x) = \beta_1 x + \beta_2 \tag{4.10}$$

to this data, the error E_2 is found by minimizing the sum:

$$E_2(f) = \sum_{k=1}^{n} |f(x_k) - y_k|^2 = \sum_{k=1}^{n} (\beta_1 x_k + \beta_2 - y_k)^2. \tag{4.11}$$

Minimizing this sum requires differentiation. Specifically, the constants β_1 and β_2 are chosen so that a minimum occurs. Thus we require $\partial E_2/\partial \beta_1 = 0$ and $\partial E_2/\partial \beta_2 = 0$. Note that although a zero derivative can indicate either a minimum or a maximum, we know this must be a minimum of the error since there is no maximum error, i.e., we can always choose a line that has a larger error. The minimization condition gives

$$\frac{\partial E_2}{\partial \beta_1} = 0: \quad \sum_{k=1}^{n} 2(\beta_1 x_k + \beta_2 - y_k) x_k = 0, \tag{4.12a}$$

$$\frac{\partial E_2}{\partial \beta_2} = 0: \quad \sum_{k=1}^{n} 2(\beta_1 x_k + \beta_2 - y_k) = 0. \tag{4.12b}$$

Upon rearranging, a 2×2 system of linear equations is found for A and B:

$$\begin{pmatrix} \sum_{k=1}^{n} x_k^2 & \sum_{k=1}^{n} x_k \\ \sum_{k=1}^{n} x_k & n \end{pmatrix} \begin{pmatrix} \beta_1 \\ \beta_2 \end{pmatrix} = \begin{pmatrix} \sum_{k=1}^{n} x_k y_k \\ \sum_{k=1}^{n} y_k \end{pmatrix} \quad \Longrightarrow \quad \mathbf{A}\mathbf{x} = \mathbf{b}. \tag{4.13}$$

This linear system of equations can be solved using the backslash command in MATLAB. Thus an optimization procedure is unnecessary since the solution is computed exactly from a 2×2 matrix.

This method can be easily generalized to higher polynomial fits. In particular, a parabolic fit to a set of data requires the fitting function

$$f(x) = \beta_1 x^2 + \beta_2 x + \beta_3, \tag{4.14}$$

where now the three constants β_1, β_2, and β_3 must be found. These can be solved for with the 3×3 system resulting from minimizing the error $E_2(\beta_1, \beta_2, \beta_3)$ by taking

$$\frac{\partial E_2}{\partial \beta_1} = 0, \tag{4.15a}$$

$$\frac{\partial E_2}{\partial \beta_2} = 0, \tag{4.15b}$$

$$\frac{\partial E_2}{\partial \beta_3} = 0. \tag{4.15c}$$

In fact, any polynomial fit of degree k will yield a $(k+1) \times (k+1)$ linear system of equations $\mathbf{A}\mathbf{x} = \mathbf{b}$ whose solution can be found.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Data Linearization

Although a powerful method, the minimization procedure for general fitting of arbitrary functions results in equations which are non-trivial to solve. Specifically, consider fitting data to the exponential function

$$f(x) = \beta_2 \exp(\beta_1 x). \tag{4.16}$$

The error to be minimized is

$$E_2(\beta_1, \beta_2) = \sum_{k=1}^{n} (\beta_2 \exp(\beta_1 x_k) - y_k)^2. \tag{4.17}$$

Applying the minimizing conditions leads to

$$\frac{\partial E_2}{\partial \beta_1} = 0: \quad \sum_{k=1}^{n} 2(\beta_2 \exp(\beta_1 x_k) - y_k)\beta_2 x_k \exp(\beta_1 x_k) = 0, \tag{4.18a}$$

$$\frac{\partial E_2}{\partial \beta_2} = 0: \quad \sum_{k=1}^{n} 2(\beta_2 \exp(\beta_1 x_k) - y_k) \exp(\beta_1 x_k) = 0. \tag{4.18b}$$

This in turn leads to the 2×2 system

$$\beta_2 \sum_{k=1}^{n} x_k \exp(2\beta_1 x_k) - \sum_{k=1}^{n} x_k y_k \exp(\beta_1 x_k) = 0, \tag{4.19a}$$

$$\beta_2 \sum_{k=1}^{n} \exp(2\beta_1 x_k) - \sum_{k=1}^{n} y_k \exp(\beta_1 x_k) = 0. \tag{4.19b}$$

This system of equations is nonlinear and cannot be solved in a straightforward fashion. Indeed, a solution may not even exist. Or many solutions may exist. Section 4.2 describes a possible iterative procedure, called gradient descent, for solving this nonlinear system of equations.

To avoid the difficulty of solving this nonlinear system, the exponential fit can be *linearized* by the transformation

$$Y = \ln(y), \tag{4.20a}$$

$$X = x, \tag{4.20b}$$

$$\beta_3 = \ln \beta_2. \tag{4.20c}$$

Then the fit function

$$f(x) = y = \beta_2 \exp(\beta_1 x) \tag{4.21}$$

can be linearized by taking the natural log of both sides so that

$$\begin{aligned} \ln y = \ln(\beta_2 \exp(\beta_1 x)) = \ln \beta_2 + \ln(\exp(\beta_1 x)) = \beta_3 + \beta_1 x \\ \implies \quad Y = \beta_1 X + \beta_3. \end{aligned} \tag{4.22}$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

By fitting to the natural log of the y data,

$$(x_i, y_i) \to (x_i, \ln y_i) = (X_i, Y_i), \tag{4.23}$$

the curve fit for the exponential function becomes a linear fitting problem, which is easily handled. Thus, if a transform exists that linearizes the data, then standard polynomial fitting methods can be used to solve the resulting linear system $\mathbf{A}\mathbf{x} = \mathbf{b}$.

4.2 Nonlinear Regression and Gradient Descent

Polynomial and exponential curve fitting admit analytically tractable, best-fit least-squares solutions. However, such curve fits are highly specialized and a more general mathematical framework is necessary for solving a broader set of problems. For instance, one may wish to fit a nonlinear function of the form $f(x) = \beta_1 \cos(\beta_2 x + \beta_3) + \beta_4$ to a data set. Instead of solving a linear system of equations, general nonlinear curve fitting leads to a system of nonlinear equations. The general theory of nonlinear regression assumes that the fitting function takes the general form

$$f(x) = f(x, \boldsymbol{\beta}), \tag{4.24}$$

where the $m < n$ fitting coefficients $\boldsymbol{\beta} \in \mathbb{R}^m$ are used to minimize the error. The root-mean-square error is then defined as

$$E_2(\boldsymbol{\beta}) = \sum_{k=1}^{n} (f(x_k, \boldsymbol{\beta}) - y_k)^2, \tag{4.25}$$

which can be minimized by considering the $m \times m$ system generated from minimizing with respect to each parameter β_j:

$$\frac{\partial E_2}{\partial \beta_j} = 0 \quad \text{for } j = 1, 2, \ldots, m. \tag{4.26}$$

In general, this gives the *nonlinear* set of equations

$$\sum_{k=1}^{n} (f(x_k, \boldsymbol{\beta}) - y_k) \frac{\partial f}{\partial \beta_j} = 0 \quad \text{for } j = 1, 2, 3, \ldots, m. \tag{4.27}$$

There are no general methods available for solving such nonlinear systems. Indeed, nonlinear systems can have no solutions, several solutions, or even an infinite number of solutions. Most attempts at solving nonlinear systems are based on iterative schemes which require a good initial guess to converge to the global minimum error. Regardless, the general fitting procedure is straightforward and allows for the construction of a best-fit curve to match the data.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

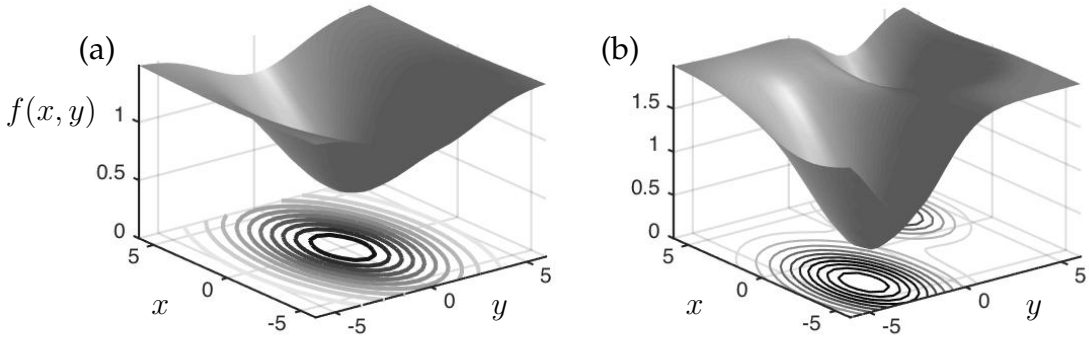


Figure 4.3: Two objective function landscapes representing (a) a convex function and (b) a non-convex function. Convex functions have many guarantees of convergence, while non-convex functions have a variety of pitfalls that can limit the success of gradient descent. For non-convex functions, local minima and an inability to compute gradient directions (derivatives that are near zero) make it challenging for optimization.

In such a solution procedure, it is imperative that a reasonable initial guess be provided by the user. Otherwise, rapid convergence to the desired root may not be achieved.

Figure 4.3 shows two example functions to be minimized. The first is a convex function (Fig. 4.3(a)). Convex functions are ideal in that guarantees of convergence exist for many algorithms, and gradient descent can be tuned to perform exceptionally well for such functions. The second illustrates a non-convex function (Fig. 4.3(b)) and shows many of the typical problems associated with gradient descent, including the fact that the function has multiple local minima as well as flat regions where gradients are difficult to actually compute, i.e., the gradient is near zero. Optimizing this non-convex function requires a good guess for the initial conditions of the gradient descent algorithm, although there are many advances around gradient descent for restarting and ensuring that one is not stuck in a local minimum. Recent training algorithms for deep neural networks have greatly advanced gradient descent innovations. This will be further considered in Chapter 6 on neural networks.

Gradient Descent

For high-dimensional systems, we generalize the concept of a minimum or maximum, i.e., an extremum of a multi-dimensional function $f(\mathbf{x})$. At an extremum, the gradient must be zero, so that

$$\nabla f(\mathbf{x}) = \mathbf{0}. \tag{4.28}$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Since saddles exist in higher-dimensional spaces, one must test if the extremum point is a minimum or a maximum. The idea behind gradient descent, or steepest descent, is to use the derivative information as the basis of an iterative algorithm that progressively converges to a local minimum point of $f(\mathbf{x})$.

To illustrate how to proceed in practice, consider the simple two-dimensional surface

$$f(x, y) = x^2 + 3y^2, \tag{4.29}$$

which has a single minimum located at the origin $(x, y) = 0$. The gradient for this function is

$$\nabla f(\mathbf{x}) = \frac{\partial f}{\partial x}\hat{\mathbf{x}} + \frac{\partial f}{\partial y}\hat{\mathbf{y}} = 2x\,\hat{\mathbf{x}} + 6y\,\hat{\mathbf{y}}, \tag{4.30}$$

where $\hat{\mathbf{x}}$ and $\hat{\mathbf{y}}$ are unit vectors in the x and y directions, respectively.

Figure 4.4 illustrates the gradient steepest descent algorithm. At the initial guess point, the gradient $\nabla f(\mathbf{x})$ is computed. This gives the direction of steepest descent towards the minimum point of $f(\mathbf{x})$, i.e., the minimum is located in the direction given by $-\nabla f(\mathbf{x})$. Note that the gradient does not point at the minimum, but rather gives the locally steepest path for minimizing $f(\mathbf{x})$. The geometry of the steepest descent suggests the construction of an algorithm whereby the next point in the iteration is picked by following the steepest descent, so that

$$\mathbf{x}_{k+1}(\delta) = \mathbf{x}_k - \delta\,\nabla f(\mathbf{x}_k), \tag{4.31}$$

where the parameter δ dictates how far to move along the gradient descent curve. This formula represents a generalization of a Newton method where the derivative is used to compute an update in the iteration scheme. In gradient descent, it is crucial to determine how much to step forward according to the computed gradient, so that the algorithm is always *going downhill* in an optimal way. This requires the determination of the correct value of δ in the algorithm.

To compute the value of δ, consider the construction of a new function

$$F(\delta) = f(\mathbf{x}_{k+1}(\delta)), \tag{4.32}$$

which must be minimized now as a function of δ. This is accomplished by computing $\partial F/\partial\delta = 0$. Thus one finds

$$\frac{\partial F}{\partial \delta} = -\nabla f(\mathbf{x}_{k+1})\nabla f(\mathbf{x}_k) = 0. \tag{4.33}$$

The geometrical interpretation of this result is the following: $\nabla f(\mathbf{x}_k)$ is the gradient direction of the current iteration point and $\nabla f(\mathbf{x}_{k+1})$ is the gradient direction of the future point; thus δ is chosen so that the two gradient directions are orthogonal.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

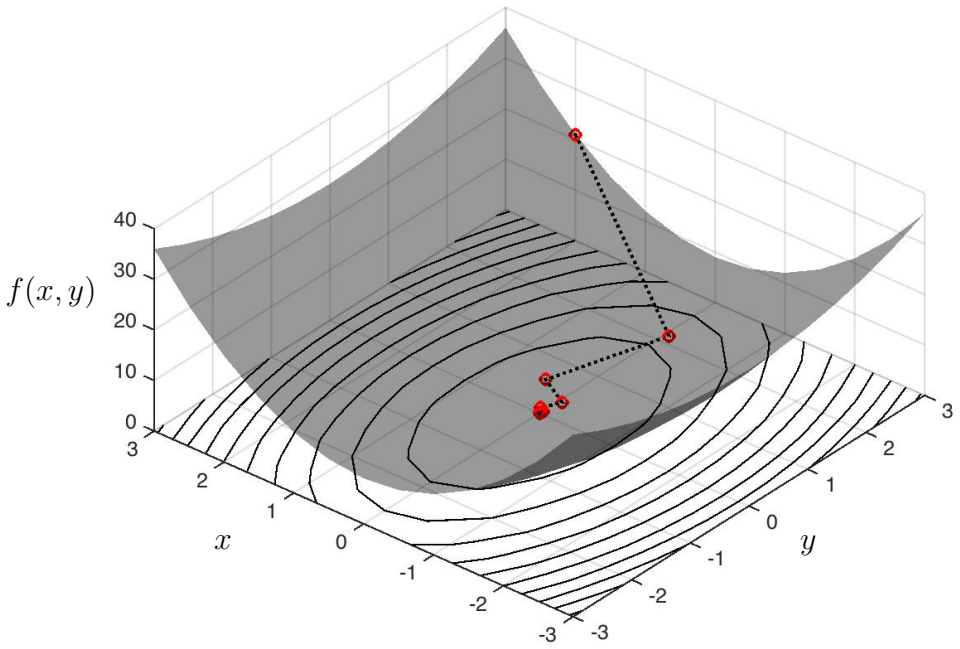


Figure 4.4: Gradient descent algorithm applied to the function $f(x,y) = x^2 + 3y^2$. The contours are plotted for each successive value (x,y) in the iteration algorithm given the initial guess $(x,y) = (3,2)$. Note the orthogonality of each successive gradient in the steepest descent algorithm.

For the example given above with $f(x,y) = x^2 + 3y^2$, we can compute this condition as follows:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \delta \nabla f(\mathbf{x}_k) = (1-2\delta)x\,\hat{\mathbf{x}} + (1-6\delta)y\,\hat{\mathbf{y}}. \tag{4.34}$$

This expression is used to compute

$$F(\delta) = f(\mathbf{x}_{k+1}(\delta)) = (1-2\delta)^2x^2 + 3(1-6\delta)^2y^2, \tag{4.35}$$

whereby its derivative with respect to δ gives

$$F'(\delta) = -4(1-2\delta)x^2 - 36(1-6\delta)y^2. \tag{4.36}$$

Setting $F'(\delta) = 0$ then gives

$$\delta = \frac{x^2 + 9y^2}{2x^2 + 54y^2} \tag{4.37}$$

as the optimal descent step length. Note that the length of δ is updated as the algorithm progresses. This gives us all the information necessary to perform the steepest descent search for the minimum of the given function.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Code 4.3: [MATLAB] Gradient descent example.

```
x(1)=3; y(1)=2; % initial guess
f(1)=x(1)^2+3*y(1)^2; % initial function value
for j=1:10
    del=(x(j)^2 +9*y(j)^2)/(2*x(j)^2 + 54*y(j)^2);
    x(j+1)=(1-2*del)*x(j); % update values
    y(j+1)=(1-6*del)*y(j);
    f(j+1)=x(j+1)^2+3*y(j+1)^2;

    if abs(f(j+1)-f(j))<10^(-6) % check convergence
        break
    end
end
```

Code 4.3: [Python] Gradient descent example.

```
x[0] = 3; y[0] = 2  # Initial guess

f[0] = x[0]**2 + 3*y[0]**2 # Initial function value

for j in range(len(x)-1):
    Del = (x[j]**2 + 9*y[j]**2)/(2*x[j]**2 + 54*y[j]**2)
    x[j+1] = (1 - 2*Del)*x[j] # update values
    y[j+1] = (1 - 6*Del)*y[j]
    f[j+1] = x[j+1]**2 + 3*y[j+1]**2

    if np.abs(f[j+1]-f[j]) < 10**(-6): # check convergence
        x = x[:j+2]
        y = y[:j+2]
        f = f[:j+2]
        break
```

As is clearly evident, this descent search algorithm based on derivative information is similar to Newton's method for root finding both in one dimension as well as in higher dimensions. Figure 4.4 shows the rapid convergence to the minimum for this convex function. Moreover, the gradient descent algorithm is the core algorithm of advanced iterative solvers such as the bi-conjugate gradient descent method (**bicgstab**) and the generalized method of residuals (**gmres**) [295].

In the example above, the gradient could be computed analytically. More generally, given just data itself, the gradient can be computed with numerical algorithms. The **gradient** command can be used to compute local or global gradients. Figure 4.5 shows the gradient terms $\partial f/\partial x$ and $\partial f/\partial y$ for the two functions shown in Fig. 4.3, where the function $f(x, y)$ is a two-dimensional function computed from a known function or directly from data. The output

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

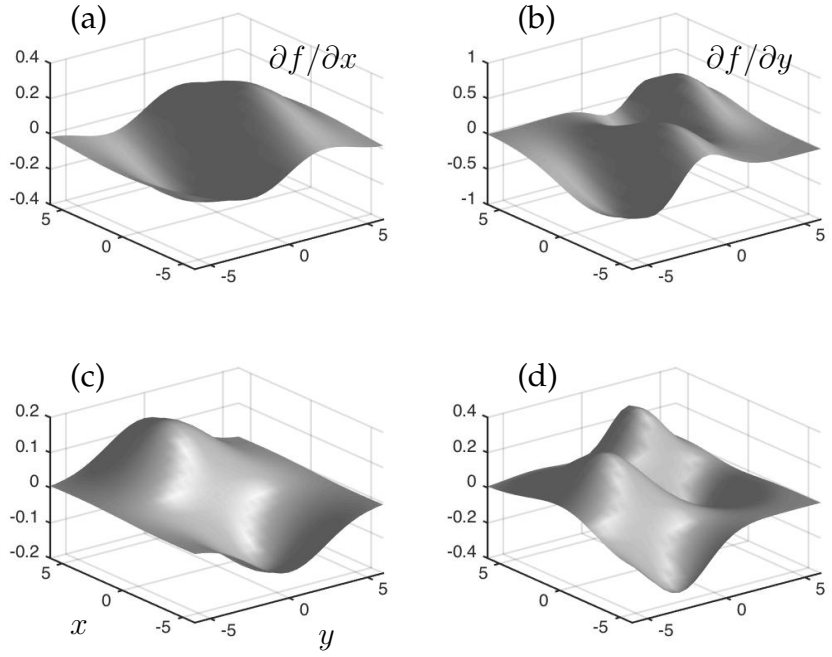


Figure 4.5: Computation of the gradient for the two functions illustrated in Fig. 4.3. In the left panels, the gradient terms (a) $\partial f/\partial x$ and (c) $\partial f/\partial y$ are computed for Fig. 4.3(a), while the right panels compute these same terms for Fig. 4.3(b) in panels (b) and (d), respectively. The **gradient** command numerically generates the gradient.

are matrices containing the values of $\partial f/\partial x$ and $\partial f/\partial y$ over the discretized domain. The gradient can be used to approximate either local or global gradients to execute the gradient descent, as shown in Fig. 4.5.

The above discussion provides a rudimentary introduction to gradient descent. A wide range of innovations have attempted to speed up this dominant nonlinear optimization procedure, including alternating descent methods. Some of these will be discussed further in the neural network chapter where gradient descent plays a critical role in training a network. For now, one can see that there are a number of issues for this nonlinear optimization procedure, including determining the initial guess, and step size δ, and computing the gradient efficiently.

Alternating Descent

Another common technique for optimizing nonlinear functions of several variables is the *alternating descent method* (ADM). Instead of computing the gradient in several variables, optimization is done iteratively in one variable at a time,

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

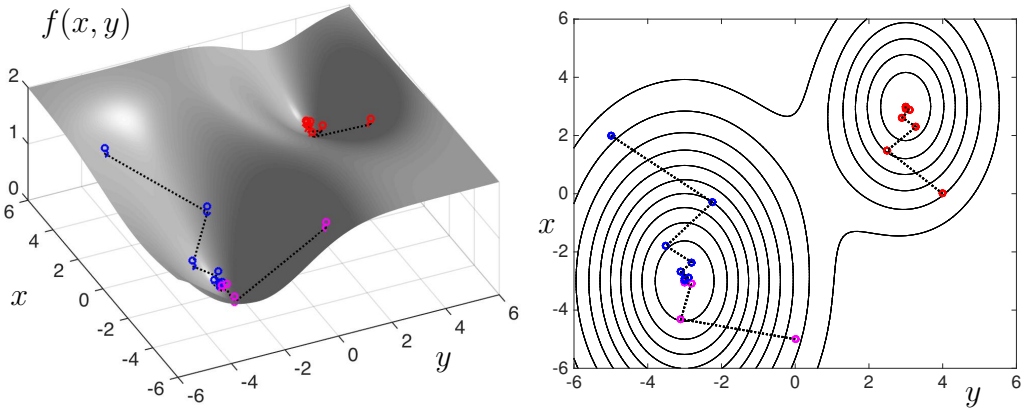


Figure 4.6: Gradient descent applied to the function featured in Fig. 4.3(b). Three initial conditions are shown: $(x_0, y_0) = \{(0,4), (-5,0), (2,-5)\}$. The first of these (red circles) gets stuck in a local minimum, while the other two initial conditions (blue and magenta) find the global minimum. Interpolation of the gradient functions of Fig. 4.5 are used to update the solutions.

as shown in Fig. 4.7. For the example just demonstrated, this would make the computation of the gradient unnecessary. The basic strategy is simple: optimize along one variable at a time, seeking the minimum while holding all other variables fixed. After passing through each variable once, the process is repeated until a desired convergence is reached. Note that the alternating descent only requires a line search along one variable at a time, thus potentially speeding up computations. Moreover, the method is derivative-free, which is attractive in many applications.

4.3 Regression and $\mathbf{A}\mathbf{x} = \mathbf{b}$: Over- and Under-Determined Systems

Curve fitting, as shown in the previous two sections, results in an optimization problem. In many cases, the optimization can be mathematically framed as solving the linear system of equations $\mathbf{A}\mathbf{x} = \mathbf{b}$. Before proceeding to discuss model selection and the various optimization methods available for this problem, it is instructive to consider that, in many circumstances in modern data science, the linear system $\mathbf{A}\mathbf{x} = \mathbf{b}$ is typically massively over- or under-determined. Over-determined systems have more constraints (equations) than unknown variables while under-determined systems have more unknowns than constraints. Thus in the former case, there are generally no solutions satisfying the linear system, and, instead, approximate solutions are found to minimize a given error. In the latter case, there are an infinite number of solutions, and

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

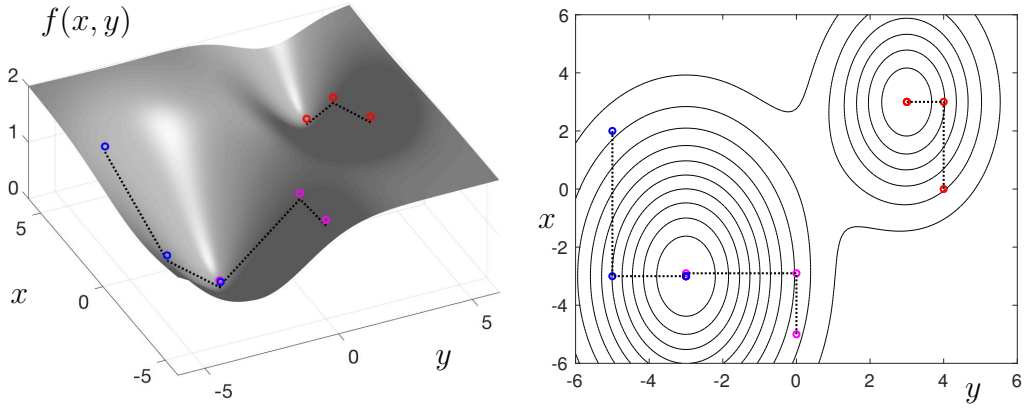


Figure 4.7: Alternating descent applied to the function in Fig. 4.3(b). Three initial conditions are shown: $(x_0, y_0) = \{(4, 0), (0, -5), (-5, 2)\}$. The first of these (red circles) gets stuck in a local minimum, while the other two initial conditions (blue and magenta) find the global minimum. No gradients are computed to update the solution. Note the rapid convergence in comparison with Fig. 4.6.

some choice of constraint must be made in order to select an appropriate and unique solution. The goal of this section is to highlight two different norms (ℓ_2 and ℓ_1) used for optimization that are used to solve $\mathbf{Ax} = \mathbf{b}$ for over- and under-determined systems. The choice of norm has a profound impact on the optimal solution achieved.

Before proceeding further, it should be noted that the system $\mathbf{Ax} = \mathbf{b}$ considered here is a restricted instance of $\mathbf{Y} = f(\mathbf{X}, \boldsymbol{\beta})$ in (4.4). Thus the solution $\mathbf{x}$ contains the *loadings* or *leverage scores* characterizing the relationship between the input data $\mathbf{A}$ and outcome data $\mathbf{b}$. A simple solution for this linear problem uses the Moore–Penrose pseudo-inverse $\mathbf{A}^\dagger$ from Section 1.4:

$$\mathbf{x} = \mathbf{A}^\dagger \mathbf{b}. \tag{4.38}$$

This operator is computed with the **pinv(A)** command in MATLAB. However, such a solution is restrictive, and a greater degree of flexibility is sought for computing solutions. Our particular aim in this section is to demonstrate the interplay in solving over- and under-determined systems using the ℓ_1- and ℓ_2-norms.

Over-determined Systems

Figure 4.8 shows the general structure of an over-determined system. As already stated, there are generally no solutions that satisfy $\mathbf{Ax} = \mathbf{b}$. Thus, the optimization problem to be solved involves minimizing the error, for example,

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

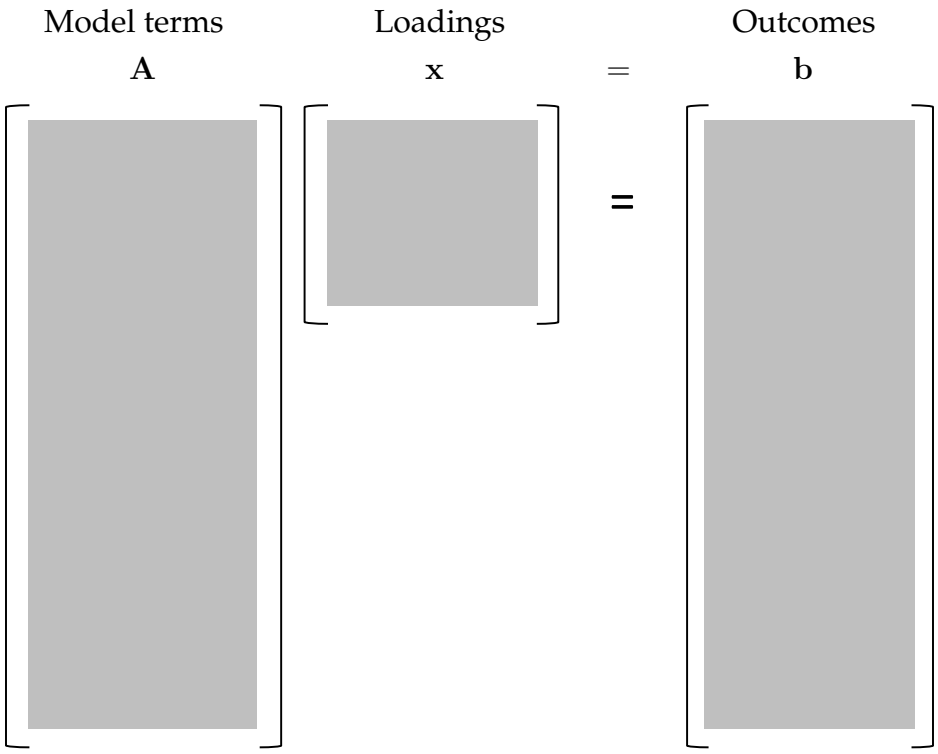


Figure 4.8: Regression framework for over-determined systems. In this case, $\mathbf{Ax} = \mathbf{b}$ cannot be satisfied in general. Thus, finding solutions for this system involves minimizing, for instance, the least-squares error $\|\mathbf{Ax} - \mathbf{b}\|_2$ subject to a constraint on the solution $\mathbf{x}$, such as minimizing the ℓ_2-norm $\|\mathbf{x}\|_2$.

the least-squares ℓ_2 error E_2, by finding an appropriate value of $\hat{\mathbf{x}}$:

$$\hat{\mathbf{x}} = \underset{\mathbf{x}}{\operatorname{argmin}} \|\mathbf{Ax} - \mathbf{b}\|_2. \tag{4.39}$$

This basic architecture does not explicitly enforce any constraints on the loadings $\mathbf{x}$. In order to both minimize the error and enforce a constraint on the solution, the basic optimization architecture can be modified to the following:

$$\hat{\mathbf{x}} = \underset{\mathbf{x}}{\operatorname{argmin}} \|\mathbf{Ax} - \mathbf{b}\|_2 + \lambda_1 \|\mathbf{x}\|_1 + \lambda_2 \|\mathbf{x}\|_2, \tag{4.40}$$

where the parameters λ_1 and λ_2 control the penalization of the ℓ_1- and ℓ_2-norms, respectively. This now explicitly enforces a constraint on the solution vector itself, not just on the error. The ability to design the penalty by adding regularizing constraints is critical for understanding *model selection* in the following.

In the examples that follow, a particular focus will be given to the role of the ℓ_1-norm. The ℓ_1-norm, as already shown in Chapter 3, promotes sparsity so

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

that many of the loadings of the solution x are zero. This will play an important role in variable and model selection in the next section. For now, consider solving the optimization problem (4.40) with $\lambda_2 = 0$. We use the open-source convex optimization package **cvx** in MATLAB [293] to compute our solution to (4.40). The following code considers various values of the ℓ_1 penalization in producing solutions to an over-determined system with 500 constraints and 100 unknowns.

Code 4.4: [MATLAB] Solutions for an over-determined system.

```
n=500; m=100;
A=rand(n,m);
b=rand(n,1);
xdag=pinv(A)*b;

lam=[0 0.1 0.5];
for j=1:3

    cvx_begin;
    variable x(m)
    minimize( norm(A*x-b,2) + lam(j)*norm(x,1) );
    cvx_end;

    subplot(4,1,j),bar(x)
    subplot(4,3,9+j), hist(x,20)
end
```

Code 4.4: [Python] Solutions for an over-determined system.

```
n = 500; m = 100
A = np.random.rand(n,m)
b = np.random.rand(n)

xdag = np.linalg.pinv(A)@b
lam = np.array([0, 0.1, 0.5])

def reg_norm(x,A,b,lam):
    return np.linalg.norm(A@x-b,ord=2) + lam*np.linalg.norm(
       x,ord=1)

fig,axs = plt.subplots(len(lam),2)
    res = minimize(reg_norm,args=(A,b,lam[j]),x0=xdag)
    x = res.x
```

Figure 4.9 highlights the results of the optimization process as a function of the parameter λ_1. It should be noted that the solution with $\lambda_1 = 0$ is equivalent to the solution **xdag** produced by computing the pseudo-inverse of the matrix

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

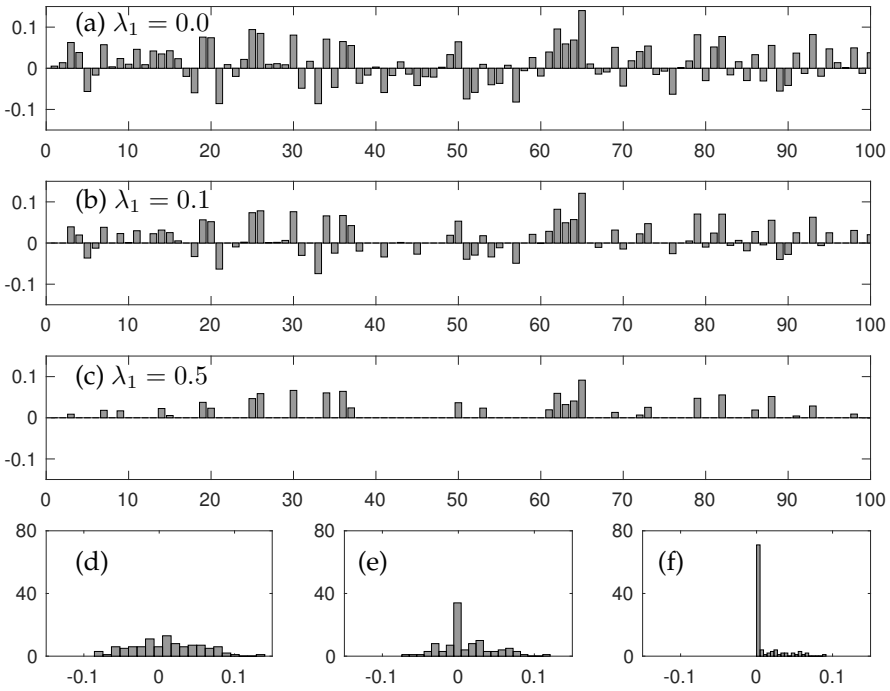


Figure 4.9: Solutions to an over-determined system with 500 constraints and 100 unknowns. Panels (a)–(c) show bar plots of the values of the loadings of the vectors $\mathbf{x}$. Note that as the ℓ_1 penalty is increased from (a) $\lambda_1 = 0$ to (b) $\lambda_1 = 0.1$ to (c) $\lambda_1 = 0.5$, the number of zero elements of the vector increases, i.e., it becomes more sparse. A histogram of the loading values for (a)–(c) is shown in panels (d)–(f), respectively. This highlights the role that the ℓ_1-norm plays in promoting sparsity in the solution.

$\mathbf{A}$. Note that the ℓ_1-norm promotes a sparse solution where many of the components of the solution vector $\mathbf{x}$ are zero. The histograms of the solution values of $\mathbf{x}$ in Fig. 4.9(d)–(f) are particularly revealing, as they show the sparsification process for increasing λ_1.

The regression for over-determined systems can be generalized to matrix systems as shown in Fig. 4.8. In this case, the **cvx** command structure simply modifies the size of the matrix $\mathbf{b}$ and solution matrix $\mathbf{x}$. Figure 4.10 shows the results of this over-determined matrix system for two different values of the added ℓ_1 penalty. Note that the addition of the ℓ_1-norm sparsifies the solution and produces a matrix that is dominated by zero entries. The two examples in Figs. 4.9 and 4.10 show the important role that the ℓ_2- and ℓ_1-norms have in generating different types of solutions. In the following sections of this book, these norms will be exploited to produce parsimonious models from data.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

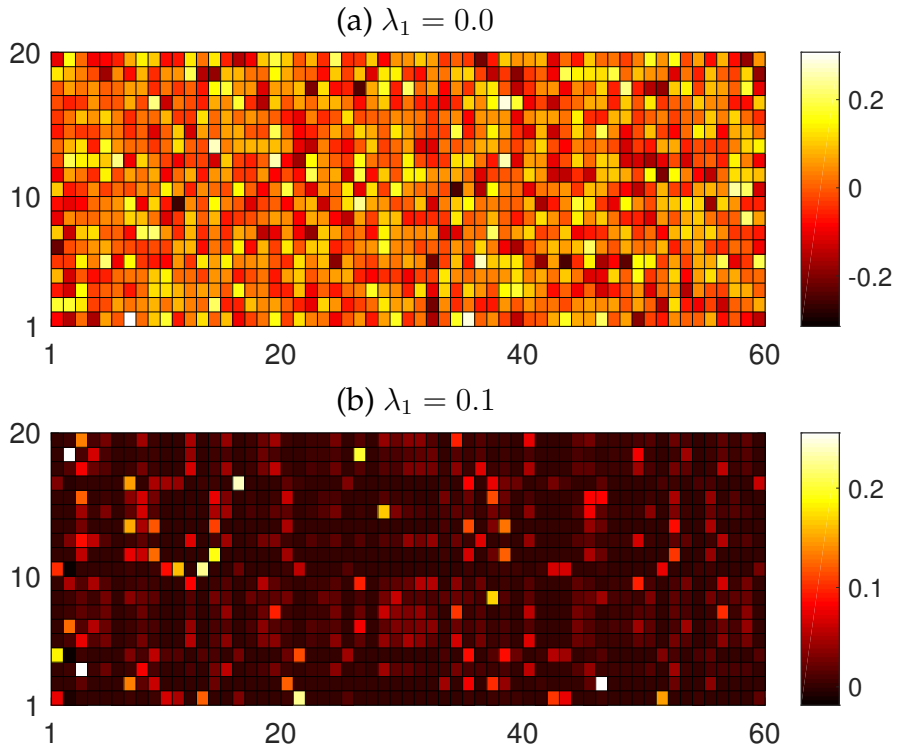


Figure 4.10: Solutions to an over-determined system $\mathbf{Ax} = \mathbf{b}$ with 300 constraints and 60×20 unknowns. Panels (a) and (b) show a plot of the values of the loadings of the matrix $\mathbf{x}$ with ℓ_1 penalty (a) $\lambda_1 = 0$ to (b) $\lambda_1 = 0.1$.

Under-determined Systems

For underdetermined systems, there are an infinite number of possible solutions satisfying $\mathbf{Ax} = \mathbf{b}$. The goal in this case is to impose an additional constraint, or set of constraints, whereby a unique solution is generated from the infinite possibilities. The basic mathematical structure is shown in Fig. 4.11. As an optimization, the solution to the under-determined system can be stated as

$$\min \|\mathbf{x}\|_p \quad \text{subject to} \quad \mathbf{Ax} = \mathbf{b}, \tag{4.41}$$

where the p denotes the p-norm of the vector $\mathbf{x}$. For simplicity, we consider the ℓ_2- and ℓ_1-norms only. As has already been shown for over-determined systems, the ℓ_1-norm promotes sparsity of the solution.

We again use the convex optimization package `cvx` to compute our solution to (4.42). The following code considers both ℓ_2 and ℓ_1 penalization in producing solutions to an under-determined system with 20 constraints and 100 unknowns.

Code 4.5: [MATLAB] Solutions for an under-determined matrix system.

```
n=20; m=100
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

```
A=rand(n,m); b=rand(n,1);

cvx_begin;
variable x2(m)
minimize( norm(x2,2) );
subject to
A*x2 == b;
cvx_end;

cvx_begin;
variable x1(m)
minimize( norm(x1,1) );
subject to
A*x1 == b;
cvx_end;
```

Code 4.5: [Python] Solutions for an under-determined matrix system.

```
n = 20; m = 100
A = np.random.rand(n,m)
b = np.random.rand(n)

def two_norm(x):
    return np.linalg.norm(x,ord=2)

def one_norm(x):
    return np.linalg.norm(x,ord=1)

constr = ({'type': 'eq', 'fun': lambda x:  A @ x - b})
x0 = np.random.rand(m)
res = minimize(two_norm, x0, method='SLSQP',constraints=
   constr)
x2 = res.x

res = minimize(one_norm, x0, method='SLSQP',constraints=
   constr)
x1 = res.x
```

This code produces two solution vectors **x2** and **x1**, which minimize the ℓ_2- and ℓ_1-norm, respectively. Note the way that **cvx** allows one to impose constraints in the optimization routine. Figure 4.12 shows a bar plot and histogram of the two solutions produced. As before, the sparsity-promoting ℓ_1-norm yields a solution vector dominated be zeros. In fact, for this case, there are exactly 80 zeros for this linear system since there are only 20 constraints for the 100 unknowns.

As with the over-determined system, the optimization can be modified to handle more general under-determined matrix equations, as shown in Fig. 4.11.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

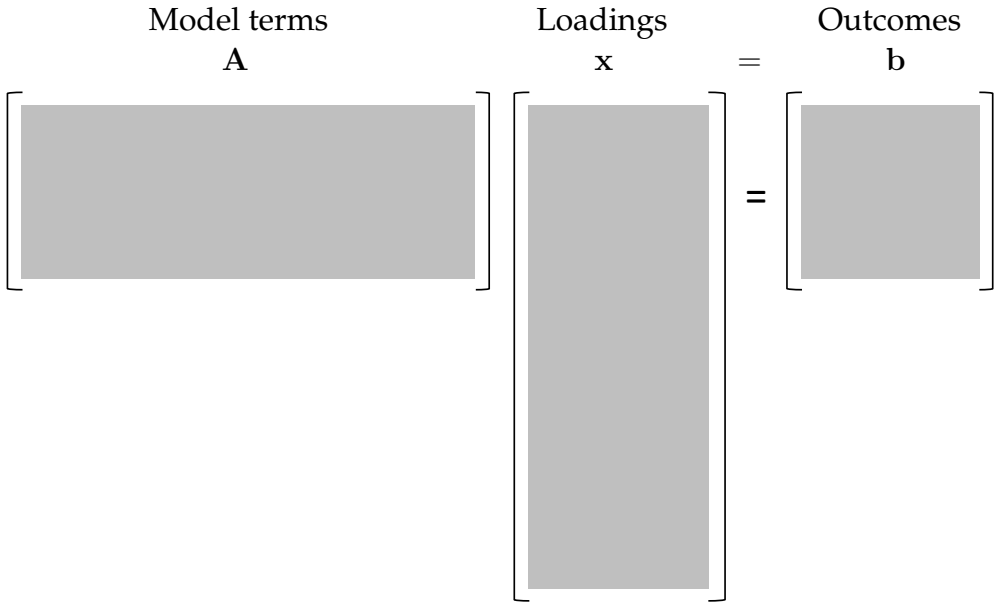


Figure 4.11: Regression framework for under-determined systems. In this case, $\mathbf{Ax} = \mathbf{b}$ can be satisfied. In fact, there are an infinite number of solutions. Thus pinning down a unique solution for this system involves minimizing a constraint. For instance, from an infinite number of solutions, we choose the one that minimizes the ℓ_2-norm $\|\mathbf{x}\|_2$, which is subject to the constraint $\mathbf{Ax} = \mathbf{b}$.

The `cvx` optimization package may be used for this case as before with over-determined systems. The software engine can also work with more general p-norms as well as minimize with both ℓ_1 and ℓ_2 penalties simultaneously. For instance, a common optimization modifies (4.42) to the following:

$$\min(\lambda_1\|\mathbf{x}\|_1 + \lambda_2\|\mathbf{x}\|_2) \quad \text{subject to} \quad \mathbf{Ax} = \mathbf{b}, \tag{4.42}$$

where the weighting between λ_1 and λ_2 can be used to promote a desired sparsification of the solution. These different optimization strategies are common and will be considered further in the following.

4.4 Optimization as the Cornerstone of Regression

In the previous two sections of this chapter, the fitting function $f(x)$ was specified. For instance, it may be desirable to produce a line fit so that $f(x) = \beta_1 x + \beta_2$. The coefficients are then found by the regression and optimization methods already discussed. In what follows, our objective is to develop techniques which allow us to objectively select a good model for fitting the data, i.e., should one use a quadratic or cubic fit? The error metric alone does not dictate a good

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

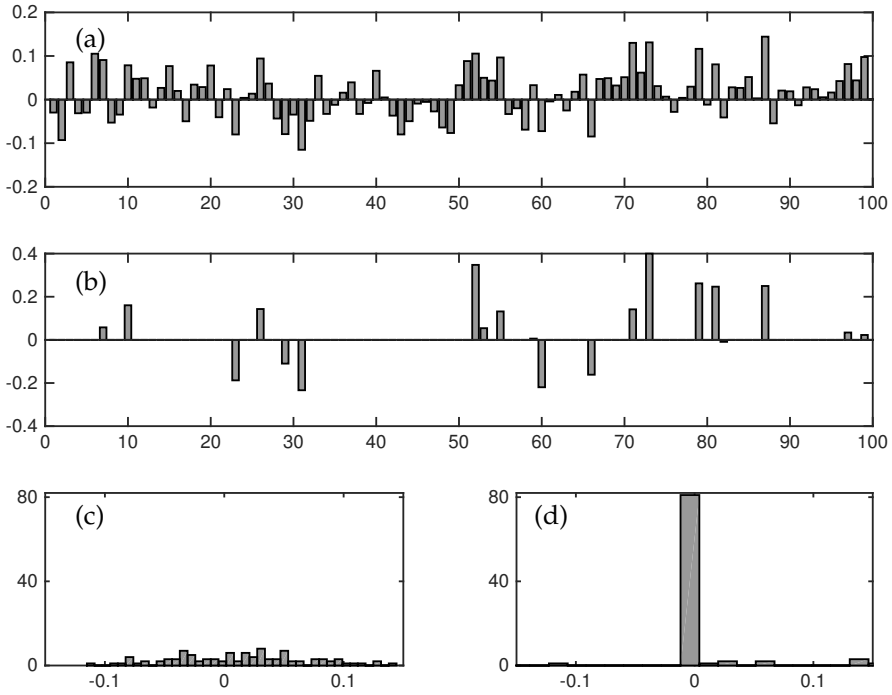


Figure 4.12: Solutions to an under-determined system with 20 constraints and 100 unknowns. Panels (a) and (b) show bar plots of the values of the loadings of the vectors x. In panel (a), the optimization is subject to minimizing the ℓ_2-norm of the solution, while panel (b) is subject to minimizing the ℓ_1-norm. Note that the ℓ_1 penalization produces a sparse solution vector. A histogram of the loading values for (a) and (b) is shown in panels (c) and (d), respectively.

model selection, as the more terms that are chosen for fitting, the more parameters are available for lowering the error, regardless of whether the additional terms have any meaning or interpretability.

Optimization strategies will play a foundational role in extracting interpretable results and meaningful models from data. As already shown in previous sections, the interplay of the ℓ_2- and ℓ_1-norms has a critical impact on the optimization outcomes. To illustrate further the role of optimization and the variety of possible outcomes, consider the simple example of data generated from noisy measurements of a parabola:

$$f(x) = x^2 + \mathcal{N}(0, \sigma), \tag{4.43}$$

where $\mathcal{N}(0, \sigma)$ is a normally distributed random variable with mean zero and standard deviation σ. Figure 4.13(a) shows an example of 100 random measurements of (4.43). The parabolic structure is clearly evident despite the noise added to the measurement. Indeed, a parabolic fit is trivial to compute using the classic least-squares fitting methods outlined in the first section of this chap-

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

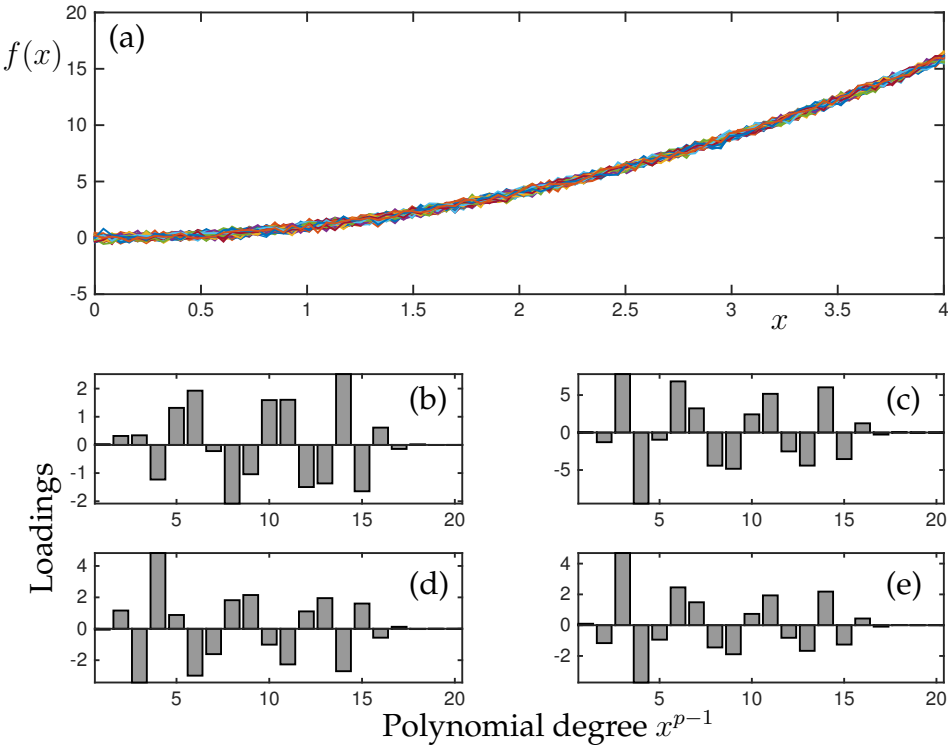


Figure 4.13: (a) One hundred realizations of the parabolic function (4.43) with additive white noise parameterized by $\sigma = 0.1$. Although the noise is small, the least-squares fitting procedure produces significant variability when fitting to a polynomial of degree 20. Panels (b)–(e) demonstrate the loadings (coefficients) for the various polynomial coefficients for four different noise realizations. This demonstrated model variability frames the model selection architecture.

ter.

The goal is to *discover* the best model for the data given. So, instead of specifying a model *a priori*, in practice, we do not know what the function is and need to discover it. We can begin by positing a regression to a set of polynomial models. In particular, consider framing the model selection problem $\mathbf{Y} = f(\mathbf{X}, \boldsymbol{\beta})$ of (4.4) as the following system $\mathbf{Ax} = \mathbf{b}$:

$$\begin{bmatrix} | & | & | & & | \\ 1 & x_j & x_j^2 & \cdots & x_j^{p-1} \\ | & | & | & & | \end{bmatrix} \begin{bmatrix} \beta_1 \\ \vdots \\ \beta_p \end{bmatrix} = \begin{bmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_{100}) \end{bmatrix}, \tag{4.44}$$

where the matrix $\mathbf{A}$ contains polynomial models up to degree $p-1$, with each row representing a measurement, the β_k are the coefficients for each polynomial, and the matrix $\mathbf{b}$ contains the outcomes (data) $f(x_j)$. In what follows, we will consider a scenario where 100 measurements are taken and a 20-term

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

(19th-order) polynomial is fit. Thus the matrix system $\mathbf{Ax} = \mathbf{b}$ results in an over-determined system as illustrated in Fig. 4.8. Figure 4.13(b)–(e) shows four typical loadings $\boldsymbol{\beta}$ computed from the regression procedure. Note that, despite the low level of noise added, the loadings are significantly different from one another. Thus each noise realization produces a very different model to explain the data.

The variability of the regression results is problematic for model selection. It suggests that even a small amount of measurement noise can lead to significantly different conclusions about the underlying model. In what follows, we quantify this variability while also considering various regression procedures for solving the over-determined linear system $\mathbf{Ax} = \mathbf{b}$. Highlighted here are five standard methods: least-squares regression (**pinv**), the backslash operator (\), (least absolute shrinkage and selection operator) LASSO (**lasso**), robust fit (**robustfit**), and ridge regression (**ridge**). Returning to the last section, and specifically (4.40), helps frame the mathematical architecture for these various $\mathbf{Ax} = \mathbf{b}$ solvers. Specifically, the Moore–Penrose pseudo-inverse (**pinv**) solves (4.40) with $\lambda_1 = \lambda_2 = 0$. The backslash command (\) for over-determined systems solves the linear system via a QR decomposition [711]. The LASSO (**lasso**) solves (4.40) with $\lambda_1 > 0$ and $\lambda_2 = 0$. Ridge regression (**ridge**) solves (4.40) with $\lambda_1 = 0$ and $\lambda_2 > 0$. However, the modern implementation of ridge in MATLAB is a bit more nuanced. The popular elastic net algorithm weights both the ℓ_2 and ℓ_1 penalty, thus providing a tunable hybrid model regression between ridge and LASSO. Robust fit (**robustfit**) solves (4.40) by a weighted least-squares fitting. Moreover, it allows one to leverage robust statistics methods and penalize according to the Huber norm so as to promote outlier rejection [343]. In the data considered here, no outliers are imposed on the data, so that the power of robust fit is not properly leveraged. Regardless, it is an important technique one should consider.

Figure 4.14 shows a series of box plots for 100 realizations of data that illustrate the differences with the various regression techniques considered. It also highlights critically important differences with optimization strategies based on the ℓ_2- and ℓ_1-norm. From a model selection point of view, the least-squares fitting procedure produces significant variability in the loading parameters $\boldsymbol{\beta}$ as illustrated in Fig. 4.14(a,b,e). The least-squares fitting was produced by the Moore–Penrose pseudo-inverse or QR decomposition, respectively. If some ℓ_1 penalty (regularization) is allowed, then Fig. 4.14(c,d,f) show that a more parsimonious model is selected with low variability. This is expected, as the ℓ_1-norm sparsifies the solution vector of loading values $\boldsymbol{\beta}$. Indeed, the standard LASSO regression correctly selects the quadratic polynomial as the dominant contribution to the data.

Despite the significant variability exhibited in Fig. 4.14 for most of the loading values by the different regression techniques, the error produced in the

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

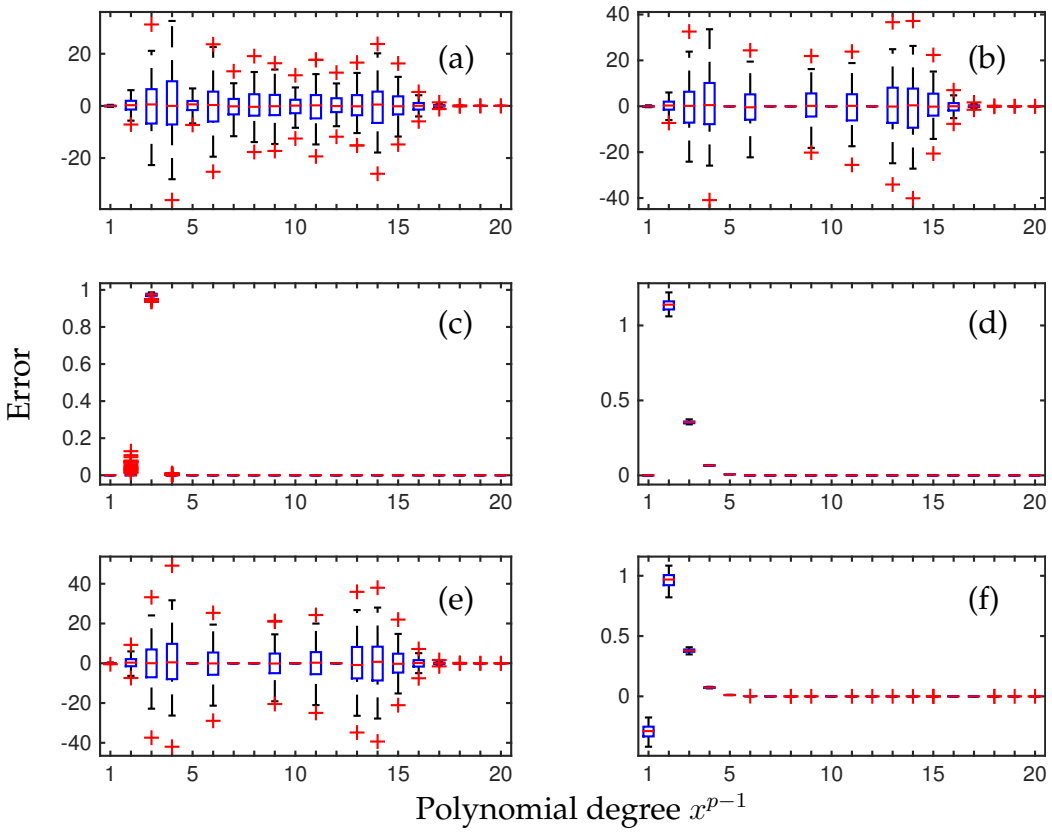


Figure 4.14: Comparison of regression methods for $\mathbf{Ax} = \mathbf{b}$ for an overdetermined system of linear equations. The 100 realizations of data are generated from a simple parabola (4.43) that is fit to a 20th-degree polynomial via (4.44). The box plots show (a) least-squares regression via the Moore–Penrose pseudo-inverse (**pinv**), (b) the backslash command (\), (c) LASSO regression (**lasso**), (d) LASSO regression with different ℓ_2 versus ℓ_1 penalization, (e) robust fit, and (f) ridge regression. Note the significant variability in the loading values for the strictly ℓ_2-based methods ((a), (b), and (e)), and the low-variability for ℓ_1-weighted methods and ridge ((c), (d), and (f)). Only the standard LASSO (c) identifies the dominance of the parabolic term.

fitting procedure has little variability. Moreover, the various methods all produce regressions that have comparable error. Thus despite their differences in optimization frameworks, the error from fitting is relatively agnostic to the underlying method. This suggests that using the error alone as a metric for model selection is potentially problematic, since almost any method can produce a reliable, low-error model. Figure 4.15(a) shows a box plot of the error produced using the regression methods of Fig. 4.14. All of the regression techniques produce comparably low-error and low-variability results using significantly different strategies.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

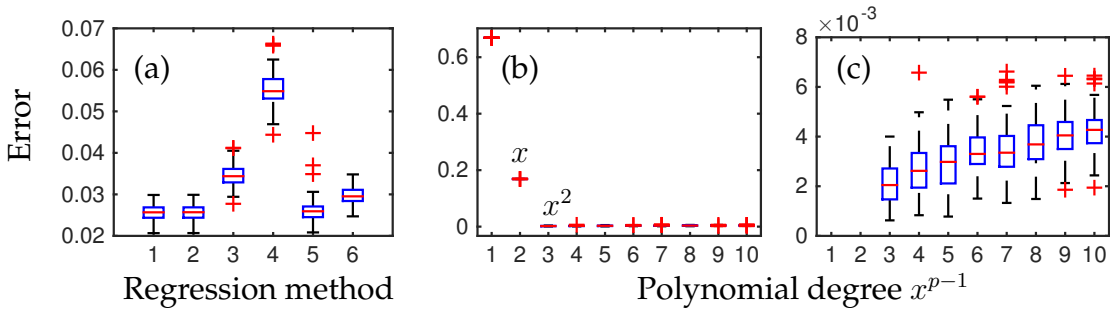


Figure 4.15: (a) Comparison of the error for the six regression methods used in Fig. 4.14. Despite the variability across the optimization methods, all of them produce low-error solutions. (b) Error using least-squares regression as a function of increasing degree of polynomial. The error drops rapidly until the quadratic term is used in the regression. (c) Detail of the error showing that the error actually increases slightly by using a higher-degree polynomial to fit the data.

As a final note to this section and the code provided, we can consider instead the regression procedure as a function of the number of polynomials in (4.44). In our example of Fig. 4.14, polynomials up to degree 20 were considered. If, instead, we sweep through polynomial degrees, then something interesting and important occurs, as illustrated in Fig. 4.15(b)–(c). Specifically, the error of the regression collapses to 10^{-3} after the quadratic term is added, as shown in panel (b). This is expected since the original model was a quadratic function with a small amount of noise. Remarkably, as more polynomial terms are added, the ensemble error actually increases in the regression procedure, as highlighted in panel (c). Thus simply adding more terms does not improve the error, which is counter-intuitive at first. Note that we have only swept through polynomials up to degree 10. Note further that Fig. 4.15(c) is a detail of panel (b). The error produced by a simple parabolic fit is approximately twice as good as a polynomial with degree 10. These results will help frame our model selection framework of the remaining sections.

4.5 The Pareto Front and *Lex Parsimoniae*

The preceding chapters have shown that regression is more nuanced than simply choosing a model and performing a least-squares fit. Not only are there numerous metrics for constraining the solution, the model itself should be carefully selected in order to achieve a better, more interpretable description of the data. Such considerations on an appropriate model date back to William of Occam (c. 1287–1347), who was an English Franciscan friar, scholastic philosopher, and theologian. Occam proposed his law of parsimony (in Latin *lex parsimo-*

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

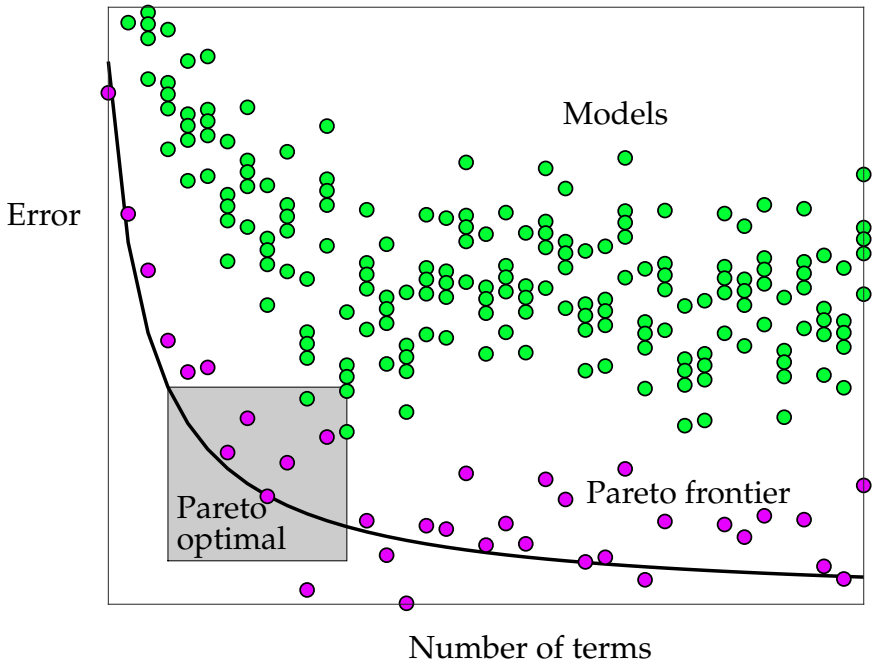


Figure 4.16: For model selection, the criteria of accuracy (low error) is balanced against parsimony. There can be a variety of models with the same number of terms (green and magenta points), but the *Pareto frontier* (magenta points) is defined by the envelope of models that produce the lowest error for a given number of terms. The solid line provides an approximation to the Pareto frontier. The *Pareto optimal* solutions (shaded region) are those models that produce accurate models while remaining parsimonious.

niae), commonly known as Occam's razor, whereby he stated that, among competing hypotheses, the one with the fewest assumptions should be selected, or when you have two competing theories that make exactly the same predictions, the simpler one is the more likely. The philosophy of Occam's razor has been used extensively throughout the physical and biological sciences for developing governing equations to model observed phenomena.

Parsimony also plays a central role in the mathematical work of Vilfredo Pareto (c. 1848–1923). Pareto was an Italian engineer, sociologist, economist, political scientist, and philosopher. He made several important contributions to economics, specifically in the study of income distribution and in the analysis of individuals' choices. He was also responsible for popularizing the use of the term *elite* in social analysis. In more recent times, he has become known for the popular 80/20 rule, which is qualitatively illustrated in Fig. 4.16, named after him as the "Pareto principle" by management consultant Joseph M. Juran in 1941. Stated simply, it is a common principle in business and consulting management, for instance, that observes that 80% of sales come from 20% of clients.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

This concept was popularized by Richard Koch's book *The 80/20 Principle* [395] (along with several follow-up books [396, 397, 398]), which illustrated a number of practical applications of the Pareto principle in business management and life.

Pareto and Occam ultimately advocated the same philosophy: explain the majority of observed data with a parsimonious model. Importantly, model selection is not simply about reducing error; rather, it is about producing a model that has a high degree of interpretability, generalization, and predictive capabilities. Figure 4.16 shows the basic concept of the *Pareto frontier* and *Pareto optimal* solutions. Specifically, for each model considered, the number of terms and the error in matching the data are computed. The solutions with the lowest error for a given number of terms define the Pareto frontier. Those parsimonious solutions that optimally balance error and complexity are in the shaded region and represent the Pareto optimal solutions. In game theory, the Pareto optimal solution is thought of as a strategy that cannot be made to perform better against one opposing strategy without performing less well against another (in this case, error and complexity). In economics, it describes a situation in which the profit of one party cannot be increased without reducing the profit of another. Our objective is to select, in a principled way, the best model from the space of Pareto optimal solutions. To this end, information criteria, which will be discussed in subsequent sections, will be used to select from candidate modes in the Pareto optimal region.

Overfitting

The Pareto concept needs amending when considering application to real data. Specifically, when building models with many free parameters, which is often the case in machine learning applications with high-dimensional data, it is easy to overfit a model to the data. Indeed, the increase in error illustrated in Fig. 4.15(c) as a function of increasing model complexity illustrates this point. Thus, unlike what is depicted in Fig. 4.16, where the error goes towards zero as the number of model terms (parameters) is increased, the error may actually increase when considering models with a higher number of terms and/or parameters. To determine the correct model, various cross-validation and model selection algorithms are necessary.

To illustrate the overfitting that occurs with real data, consider the simple example of the last section. In this example, we are simply trying to find the correct parabolic model measured with additive noise (4.43). The results of Figs. 4.15(b) and 4.15(c) already indicate that overfitting is occurring for polynomial models beyond second order. The following MATLAB example will highlight the effects of overfitting. Consider the production of a training and test set for the parabola of (4.43). The training set is on the region $x \in [0, 4]$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

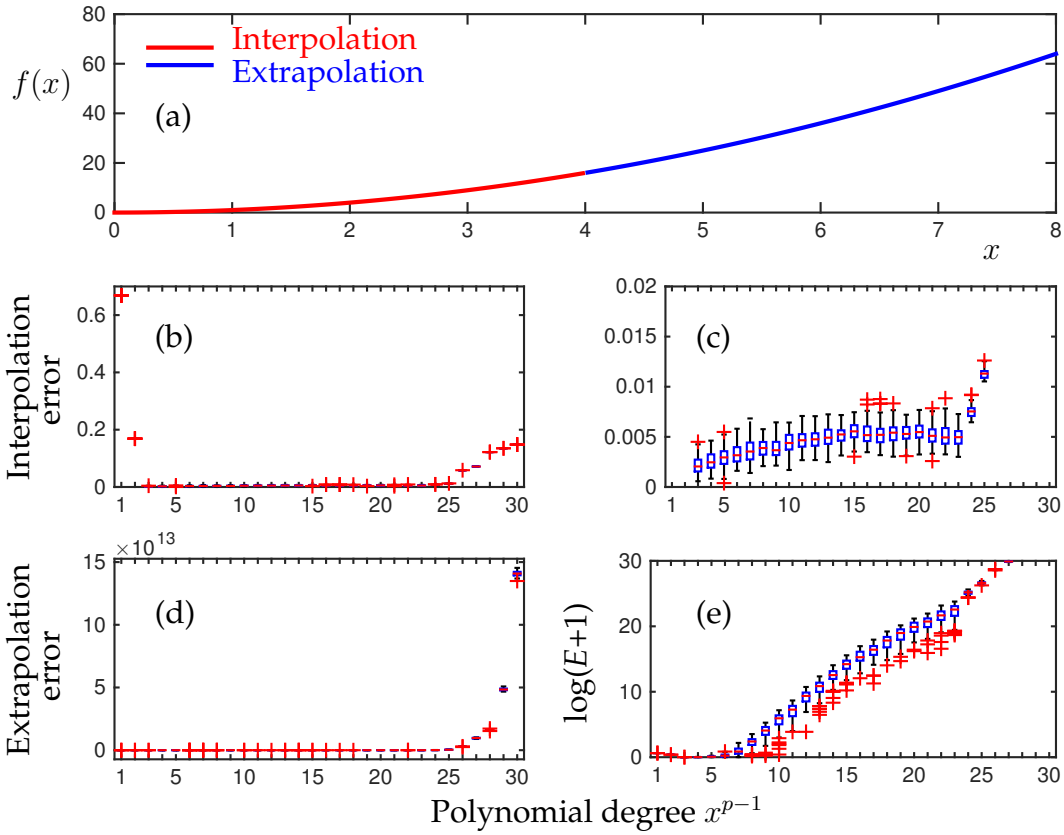


Figure 4.17: (a) The ideal model $f(x) = x^2$ over the domain $x \in [0, 8]$. Data is collected in the region $x \in [0, 4]$ in order to build a polynomial regression model (4.44) with increasing polynomial degree. In the interpolation regime $x \in [0, 4]$, the model error stays constrained, with increasing error due to overfitting for polynomials of degree greater than two. The error is shown in panel (b) with a zoom-in of the error in panel (c). For extrapolation, $x \in [4, 8]$, the error grows exponentially beyond a parabolic fit. In panel (d), the error is shown to grow to 10^{13}. A zoom-in of the region on a logarithmic scale of the error ($\log(E + 1)$, where unity is added so that zero error produces a zero score) shows the exponential growth of error. This clearly shows that the model trained on the interval $x \in [0, 4]$ does not generalize (extrapolate) to the region $x \in [4, 8]$. This example should serve as a serious warning and note of caution in model fitting.

while the test set (extrapolation region) will be for $x \in [4, 8]$.

This produces the ideal model on two distinct regions: $x \in [0, 4]$ and $x \in [4, 8]$. Once measurement noise is added to the model, then the parameters for a polynomial fit no longer produce the perfect parabolic model. We can compute for given noisy measurements both an interpolation error, where measurements are taken in the data regime of $x \in [0, 4]$, and extrapolation error,

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

where measurements are taken in the data regime of $x \in [4, 8]$. For this example, a least-squares regression is performed using the pseudo-inverse (**pinv**) from MATLAB.

This simple example shows some of the most basic and common features associated with overfitting of models. Specifically, overfitting does not allow for generalization. Consider the results of Fig. 4.17 generated from the above code. In this example, the least-squares loadings (4.44) for a polynomial are computed using the pseudo-inverse for data in the range $x \in [0, 4]$. The interpolation error for these loadings is demonstrated in Figs. 4.17(b) and (c). Note the impact of overfitting by polynomials for this interpolation of the data. Specifically, the error of the interpolated fit increases from beyond a second-degree polynomial. Extrapolation for an overfit model produces significant errors. Figure 4.17(d) and (e) show the error growth as a function of the least-squares fit pth-degree polynomial model. The error in Fig. 4.17(d) is on a logarithmic plot since it grows to 10^{13}. This demonstrates a clear inability of the overfit model to generalize to the range $x \in [4, 8]$. Indeed, only a parsimonious model with a second-degree polynomial can easily generalize to the range $x \in [4, 8]$ while keeping the error small.

The above example shows that some form of model selection to systematically deduce a parsimonious model is critical for producing viable models that can generalize outside of where data is collected. Much of machine learning revolves around (i) using data to generate predictive models, and (ii) applying cross-validation techniques to remove the most deleterious effects of overfitting. Without a cross-validation strategy, one will almost certainly produce a non-generalizable model such as that exhibited in Fig. 4.17. In what follows, we will consider some standard strategies for producing reasonable models.

4.6 Model Selection: Cross-Validation

The previous section highlights many of the fundamental problems with regression. Specifically, it is easy to overfit a model to the data, thus leading to a model that is incapable of generalizing for extrapolation. This is an especially pernicious issue in training deep neural nets. To overcome the consequences of overfitting, various techniques have been proposed to more appropriately select a parsimonious model with only a few parameters, thus balancing the error with a model that can more easily generalize or extrapolate. This provides a reinterpretation of the Pareto front in Fig. 4.16. Specifically, the error increases dramatically with the number of terms due to overfitting, especially when used for extrapolation.

There are two common mathematical strategies for circumventing the effects of overfitting in model selection: *cross-validation* and computing *informa-*

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

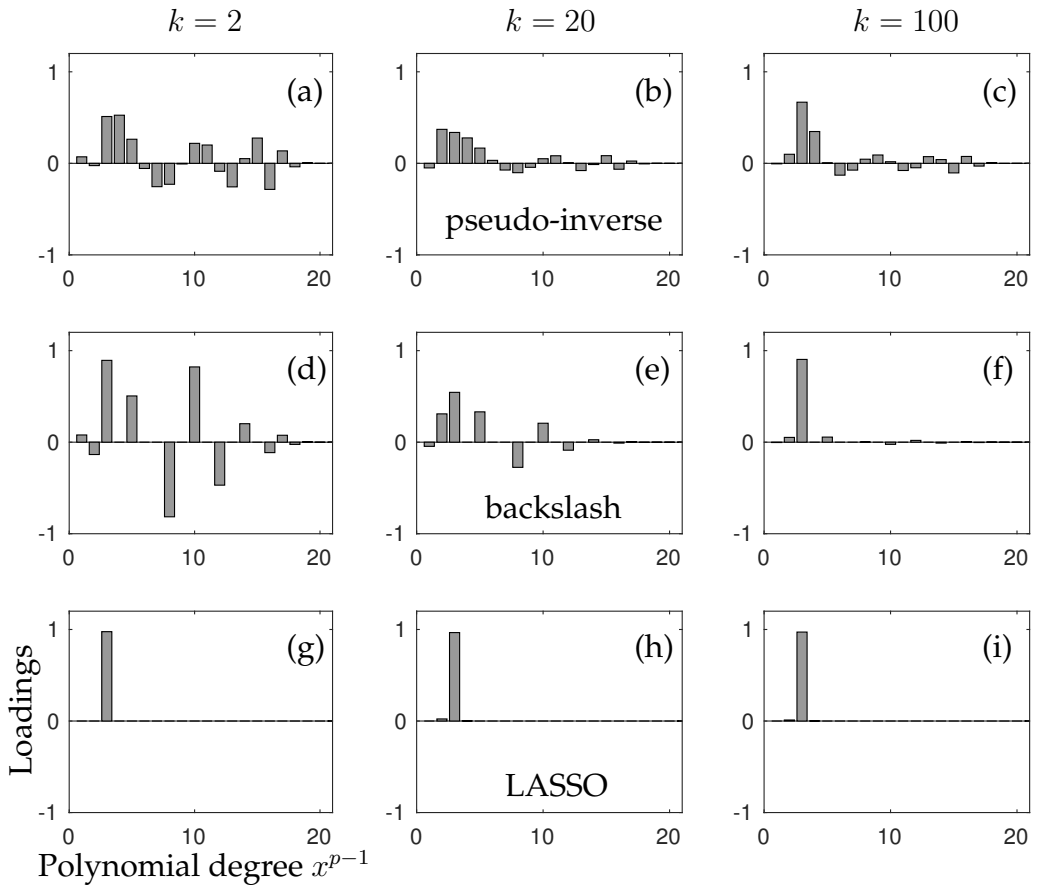


Figure 4.18: Cross-validation using k-fold strategy with $k = 2$, 20, and 100 (left, middle, and right columns, respectively). Three different regression strategies are cross-validated: least-squares fitting of pseudo-inverse, the QR-based backslash, and the sparsity-promoting LASSO. Note that the LASSO for this example produces the quadratic model within even a one- or two-fold validation. The backslash-based QR algorithm has a strong signature after 100-fold cross-validation, while the least-squares fitting suggests that the quadratic and cubic terms are both important even after 100-fold cross-validation.

tion criteria. This section considers the former, while the latter method is considered in the next section.

Cross-validation strategies are perhaps the most common and critical techniques in almost all machine learning algorithms. Indeed, one should never trust a model unless properly cross-validated. Cross-validation can be stated quite simply: Take random portions of your data and build a model. Do this k times and average the parameter scores (regression loadings) to produce the cross-validated model. Test the model predictions against withheld (extrapolation) data and evaluate whether the model is actually any good. This commonly used strategy is called k-fold cross-validation. It is simple, intuitively

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

appealing, and the k-fold model-building procedure produces a statistically based model for evaluation.

To illustrate the concept of cross-validation, we will once again consider fitting polynomial models to the simple function $f(x) = x^2$ (see Fig. 4.18). The previous sections of this chapter have already considered this problem in detail, looking at both the various regression frameworks available (pseudo-inverse, LASSO, robust fit, etc.), as well as their ability to accurately produce a model for interpolating and extrapolating data. The following MATLAB code considers three regression techniques (least-squares fitting of pseudo-inverse, the QR-based backslash, and the sparsity-promoting LASSO) for k-fold cross-validation ($k = 2$, 20, and 100). In this case, one can think of the k snapshots of data as trial measurements. As one might expect, there would be an advantage as more trials are taken, and $k = 100$ models are averaged for a final model.

Figure 4.18 shows the results of the k-fold cross-validation computations. By promoting sparsity (parsimony), the LASSO achieves the desired quadratic model after even a single $k = 1$ fold (i.e., thus this is not even cross-validated). In contrast, the least-squares regression (pseudo-inverse) and QR-based regression both require a significant number of folds to produce the dominant quadratic term. The least-squares regression, even after $k = 100$ folds, still includes both a quadratic and a cubic term.

The final model selection process under k-fold cross-validation often can involve a *thresholding* of terms that are small in the regression. We demonstrate the regression using three modeling strategies. Although the LASSO looks almost ideal, it still has a small contributing linear component. The QR strategy of backslash produces a number of small components scattered among the polynomials used in the fit. The least-squares regression has the dominant quadratic and cubic terms with a large number of non-zero coefficients scattered across the polynomials. If one thresholds the loadings, then the LASSO and backslash will produce exactly the quadratic model, while the least-squares fit produces a quadratic–cubic model. The loading coefficients are thresholded to produce the final cross-validated model. This model can then be evaluated against both the interpolated and extrapolated data regions as in Fig. 4.19.

The results of Fig. 4.19 show that the model selection process, and the regression technique used, makes a critical difference in producing a viable model. It further shows that, despite a k-fold cross-validation, the extrapolation error, or generalizability, of the model can still be poor. A good model is one that keeps errors small and also generalizes well, as does the LASSO in the previous example.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

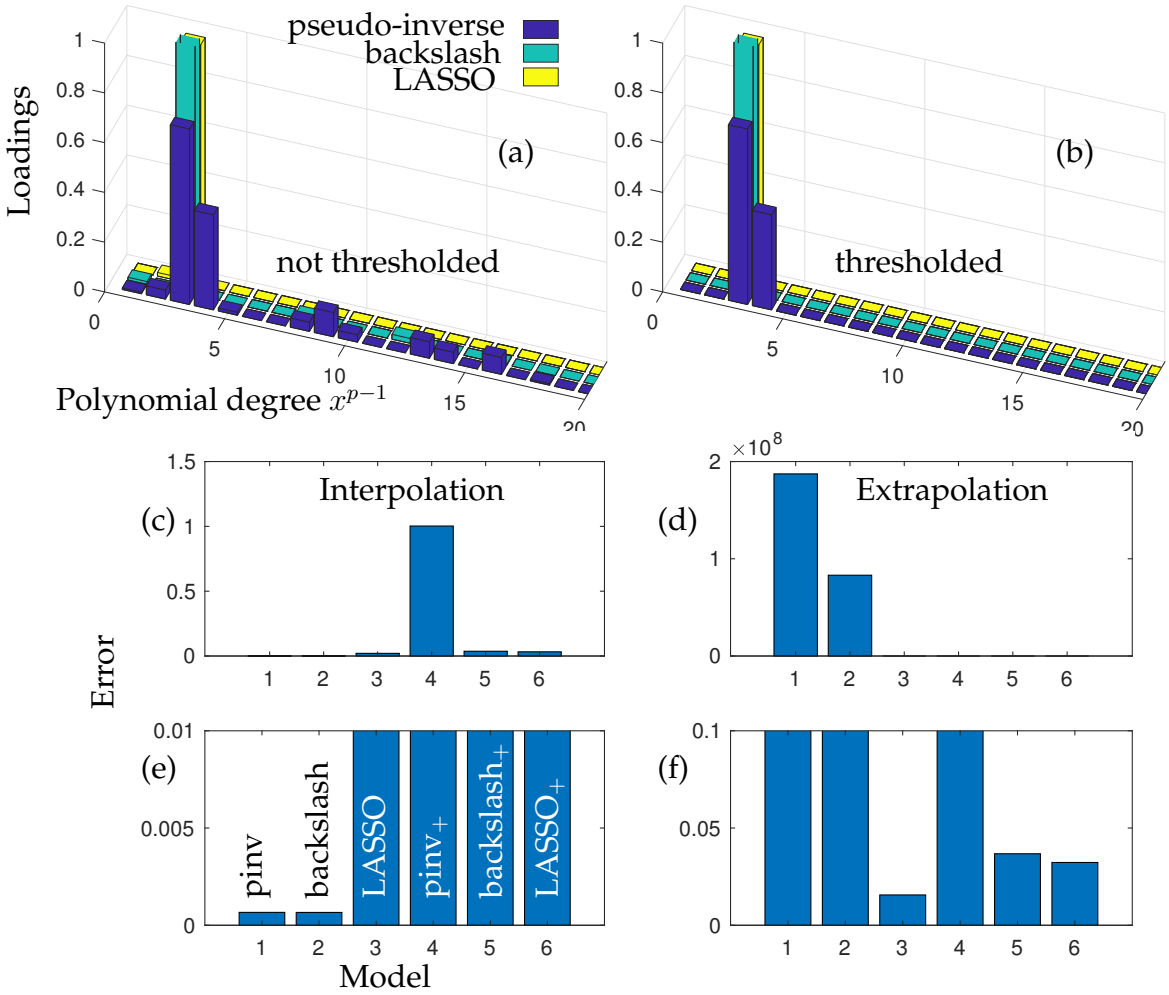


Figure 4.19: Error and loading results for $k = 100$-fold cross-validation. The loadings for the k-fold validation (panel (b) with thresholding denoted by subscript $+$, and panel (a) without thresholding) are shown for least-squares fitting of pseudo-inverse, the QR-based backslash, and the sparsity-promoting LASSO (see Fig. 4.18). Both the interpolation error (panel (c) and detail in (e)) and extrapolation error (panel (d) and detail in (f)) are computed. The LASSO performs well for both interpolation and extrapolation, while a least-squares fit gives poor performance under extrapolation. The six models considered are: 1, pseudo-inverse; 2, backslash; 3, LASSO; 4, thresholded pseudo-inverse; 5, thresholded backslash; and 6, thresholded LASSO.

k-Fold Cross-Validation

The process of k-fold cross-validation is highlighted in Fig. 4.20. The concept is to partition a data set into a training set and a test set. The test set, or withhold set, is kept separate from any training procedure for the model. Importantly, the test set is where the model produces an extrapolation approximation, which

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

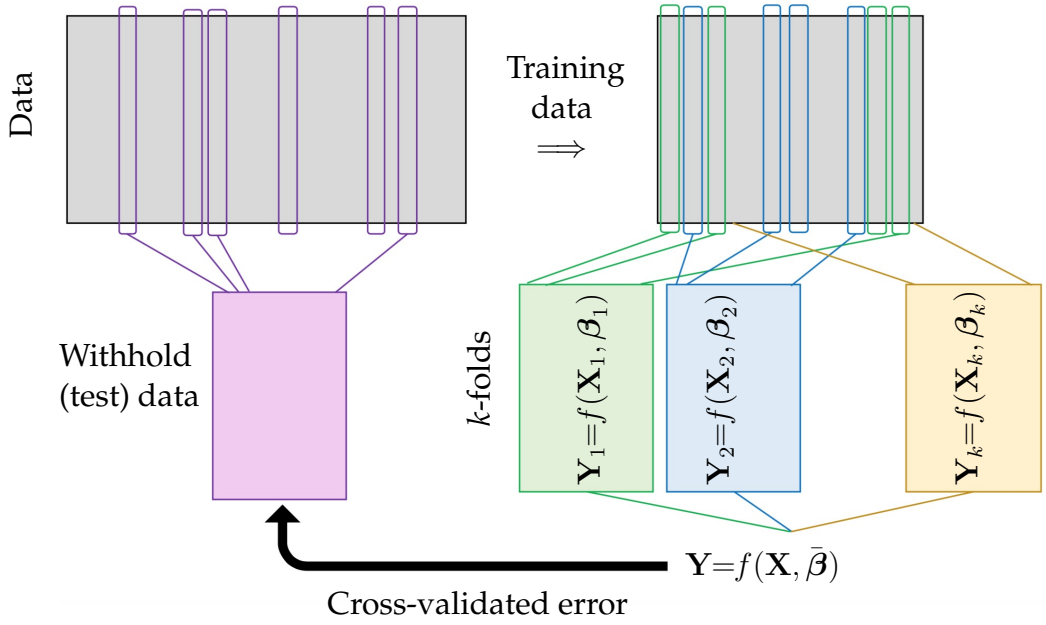


Figure 4.20: Procedure for k-fold cross-validation of models. The data is initially partitioned into a training set and test (withhold) set. Typically, the withhold set is generated from a random sample of the overall data. The training data is partitioned into k-folds whereby a random sub-selection of the training data is collected in order to build a regression model $\mathbf{Y}_j = f(\mathbf{X}_j, \boldsymbol{\beta}_j)$. Importantly, each model generates the loading parameters $\boldsymbol{\beta}_j$. After the k-fold models are generated, the best model $\mathbf{Y} = f(\mathbf{X}, \bar{\boldsymbol{\beta}})$ is produced. There are different ways to get the best model; in some cases, it may be appropriate to average the model parameters so that $\bar{\boldsymbol{\beta}} = (1/k)\sum_{j=1}^{k} \boldsymbol{\beta}_j$. One could also simply pick the best parameters from the k-fold set. In either case, the best model is then tested on the withheld data to evaluate its viability.

the figures of the last two sections show to be challenging. In k-fold cross-validation, the training data is further partitioned into k-folds, which are typically randomly selected portions of the data. For instance, in standard 10-fold cross-validation, the training data is randomly partitioned into 10 partitions (or folds). Each partition is used to construct a regression model $\mathbf{Y}_j = f(\mathbf{X}_j, \boldsymbol{\beta}_j)$ for $j = 1, 2, \cdots, 10$. One method for constructing the final model is to average the loading values $\bar{\boldsymbol{\beta}} = (1/k)\sum_{j=1}^{k} \boldsymbol{\beta}_j$, which are then used for the final, cross-validated regression model $\mathbf{Y} = f(\mathbf{X}, \bar{\boldsymbol{\beta}})$. This model is then used on the withhold data to test its extrapolation power, or generalizability. The error on this withhold test set is what determines the efficacy of the model. There are a variety of other methods for selecting the best model, including simply choosing the best of the k-fold models. As for partitioning the data, a common strat-

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

egy is to break the data into 70% training data, 20% validation data, and 10% withheld data. For very large data sets, the validation and withheld data sets can be reduced provided there is enough data to accurately assess the model constructed.

Leave-p-Out Cross-Validation

Another standard technique for cross-validation involves the so-called *leave-p-out cross-validation* (LpO CV). In this case, p samples of the training data are removed from the data and kept as the validation set. A model is built on the remaining training data, and the accuracy of the model is tested on the p withheld samples. This is repeated with a new selection of p samples until all the training data has been part of the validation data set. The accuracy of the model is then evaluated on the withheld data from averaging the accuracy of the models and the loadings produced from the various partitions of the data.

4.7 Model Selection: Information Criteria

There is a different approach to model selection than the cross-validation strategies outlined in the previous section. Indeed, model selection has a rigorous set of mathematical innovations starting from the early 1950s. The Kullback–Leibler (KL) divergence [418] measures the distance between two probability density distributions (or data sets which represent the truth and a model) and is the core of modern information theory criteria for evaluating the viability of a model. The KL divergence has deep mathematical connections to statistical methods characterizing entropy as developed by Ludwig E. Boltzmann (c. 1844–1906), as well as a relation to information theory developed by Claude Shannon [653]. Model selection is a well-developed field with a large body of literature, most of which is exceptionally well reviewed by Burnham and Anderson [142]. In what follows, only brief highlights will be given to demonstrate some of the standard methods.

The KL divergence between two models $f(\mathbf{X}, \boldsymbol{\beta})$ and $g(\mathbf{X}, \boldsymbol{\mu})$ is defined as

$$I(f, g) = \int f(\mathbf{X}, \boldsymbol{\beta}) \log \left[\frac{f(\mathbf{X}, \boldsymbol{\beta})}{g(\mathbf{X}, \boldsymbol{\mu})} \right] \mathrm{d}\mathbf{X}, \tag{4.45}$$

where $\boldsymbol{\beta}$ and $\boldsymbol{\mu}$ are parameterizations of the models $f(\cdot)$ and $g(\cdot)$, respectively. From an information theory perspective, the quantity $I(f, g)$ measures the information lost when g is used to represent f. Note that if $f = g$, then the log term is zero (i.e., $\log(1) = 0$) and $I(f, g) = 0$, so that there is no information lost. In practice, f will represent the *truth*, or measurements of an experiment, while g will be a model proposed to describe f.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

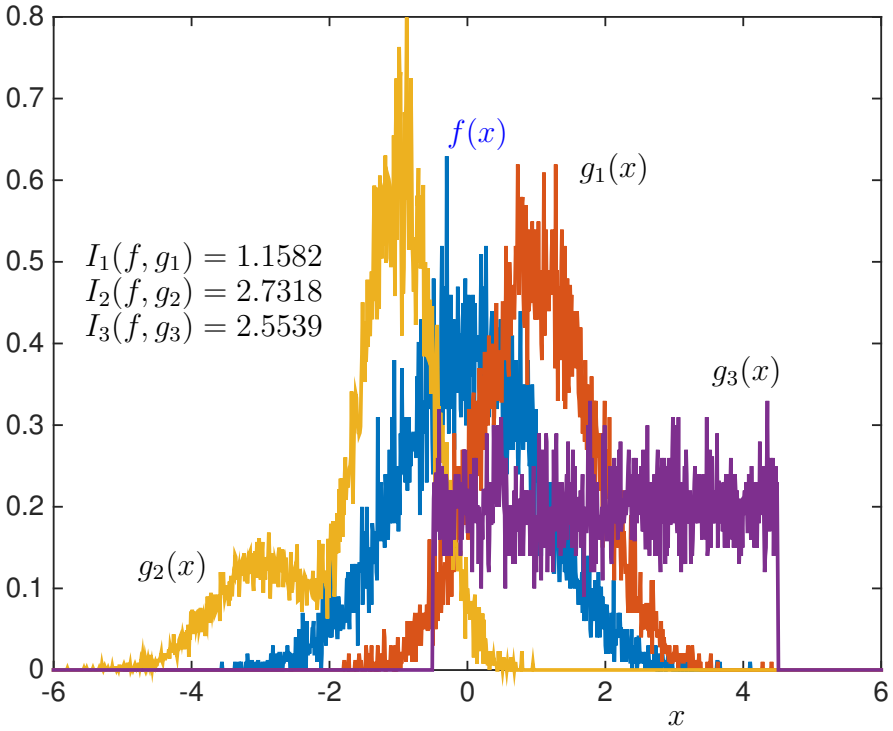


Figure 4.21: Comparison of three models $g_1(x)$, $g_2(x)$, and $g_3(x)$ against the truth model $f(x)$. The KL divergence $I_j(f, g_j)$ for each model is computed, showing that the model $g_1(x)$ is closest to statistically representing the true data.

Unlike the regression and cross-validation performed previously, when computing KL divergence, a model must be specified. Recall that we used cross-validation previously to generate a model using different regression strategies (see Fig. 4.20 for instance). Here a number of models will be posited and the loss of information, or KL divergence, of each model will be computed. The model with the lowest loss of information is generally regarded as the best model. Thus given M proposed models $g_j(\mathbf{X}, \boldsymbol{\mu}_j)$, where $j = 1, 2, \ldots, M$, we can compute $I_j(f, g_j)$ for each model. The correct model, or best model, is the one that minimizes the information loss $\min_j I_j(f, g_j)$.

As a simple example, consider Fig. 4.21, which shows three different models that are compared to the truth data. The computation of the KL divergence score is also illustrated. Note that, in order to avoid division by zero, a constant offset is added to each probability distribution. The truth data generated, $f(x)$, is a simple normally distributed variable. The three models shown are variants of normally and uniformly distributed functions.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Information Criteria: AIC and BIC

This simple example shows the basic ideas behind model selection: compute a distance between a proposed model output $g_j(x)$ and the measured truth $f(x)$. In the early 1970s, Hirotugu Akaike combined Fisher's maximum-likelihood computation [244] with the KL divergence score to produce what is now called the *Akaike information criterion* (AIC) [9]. This was later modified by Gideon Schwarz to the so-called *Bayesian information criterion* (BIC) [646], which provided an information score that was guaranteed to converge to the correct model in the large-data limit, provided the correct model was included in the set of candidate models.

To be more precise, we turn to Akaike's seminal contribution [9]. Akaike was aware that KL divergence cannot be computed in practice since it requires full knowledge of the statistics of the truth model $f(x)$ and of all the parameters in the proposed models $g_j(x)$. Thus, Akaike proposed an alternative way to estimate KL divergence based on the empirical log-likelihood function at its maximum point. This is computable in practice and was a critically enabling insight for rigorous methods of model selection. The technical aspects of Akaike's work connecting log-likelihood estimates and KL divergence [9, 142] was a paradigm shifting mathematical achievement, and thus led to the development of the AIC score

$$AIC = 2K - 2\log[\mathcal{L}(\hat{\boldsymbol{\mu}}|\mathbf{x})], \tag{4.46}$$

where K is the number of parameters used in the model, $\hat{\boldsymbol{\mu}}$ is an estimate of the best parameters used (i.e., lowest KL divergence) in $g(\mathbf{X}, \boldsymbol{\mu})$ computed from a *maximum-likelihood estimate* (MLE), and $\mathbf{x}$ are independent samples of the data to be fit. Thus, instead of a direct measure of the distance between two models, the AIC provides an estimate of the relative distance between the approximating model and the true model or data. As the number of terms gets large in a proposed model, the AIC score increases with slope $2K$, thus providing a penalty for non-parsimonious models. Importantly, due to its relative measure, it will always result in an objective "best" model with the lowest AIC score, but this best model may still be quite poor in prediction and reconstruction of the data.

AIC is one of the standard model selection criteria used today. However, there are others. Highlighted here is the modification of AIC by Schwarz to construct BIC [646]. BIC is almost identical to AIC aside from the penalization of the information criteria by the number of terms. Specifically, BIC is defined as

$$BIC = \log(n)K - 2\log[\mathcal{L}(\hat{\boldsymbol{\mu}}|\mathbf{x})], \tag{4.47}$$

where n is the number of data points, or sample size, considered. This slightly different version of the information criterion has one significant consequence. The seminal contribution of Schwarz was to prove that, if the correct model was

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

included along with a set of candidate models, then it would be theoretically guaranteed to be selected as the best model based upon BIC for a sufficiently large set of data x. This is in contrast to AIC, which, in certain pathological cases, can select the wrong model.

Computing AIC and BIC Scores

MATLAB allows us to directly compute the AIC and/or BIC score from the **aicbic** command. This computational tool is embedded in the econometrics toolbox, and it allows one to evaluate a set of models against one another. The evaluation is made from the log-likelihood estimate of the models under consideration. An arbitrary number of models can be compared.

In the specific example considered here, we consider a ground-truth model constructed from the autoregressive model

$$x_n = -4 + 0.2x_{n-1} + 0.5x_{n-2} + \mathcal{N}(0, 2), \tag{4.48}$$

where x_n is the value of the time series at time t_n and $\mathcal{N}(0, 2)$ is a white-noise process with mean zero and variance two. We fit three autoregressive integrated moving average (ARIMA) models to the data. The three ARIMA models have one, two, and three time delays in their models. The following code computes their log-likelihood and corresponding AIC and BIC scores.

Code 4.6: [MATLAB] Computation of AIC and BIC scores.

```
T = 100; % Sample size
DGP = arima('Constant',-4,'AR',[0.2, 0.5],'Variance',2);
y = simulate(DGP,T);

EstMdl1 = arima('ARLags',1);
EstMdl2 = arima('ARLags',1:2);
EstMdl3 = arima('ARLags',1:3);

logL = zeros(3,1); % Preallocate loglikelihood vector
[~,~,logL(1)] = estimate(EstMdl1,y);%,'print',false);
[~,~,logL(2)] = estimate(EstMdl2,y);%,'print',false);
[~,~,logL(3)] = estimate(EstMdl3,y);%,'print',false);

[aic,bic] = aicbic(logL, [3; 4; 5], T*ones(3,1))
```

Code 4.6: [Python] Computation of AIC and BIC scores.

```
arparams = np.array([-4, .2, 0.5])
maparams = np.array([1])
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

```
arma_process = sm.tsa.arima_process.ArmaProcess(arparams,
   maparams)
y = arma_process.generate_sample(T,scale=2)

logL = np.zeros(3) # log likelihood vector
aic = np.zeros(3)  # AIC vector
bic = np.zeros(3)  # BIC vector

for j in range(2):
    model_res = sm.tsa.arima_model.ARMA(y, (0,0)).fit(trend=
       'c', disp=0,start_ar_lags=j+1,method='mle')
    logL[j] = model_res.llf
    aic[j] = model_res.aic
    bic[j] = model_res.bic
```

Note that the best model, the one with both the lowest AIC and BIC scores, is the second model, which has two time delays. This is expected, as it corresponds to the ground-truth model. The output in this case is given by the following.

```
aic =
  381.7732
  358.2422
  358.8479

bic =
  389.5887
  368.6629
  371.8737
```

The lowest AIC and BIC score is 358.2422 and 368.6629, respectively. Note that, although the correct model was selected, the AIC score provides little distinction between models, especially the two and three time delay models.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Suggested Reading

Texts

(1) Model selection and multimodel inference, by K. P. Burnham and D. R. Anderson [142].

(2) Multivariate analysis, by R. A. Johnson and D. Wichern, 2002 [350].

(3) An introduction to statistical learning, by G. James, D. Witten, T. Hastie, and R. Tibshirani, 2013 [348].

Papers and reviews

(1) On the mathematical foundations of theoretical statistics, by R. A. Fischer, *Philosophical Transactions of the Royal Society of London*, 1922 [244].

(2) A new look at the statistical model identification, by H. Akaike, *IEEE Transactions on Automatic Control*, 1974 [9].

(3) Estimating the dimension of a model, by G. Schwarz et al., *The Annals of Statistics*, 1978 [646].

(4) On information and sufficiency, by S. Kullback and R. A. Leibler, *The Annals of Statistics*, 1951 [418].

(5) A mathematical theory of communication, by C. Shannon, *ACM SIGMOBILE Mobile Computing and Communications Review*, 2001 [646].

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Homework

Exercise 4-1. Derive in closed form the 3×3 matrix which results from a least-squares regression to a parabolic fit $f(x) = Ax^2 + Bx + C$.

Exercise 4-2. Consider the following temperature data taken over a 24-hour (military time) cycle:

75 at 01	77 at 02	76 at 03	73 at 04	69 at 05	68 at 06	63 at 07	59 at 08
57 at 09	55 at 10	54 at 11	52 at 12	50 at 13	50 at 14	49 at 15	49 at 16
49 at 17	50 at 18	54 at 19	56 at 20	59 at 21	63 at 22	67 at 23	72 at 24

Fit the data with the parabolic fit

$$f(x) = Ax^2 + Bx + C \tag{4.49}$$

and calculate the E_2 error. Use both a linear interpolation and spline to generate an interpolated approximation to the data for $x = 1 : 0.01 : 24$.

Develop a least-squares algorithm and calculate E_2 for

$$y = A\cos(Bx) + C. \tag{4.50}$$

Evaluate the resulting fit as a function of the initial guess for the values of A, B, and C.

Exercise 4-3. For the temperature data of the previous example, consider a polynomial fit of the form

$$f(x) = \sum_{k=0}^{10} \alpha_k x^k, \tag{4.51}$$

where the loadings α_k are to be determined by four regression techniques: least-squares, LASSO, ridge, and elastic net. Compare the models for each against each other.

Randomly pick any time point and corrupt the temperature measurement at that location. For instance, the temperature reading at that location could be zero. Investigate the resulting model and E_2 error for the four regression techniques considered. Identify the models that are robust to such an outlier and those that are not. Explicitly calculate the variance of the loading coefficients α_k for each method for a number of random trials with one or more corrupt data points.

Exercise 4-4. Download the MNIST data set (both training and test sets and labels) from `http://yann.lecun.com/exdb/mnist/`.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

The labels will tell you which digit it is: $1, 2, 3, 4, 5, 6, 7, 8, 9, 0$. Let each output be denoted by the vector $\mathbf{y}_j$.

$$\text{“1”} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}, \quad \text{“2”} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}, \quad \ldots, \quad \text{“9”} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \\ 0 \end{bmatrix}, \quad \text{“0”} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}. \tag{4.52}$$

Now let $\mathbf{B}$ be the set of output vectors

$$\mathbf{B} = [\mathbf{y}_1 \quad \mathbf{y}_2 \quad \mathbf{y}_3 \quad \ldots \quad \mathbf{y}_n] \tag{4.53}$$

and let the matrix $\mathbf{A}$ be the corresponding reshaped (vectorized) MNIST images

$$\mathbf{A} = [\mathbf{x}_1 \quad \mathbf{x}_2 \quad \mathbf{x}_3 \quad \ldots \quad \mathbf{x}_n]. \tag{4.54}$$

Thus each vector $\mathbf{x}_j \in \mathbb{R}^{n^2}$ is a vector reshaped from the $n \times n$ image.

Using various $\mathbf{AX} = \mathbf{B}$ solvers, determine a mapping from the image space to the label space.

By promoting sparsity, determine and rank which pixels in the MNIST set are most informative for correctly labeling the digits. (You will have to come up with your own heuristics or empirical rules for this. Be sure to visualize the results from $\mathbf{X}$.) Apply your most important pixels to the test data set to see how accurate you are with as few pixels as possible. Redo the analysis with each digit individually to find the most important pixels for each digit. Think about the interpretation of what you are doing with this $\mathbf{AX} = \mathbf{B}$ problem.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Chapter 5

Clustering and Classification

Machine learning is based upon optimization techniques for data. The goal is to find both a low-rank subspace for optimally embedding the data, as well as regression methods for clustering and classification of different data types. Machine learning thus provides a principled set of mathematical methods for extracting meaningful features from data, i.e., data mining, as well as binning the data into distinct and meaningful patterns that can be exploited for decision making. Specifically, it learns from and makes predictions based on data. For business applications, this is often called *predictive analytics*, and it is at the forefront of modern data-driven decision making. In an integrated system, such as is found in autonomous robotics, various machine learning components (e.g., for processing visual and tactile stimuli) can be integrated to form what we now call *artificial intelligence* (AI). To be explicit: AI is built upon integrated machine learning algorithms, which in turn are fundamentally rooted in optimization.

There are two broad categories for machine learning: *supervised machine learning* and *unsupervised machine learning*. In the former, the algorithm is presented with labeled data sets. The training data, as outlined in the cross-validation method of the last chapter, is labeled by a teacher/expert. Thus examples of the input and output of a desired model are explicitly given, and regression methods are used to find the best model for the given labeled data, via optimization. This model is then used for prediction and classification using new data. There are important variants of supervised methods, including *semi-supervised learning* in which incomplete training is given so that some of the input/output relationships are missing, i.e., for some input data, the actual output is missing. *Active learning* is another common subclass of supervised methods whereby the algorithm can only obtain training labels for a limited set of instances, based on a budget, and also has to optimize its choice of objects for which to acquire labels. In an interactive framework, these can be presented to the user for labeling. Finally, in *reinforcement learning*, rewards or punishments are the training labels that help shape the regression architecture in order to build the best model. In contrast, no labels are given for *unsupervised learning* algorithms.

Thus, they must find patterns in the data in a principled way in order to determine how to cluster data and generate labels for predicting and classifying new data. In unsupervised learning, the goal itself may be to discover patterns in the data embedded in the low-rank subspaces so that *feature engineering* or *feature extraction* can be used to build an appropriate model.

In this chapter, we will consider some of the most commonly used supervised and unsupervised machine learning methods. As will be seen, our goal is to highlight how data mining can produce important data features (feature engineering) for later use in model building. We will also show that the machine learning methods can be broadly used for clustering and classification, as well as for building regression models for prediction. Critical to all of this machine learning architecture is finding low-rank feature spaces that are informative and interpretable.

5.1 Feature Selection and Data Mining

To exploit data for diagnostics, prediction, and control, dominant features of the data must be extracted. In the opening chapter of this book, singular value decomposition (SVD) and principal component analysis (PCA) were introduced as methods for determining the dominant correlated structures contained within a data set. In the eigenfaces example of Section 1.6, for instance, the dominant features of a large number of cropped face images were shown. These eigenfaces, which are ordered by their ability to account for commonality (correlation) across the database of faces, were guaranteed to give the best set of r features for reconstructing a given face in an ℓ_2 sense with a rank-r truncation. The eigenface modes gave clear and interpretable features for identifying faces, including highlighting the eyes, nose, and mouth regions, as might be expected. Importantly, instead of working with the high-dimensional measurement space, the feature space allows one to consider a significantly reduced subspace where diagnostics can be performed.

The goal of data mining and machine learning is *to construct and exploit the intrinsic low-rank feature space of a given data set*. The feature space can be found in an unsupervised fashion by an algorithm, or it can be explicitly constructed by expert knowledge and/or correlations among the data. For eigenfaces, the features are the PCA modes generated by the SVD. Thus each PCA mode is high-dimensional, but the only quantity of importance in feature space is the weight of that particular mode in representing a given face. If one performs an r-rank truncation, then any face needs only r features to represent it in feature space. This ultimately gives a low-rank embedding of the data in an interpretable set of r features that can be leveraged for diagnostics, prediction, reconstruction, and/or control.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

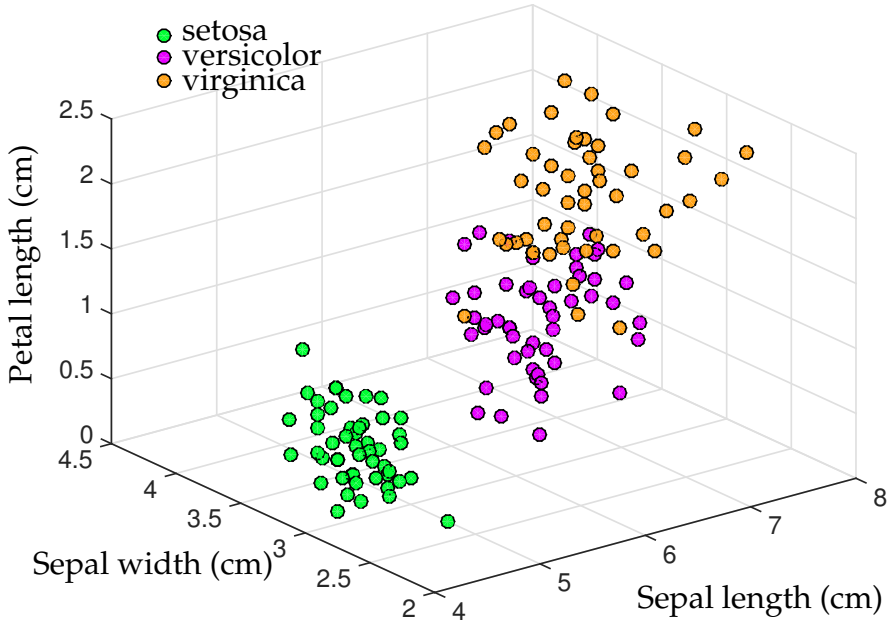


Figure 5.1: Fisher iris data set with 150 measurements over three varieties, including 50 measurements each of *Iris setosa*, *I. versicolor*, and *I. virginica*. Each flower includes a measurement of sepal length, sepal width, petal length, and petal width. The first three of these are illustrated here, showing that these simple biological features are sufficient to show that the data has distinct, quantifiable differences between the species.

Several examples will be developed that illustrate how to generate a feature space, starting with a standard data set included with MATLAB. The Fisher iris data set includes measurements of 150 irises of three varieties: *Iris setosa*, *I. versicolor*, and *I. virginica*. The 50 samples of each flower include measurements in centimeters of the sepal length, sepal width, petal length, and petal width. For this data set, the four features are already defined in terms of interpretable properties of the biology of the plants. For visualization purposes, Fig. 5.1 considers only the first three of these features. The following code accesses the Fisher iris data set:

Code 5.1: [MATLAB] Features of the Fisher irises.

```
load fisheriris;
x1=meas(1:50,:);     % setosa
x2=meas(51:100,:);   % versicolor
x3=meas(101:150,:); % virginica
```

Code 5.1: [Python] Features of the Fisher irises.

```
fisheriris_mat = io.loadmat(os.path.join('..','DATA','
    fisheriris.mat'))
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.



Figure 5.2: Example images of dogs (left) and cats (right). Our goal is to construct a feature space where automated classification of these images can be efficiently computed.

```
meas = fisheriris_mat['meas']
x1 = meas[:50,:] # setosa
x2 = meas[50:100,:] # versicolor
x3 = meas[100:,:] # virginica
```

Figure 5.1 shows that the properties measured can be used as a good set of features for clustering and classification purposes. Specifically, the three iris varieties are well separated in this feature space. The setosa is most distinctive in its feature profile, while the versicolor and virginica have a small overlap among the samples taken. For this data set, machine learning is certainly not required to generate a good classification scheme. However, data generally does not so readily reduce down to simple two- and three-dimensional visual cues. Rather, decisions about clustering in feature space occur with many more variables, thus requiring the aid of computational methods to provide good classification schemes.

As a second example, we consider in Fig. 5.2 a selection from an image database of 80 dogs and 80 cats. A specific goal for this data set is to develop an automated classification method whereby the computer can distinguish between cats and dogs. In this case, the data for each cat and dog is the 64×64 pixel space of the image. Thus each image has 4096 measurements, in contrast to the four measurements for each example in the iris data set. Like eigenfaces, we will use the SVD to extract the dominant correlations among the images. The following code loads the data and performs a singular value decomposition on the data after the mean is subtracted. The SVD produces an ordered set of modes characterizing the correlation between all the dog and cat images. Figure 5.3 shows the first four SVD modes of the 160 images (80 dogs and 80 cats).

Code 5.2: [MATLAB] Features of dogs and cats.

```
load dogData.mat
load catData.mat
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

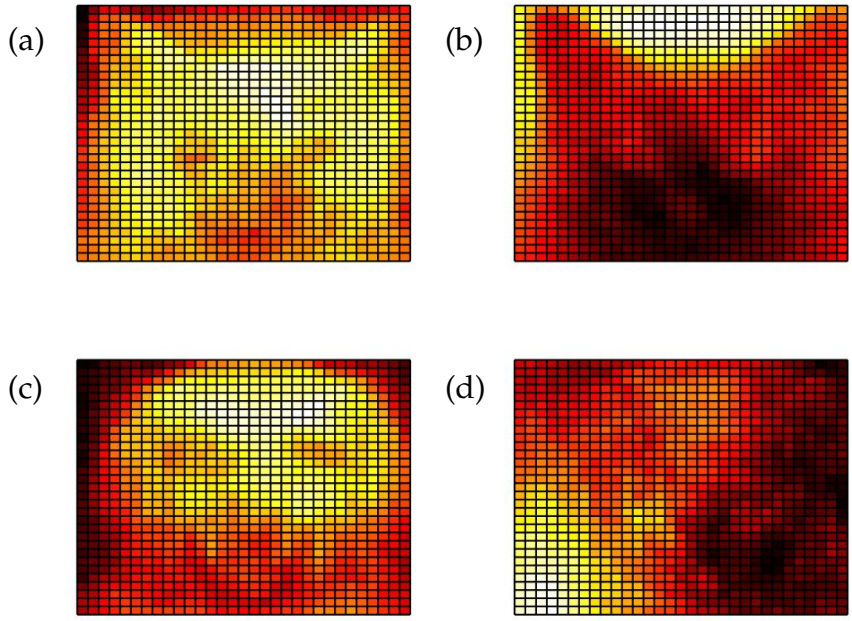


Figure 5.3: First four features (a)–(d) generated from the SVD of the 160 images of dogs and cats, i.e., these are the first four columns of the U matrix of the SVD. Typical cat and dog images are shown in Fig. 5.2. Note that the first two modes (a) and (b) show that the triangular ears are important features when images are correlated. This is certainly a distinguishing feature for cats, while dogs tend to lack this feature. Thus, in feature space, cats generally add these two dominant modes to promote this feature, while dogs tend to subtract these features to remove the triangular ears from their representation.

```
CD=double([dog cat]);
[u,s,v]=svd(CD-mean(CD(:)),'econ');
```

Code 5.2: [Python] Features of dogs and cats.

```
dog = dogdata_mat['dog']
cat = catdata_mat['cat']
CD = np.concatenate((dog,cat),axis=1)
u,s,vT = np.linalg.svd(CD-np.mean(CD),full_matrices=0)
```

The original image space, or pixel space, is only one potential set of data to work with. The data can be transformed into a wavelet representation where edges of the images are emphasized. The following code loads the images in their wavelet representation and computes a new low-rank embedding space.

Code 5.3: [MATLAB] Wavelet features of dogs and cats.

```
load catData_w.mat
load dogData_w.mat
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

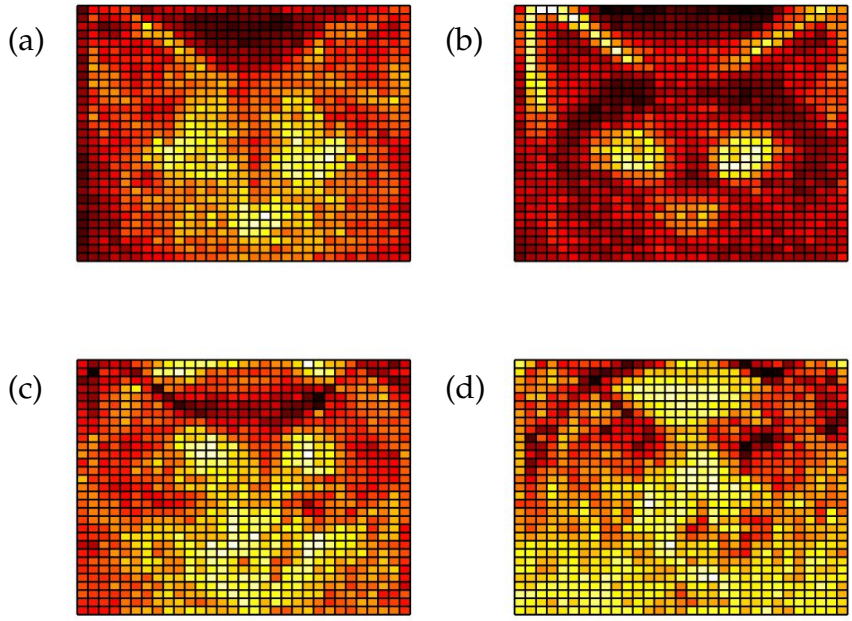


Figure 5.4: First four features (a)–(d) generated from the SVD of the 160 images of dogs and cats in the wavelet domain. As before, the first two modes (a) and (b) show that the triangular ears are important. This is an alternative representation of the dogs and cats that can help better classify dogs versus cats.

```
CD2=[dog_wave cat_wave];
[u2,s2,v2]=svd(CD2-mean(CD2(:)),'econ');
```

Code 5.3: [Python] Wavelet features of dogs and cats.

```
dog_wave = dogdata_w_mat['dog_wave']
cat_wave = catdata_w_mat['cat_wave']
CD2 = np.concatenate((dog_wave,cat_wave),axis=1)
u2,s2,vT2 = np.linalg.svd(CD2-np.mean(CD2),full_matrices=0)
```

The equivalent of Fig. 5.3 in wavelet space is shown in Fig. 5.4. Note that the wavelet representation helps emphasize many key features such as the eyes, nose, and ears, potentially making it easier to make a classification decision. Generating a feature space that enables classification is critical for constructing effective machine learning algorithms.

Whether using the image space directly or a wavelet representation, Figs. 5.3 and 5.4, respectively, the goal is to project the data onto the feature space generated by each. A good feature space helps find distinguishing features that allow one to perform a variety of tasks that may include clustering, classification, and prediction. The importance of each feature to an individual image is given by the $\mathbf{V}$ matrix in the SVD. Specifically, each column of $\mathbf{V}$ determines the loading, or weighting, of each feature onto a specific image. Histograms of

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

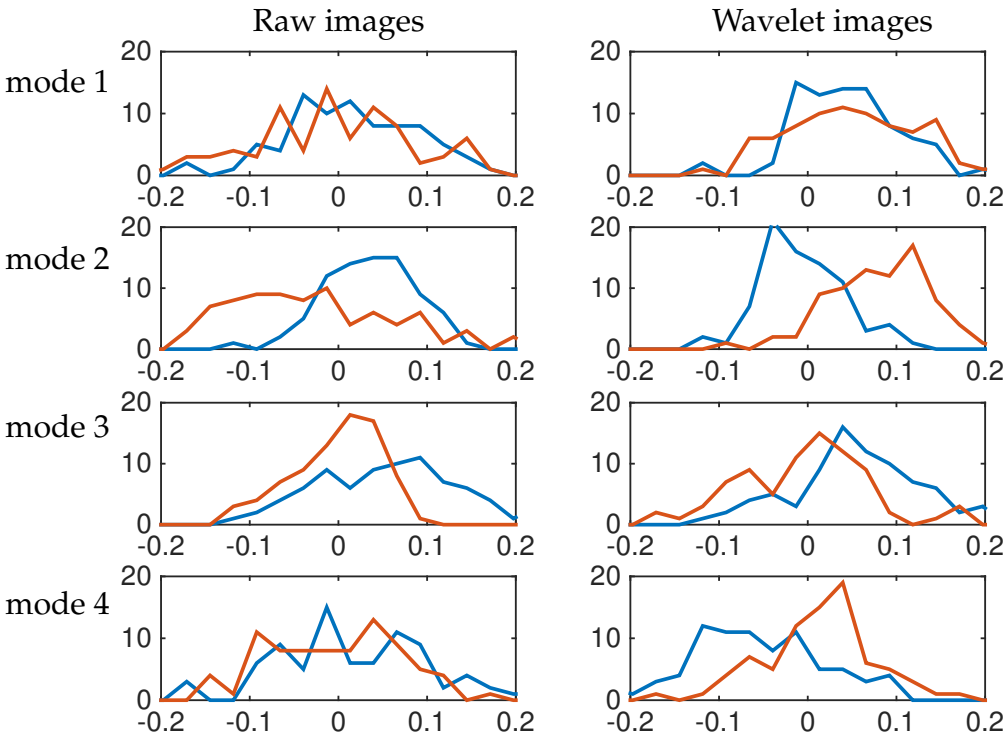


Figure 5.5: Histogram of the distribution of loadings for dogs (blue) and cats (red) on the first four dominant SVD modes. The left panels show the distributions for the raw images (see Fig. 5.3) while the right panels show the distribution for wavelet-transformed data (see Fig. 5.4). The loadings come from the columns of the $\mathbf{V}$ matrix of the SVD. Note the good separability between dogs and cats using the second mode.

these loadings can then be used to visualize how distinguishable cats and dogs are from each other by each feature (see Fig. 5.5). The following code produces a histogram of the distribution of loadings for the dogs and the cats (first 80 images versus second 80 images, respectively).

Code 5.4: [MATLAB] Feature histograms of dogs and cats.

```
xbin=linspace(-0.25,0.25,20);
for j=1:4
    subplot(4,2,2*j-1)
    pdf1=hist(v(1:80,j),xbin)
    pdf2=hist(v(81:160,j),xbin)
    plot(xbin,pdf1,xbin,pdf2,'Linewidth',[2])
end
```

Code 5.4: [Python] Feature histograms of dogs and cats.

```
    pdf1 = np.histogram(vT[j,:80],bins=xbin_edges)[0]
    pdf2 = np.histogram(vT[j,80:],bins=xbin_edges)[0]
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

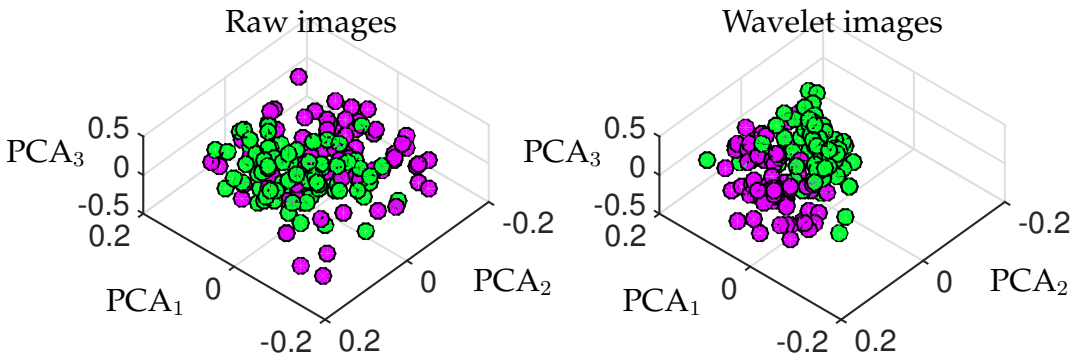


Figure 5.6: Projection of dogs (green) and cats (magenta) into feature space. Note that the raw images and their wavelet counterparts produce different embeddings of the data. Both exhibit clustering around their labeled states of dogs and cats. This is exploited in the learning algorithms that follow. The wavelet images are especially good for clustering and classification, as this feature space more easily separates the data.

Figure 5.5 shows the distribution of loading scores for the first four modes for both the raw images as well as the wavelet-transformed images. For both the sets of images, the distribution of loadings on the second mode clearly shows a strong separability between dogs and cats. The wavelet-processed images also show a nice separability on the fourth mode. Note that the first mode for both shows very little discrimination between the distributions and is thus not useful for classification and clustering objectives.

Features that provide strong separability between different types of data (e.g., dogs and cats) are typically exploited for machine learning tasks. This simple example shows that feature engineering is a process whereby an initial data exploration is used to help identify potential pre-processing methods. These features can then help the computer identify highly distinguishable features in a higher-dimensional space for accurate clustering, classification, and prediction. As a final note, consider Fig. 5.6, which projects the dogs and cats data onto the first three PCA modes (SVD modes) discovered from the raw images or their wavelet-transformed counterparts. As will be seen later, the wavelet-transformed images provide a higher degree of separability, and thus improved classification.

5.2 Supervised versus Unsupervised Learning

As previously stated, the goal of data mining and machine learning is to construct and exploit the intrinsic low-rank feature space of a given data set. Good feature engineering and feature extraction algorithms can then be used to learn classifiers and predictors for the data. Two dominant paradigms exist for learn-

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

ing from data: *supervised methods* and *unsupervised methods*. Supervised data-mining algorithms are presented with labeled data sets, where the training data is labeled by a teacher/expert/supervisor. Thus examples of the input and output of a desired model are explicitly given, and regression methods are used to find the best model via optimization for the given labeled data. This model is then used for prediction and classification using new data. There are important variants of this basic architecture which include semi-supervised learning, active learning, and reinforcement learning. For unsupervised learning algorithms, no training labels are given, so that an algorithm must find patterns in the data in a principled way in order to determine how to cluster and classify new data. In unsupervised learning, the goal itself may be to discover patterns in the data embedded in the low-rank subspaces so that feature engineering or feature extraction can be used to build an appropriate model.

To illustrate the difference in supervised versus unsupervised learning, consider Fig. 5.7. This shows a scatter plot of two Gaussian distributions. In one case, the data are well separated so that their means are sufficiently far apart and two distinct clusters are observed. In the second case, the two distributions are brought close together so that separating the data is a challenging task. The goal of unsupervised learning is to discover clusters in the data. This is a trivial task by visual inspection, provided the two distributions are sufficiently separated. Otherwise, it becomes very difficult to distinguish clusters in the data. Supervised learning provides labels for some of the data. In this case, points are labeled with either green dots or magenta dots and the task is to classify the unlabeled data (grey dots) as either green or magenta. Much like the unsupervised architecture, if the statistical distributions that produced the data are well separated, then using the labels in combination with the data provides a simple way to classify all the unlabeled data points. Supervised algorithms also perform poorly if the data distributions have significant overlap.

Supervised and unsupervised learning can be stated mathematically. Let

$$\mathcal{D} \subset \mathbb{R}^n, \tag{5.1}$$

so that $\mathcal{D}$ is an open bounded set of dimension n. Further, let

$$\mathcal{D}' \subset \mathcal{D}. \tag{5.2}$$

The goal of classification is to build a classifier labeling all data in $\mathcal{D}$ given data from $\mathcal{D}'$.

To make our problem statement more precise, consider a set of data points $\mathbf{x}_j \in \mathbb{R}^n$ and labels $\mathbf{y}_j$ for each point, where $j = 1, 2, \ldots, m$. Labels for the data can come in many forms, from numeric values, including integer labels, to text strings. For simplicity, we will label the data in a binary way as either plus one or minus one, so that $\mathbf{y}_j \in \{\pm 1\}$.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

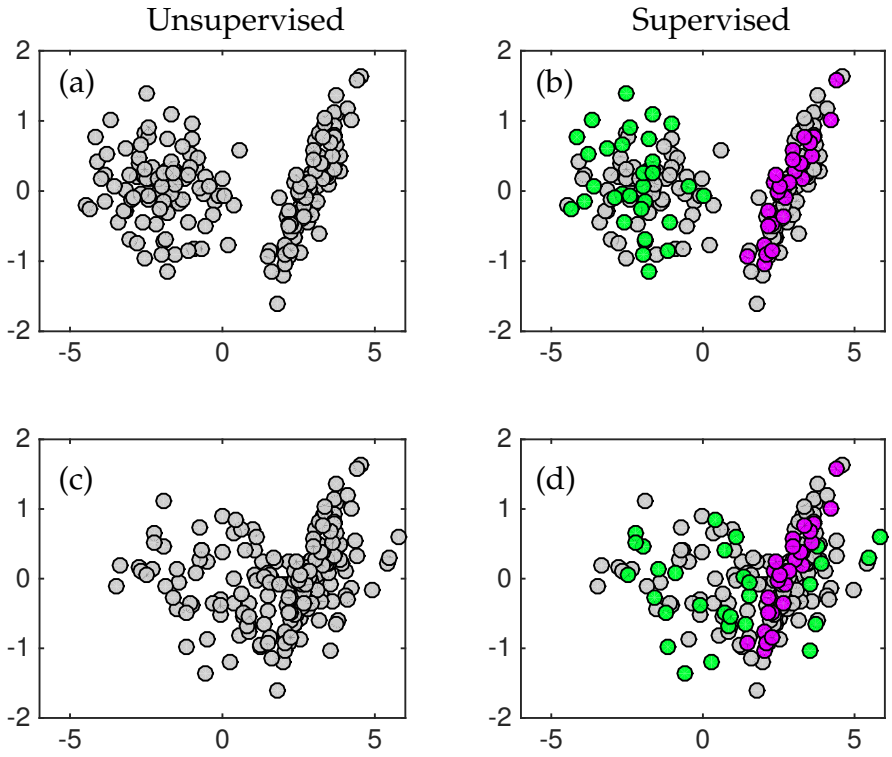


Figure 5.7: Illustration of unsupervised versus supervised learning. In panels (a) and (c), unsupervised learning attempts to find clusters for the data in order to classify them into two groups. For well-separated data (a), the task is straightforward and labels can easily be produced. For overlapping data (c), it is a very difficult task for an unsupervised algorithm to accomplish. In panels (b) and (d), supervised learning provides a number of labels: green balls and magenta balls. The remaining unlabeled data is then classified as green or magenta. For well-separated data (b), labeling data is easy, while overlapping data presents significant challenge.

For unsupervised learning, the following inputs and outputs are then associated with learning a classification task:

Input

$$\text{data} \quad \{\mathbf{x}_j \in \mathbb{R}^n,\ j \in Z := \{1, 2, \ldots, m\}\}, \tag{5.3a}$$

Output

$$\text{labels} \quad \{\mathbf{y}_j \in \{\pm 1\},\ j \in Z\}. \tag{5.3b}$$

Thus the mathematical framing of unsupervised learning is focused on producing labels $\mathbf{y}_j$ for all the data. Generally, the data $\mathbf{x}_j$ used for training the classifier is from $\mathcal{D}'$. The classifier is then more broadly applied, i.e., it generalizes, to the open bounded domain $\mathcal{D}$. If the data used to build a classifier only samples a small portion of the larger domain, then it is often the case that the classifier will not generalize well.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Supervised learning provides labels for the training stage. The inputs and outputs for this learning classification task can be stated as follows:

Input

$$\text{data} \quad \{\mathbf{x}_j \in \mathbb{R}^n,\ j \in Z := \{1, 2, \ldots, m\}\}, \tag{5.4a}$$

$$\text{labels} \quad \{\mathbf{y}_j \in \{\pm 1\},\ j \in Z' \subset Z\}, \tag{5.4b}$$

Output

$$\text{labels} \quad \{\mathbf{y}_j \in \{\pm 1\},\ j \in Z\}. \tag{5.4c}$$

In this case, a subset of the data is labeled and the missing labels are provided for the remaining data. Technically speaking, this is a semi-supervised learning task, since some of the training labels are missing. For supervised learning, all the labels are known in order to build the classifier on $\mathcal{D}'$. The classifier is then applied to $\mathcal{D}$. As with unsupervised learning, if the data used to build a classifier only samples a small portion of the larger domain, then it is often the case that the classifier will not generalize well.

For the data sets considered in our feature selection and data-mining section, we can consider in more detail the key components required to build a classification model: $\mathbf{x}_j$, $\mathbf{y}_j$, $\mathcal{D}$, and $\mathcal{D}'$. The Fisher iris data of Fig. 5.1 is a classic example for which we can detail these quantities. We begin with the data collected:

$$\mathbf{x}_j = \{\text{sepal length, sepal width, petal length, petal width}\}. \tag{5.5}$$

Thus each iris measurement contains four data fields, or features, for our analysis. The labels can be one of the following:

$$\mathbf{y}_j = \{\text{setosa, versicolor, virginica}\}. \tag{5.6}$$

In this case the labels are text strings, and there are three of them. Note that, in our formulation of supervised and unsupervised learning, there were only two outputs (binary), which were labeled either ± 1. Generally, there can be many labels, and they are often text strings. Finally, there is the domain of the data. For this case,

$$\mathcal{D}' \in \{\text{150 iris samples: 50 setosa, 50 versicolor, and 50 virginica}\} \tag{5.7}$$

and

$$\mathcal{D} \in \{\text{the universe of setosa, versicolor, and virginica irises}\}. \tag{5.8}$$

We can similarly assess the dogs and cats data as follows:

$$\mathbf{x}_j = \{64 \times 64 \text{ image} = 4096 \text{ pixels}\}, \tag{5.9}$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

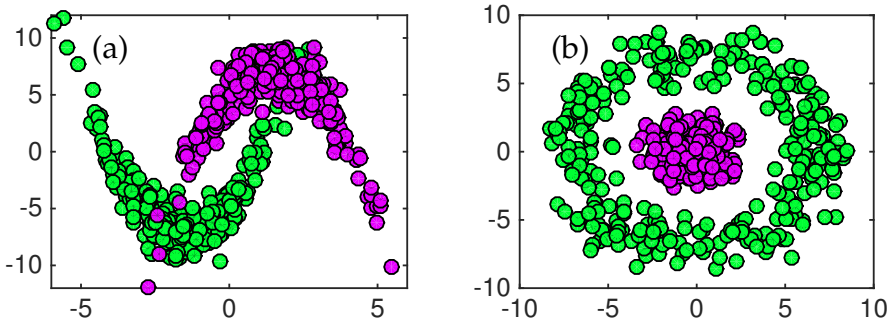


Figure 5.8: Classification and regression models for data can be difficult when the data have nonlinear functions which separate them. In this case, the function separating the green and magenta balls can be difficult to extract. Moreover, if only a small sample of the data $\mathcal{D}'$ is available, then a generalizable model may be impossible to construct for $\mathcal{D}$. The data set in panel (a) represents two half-moon shapes that are just superimposed, while the concentric rings in panel (b) require a circle as a separation boundary between the data. Both are challenging to produce.

where each dog and cat is labeled as

$$\mathbf{y}_j = \{\text{dog, cat}\} = \{1, -1\}. \tag{5.10}$$

In this case the labels are text strings, which can also be translated to numeric values. This is consistent with our formulation of supervised and unsupervised learning, where there are only two outputs (binary), labeled either ± 1. Finally, there is the domain of the data, which is

$$\mathcal{D}' \in \{\text{160 image samples: 80 dogs and 80 cats}\} \tag{5.11}$$

and

$$\mathcal{D} \in \{\text{the universe of dogs and cats}\}. \tag{5.12}$$

Supervised and unsupervised learning methods aim to create algorithms for classification, clustering, or regression. The discussion above is a general strategy for classification. The previous chapter discusses regression architectures. For both tasks, the goal is to build a model from data on $\mathcal{D}'$ that can generalize to $\mathcal{D}$. As already shown in the preceding chapter on regression, generalization can be very difficult, and cross-validation strategies are critical. Deep neural networks, which are state-of-the-art machine learning algorithms for regression and classification, often have difficulty generalizing. Creating strong generalization schemes is at the forefront of machine learning research.

Some of the difficulties in generalization can be illustrated in Fig. 5.8. These data sets, although easily classified and clustered through visual inspection, can be difficult for many regression and classification schemes. Essentially, the boundary between the data forms a nonlinear manifold that is often difficult

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

to characterize. Moreover, if the sampling data $\mathcal{D}'$ only captures a portion of the manifold, then a classification or regression model will almost surely fail in characterizing $\mathcal{D}$. These are also only two-dimensional depictions of a classification problem. It is not difficult to imagine how complicated such data embeddings can be in higher-dimensional space. Visualization in such cases is essentially impossible and one must rely on algorithms to extract the meaningful boundaries separating data. What follows in this chapter and the next are methods for classification and regression given data on $\mathcal{D}'$ that may or may not be labeled. There is quite a diversity of mathematical methods available for performing such tasks.

5.3 Unsupervised Learning: k-Means Clustering

A variety of supervised and unsupervised algorithms will be highlighted in this chapter. We will start with one of the most prominent unsupervised algorithms in use today: k-means clustering. The k-means algorithm assumes one is given a set of vector-valued data with the goal of partitioning m observations into k clusters. Each observation is labeled as belonging to a cluster with the nearest mean, which serves as a proxy (prototype) for that cluster. This results in a partitioning of the data space into Voronoi cells.

Although the number of observations and the dimension of the system are known, the number of partitions k is generally unknown and must also be determined. Alternatively, the user simply chooses a number of clusters to extract from the data. The k-means algorithm is iterative, first assuming initial values for the mean of each cluster and then updating the means until the algorithm has converged. Figure 5.9 depicts the update rule of the k-means algorithm. The algorithm proceeds as follows: (i) Given initial values for k distinct means, compute the distance of each observation $\mathbf{x}_j$ to each of the k means. (ii) Label each observation as belonging to the nearest mean. (iii) Once labeling is completed, find the *center-of-mass* (mean) for each group of labeled points. These new means are then used to start back at step (i) in the algorithm. This is a heuristic algorithm that was first proposed by Stuart Lloyd in 1957 [452], although it was not published until 1982.

The k-means objective can be stated formally in terms of an optimization problem. Specifically, the following minimization describes this process:

$$\underset{\boldsymbol{\mu}_j}{\operatorname{argmin}} \sum_{j=1}^{k} \sum_{\mathbf{x}_n \in \mathcal{D}'_j} \|\mathbf{x}_n - \boldsymbol{\mu}_j\|^2, \tag{5.13}$$

where $\boldsymbol{\mu}_j$ denotes the mean of the jth cluster and $\mathcal{D}'_j$ denotes the subdomain of data associated with that cluster. This minimizes the within-cluster sum of

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

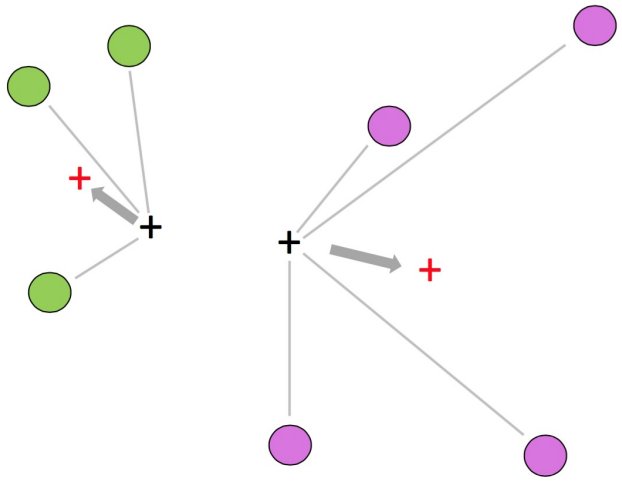


Figure 5.9: Illustration of the k-means algorithm for $k = 2$. Two initial starting values of the mean are given (black +). Each point is labeled as belonging to one of the two means. The green balls are thus labeled as part of the cluster with the left + and the magenta balls are labeled as part of the right +. Once labeled, the mean of the two clusters is recomputed (red +). The process is repeated until the means converge.

squares. In general, solving the optimization problem as stated is NP-hard, making it computationally intractable. However, there a number of heuristic algorithms that provide good performance despite not having a guarantee that they will converge to the globally optimal solution.

Cross-validation of the k-means algorithm, as well as any machine learning algorithm, is critical for determining its effectiveness. Without labels, the cross-validation procedure is more nuanced, as there is no ground truth to compare with. The cross-validation methods of the last section, however, can still be used to test the robustness of the classifier to different sub-selections of the data through k-fold cross-validation. The following portions of code generate Lloyd's algorithm for k-means clustering. We first consider making two clusters of data and partitioning the data into a training set and a test set.

Code 5.5: [MATLAB] The Lloyd algorithm for k-means.

```
g1=[-1 0]; g2=[1 0]; %  Initial guess
for j=1:4
    class1=[]; class2=[];
    for jj=1:length(Y)
        d1=norm(g1-Y(jj,:));
        d2=norm(g2-Y(jj,:));
        if d1<d2
          class1=[class1; [Y(jj,1) Y(jj,2)]];
        else
          class2=[class2; [Y(jj,1) Y(jj,2)]];
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

```
        end
    end
    g1=[mean(class1(1:end,1)) mean(class1(1:end,2))];
    g2=[mean(class2(1:end,1)) mean(class2(1:end,2))];
end
```

Code 5.5: [Python] The Lloyd algorithm for k-means.

```
g1=np.array([-1,0]); g2=np.array([1,0]) # Initial guess
for j in range(4):
    class1 = np.zeros((1,2))
    class2 = np.zeros((1,2))
    for jj in range(Y.shape[0]):
        d1 = np.linalg.norm(g1-Y[jj,:],ord=2)
        d2 = np.linalg.norm(g2-Y[jj,:],ord=2)
        if d1<d2:
            class1 = np.append(class1,Y[jj,:].reshape((1,2))
                ,axis=0)
        else:
            class2 = np.append(class2,Y[jj,:].reshape((1,2))
                ,axis=0)
    class1=np.delete(class1,(0),axis=0) # remove initial
    class2=np.delete(class2,(0),axis=0)
```

Figures 5.10 and 5.11 show the data generated from two distinct Gaussian distributions. In this case, we have ground-truth data to check the k-means clustering against. In general, this is not the case. The Lloyd algorithm guesses the number of clusters and the initial cluster means, and then proceeds to update them in an iterative fashion. Thus, k-means is sensitive to the initial guess and many modern versions of the algorithm also provide principled strategies for initialization.

Figure 5.10 shows the iterative procedure of the k-means clustering. The two initial guesses are used to initially label all the data points (Fig. 5.10(a)). New means are computed and the data relabeled. After only four iterations, the clusters converge. This algorithm was explicitly developed here to show how the iteration procedure rapidly provides an unsupervised labeling of all of the data. MATLAB has a built-in k-means algorithm that only requires a data matrix and the number of clusters desired. It is simple to use and provides a valuable diagnostic tool for data. The following code uses the MATLAB command **kmeans** and also extracts the *decision line* generated from the algorithm separating the two clusters.

Code 5.6: [MATLAB] k-means using MATLAB.

```
[ind,c]=kmeans(Y,2);
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

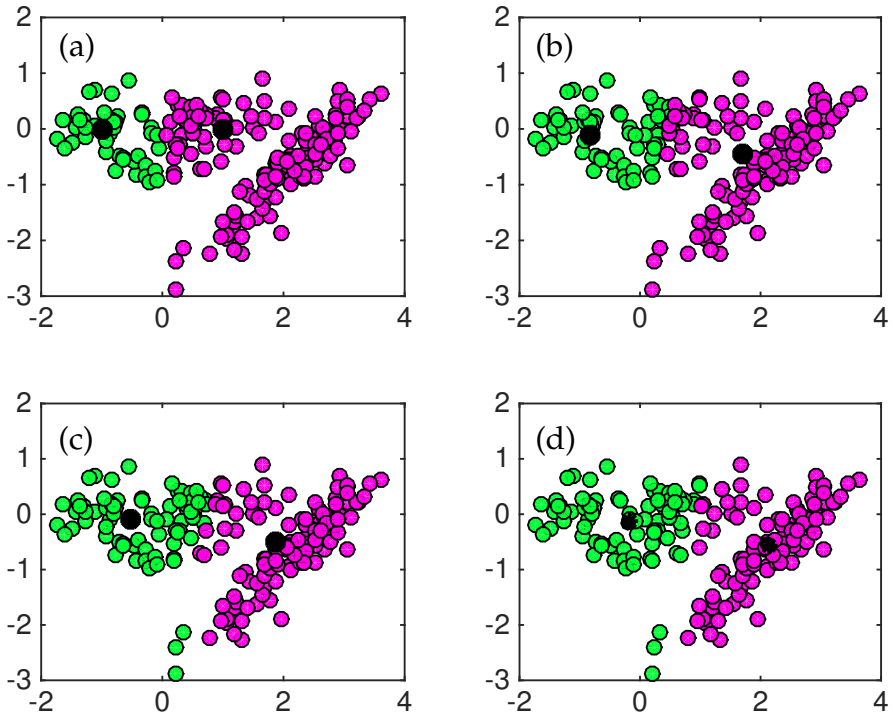


Figure 5.10: Illustration of the k-means iteration procedure based upon Lloyd's algorithm [452]. Two clusters are sought so that $k = 2$. The initial guesses (black circles in panel (a)) are used to initially label all the data according to their distance from each initial guess for the mean. The means are then updated by computing the means of the newly labeled data. This two-stage heuristic converges after approximately four iterations.

Code 5.6: [Python] k-means using Python.

```
kmeans = KMeans(n_clusters=2, random_state=0).fit(Y)
```

Figure 5.11 shows the results of the k-means algorithm and depicts the decision line separating the data into two clusters. The green and magenta balls denote the true labels of the data, showing that the k-means line does not correctly extract the labels. Indeed, a supervised algorithm is more proficient in extracting the ground-truth results, as will be shown later in this chapter. Regardless, the algorithm does get a majority of the data labeled correctly.

The success of k-means is based on two factors: (i) no supervision is required, and (ii) it is a fast heuristic algorithm. The example here shows that the method is not very accurate, but this is often the case in unsupervised methods, as the algorithm has limited knowledge of the data. Cross-validation efforts, such as k-fold cross-validation, can help improve the model and make the unsupervised learning more accurate, but it will generally be less accurate than a supervised algorithm that has labeled data.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

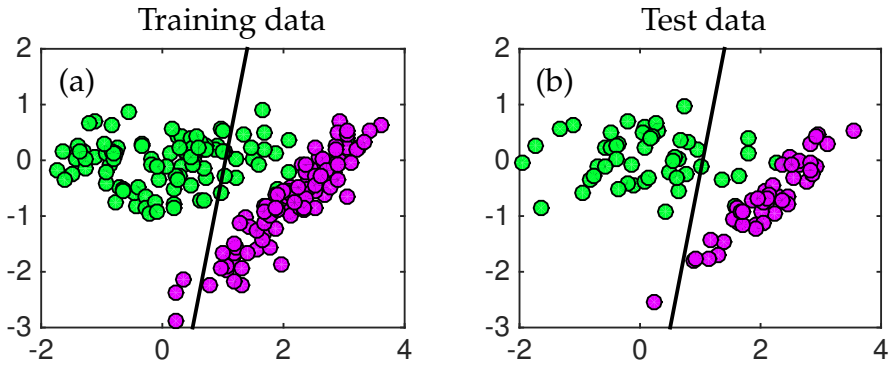


Figure 5.11: The k-means clustering of the data using MATLAB's **kmeans** command. Only the data and number of clusters need be specified. (a) The training data is used to produce a decision line (black line) separating the clusters. Note that the line is clearly not optimal. The classification line can then be used on withheld data to test the accuracy of the algorithm. For the test data, one magenta ball (of 50) would be mislabeled, while six green balls (of 50) are mislabeled.

5.4 Unsupervised Hierarchical Clustering: Dendrogram

Another commonly used unsupervised algorithm for clustering data is a *dendrogram*. Like k-means clustering, dendrograms are created from a simple hierarchical algorithm, allowing one to efficiently visualize if data is clustered without any labeling or supervision. This hierarchical approach will be applied to the data illustrated in Fig. 5.12, where a ground truth is known. Hierarchical clustering methods are generated from either a top-down or a bottom-up approach. Specifically, they are one of two types:

Agglomerative. Each data point $\mathbf{x}_j$ is its own cluster initially. The data is merged in pairs as one creates a hierarchy of clusters. The merging of data eventually stops once all the data has been merged into a single über cluster. This is the bottom-up approach in hierarchical clustering.

Divisive. In this case, all the observations $\mathbf{x}_j$ are initially part of a single giant cluster. The data is then recursively split into smaller and smaller clusters. The splitting continues until the algorithm stops according to a user-specified objective. The divisive method can split the data until each data point is its own node.

In general, the merging and splitting of data is accomplished with a heuristic, greedy algorithm, which is easy to execute computationally. The results of hierarchical clustering are usually presented in a dendrogram.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

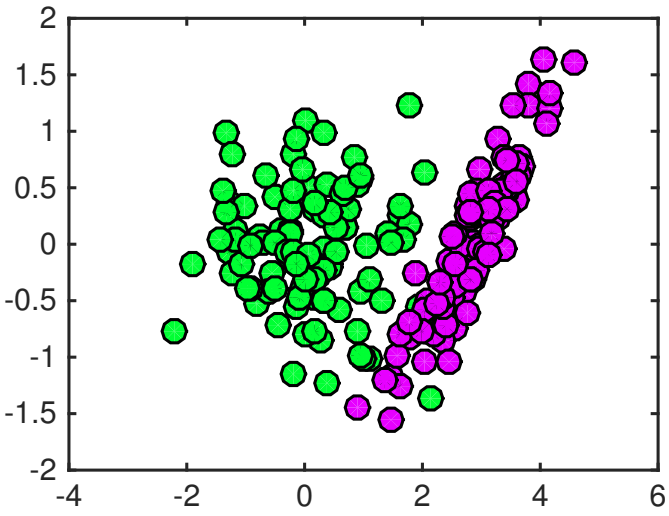


Figure 5.12: Example data used for construction of a dendrogram. The data is constructed from two Gaussian distributions (50 points each) that are easy to discern through a visual inspection. The dendrogram will produce a hierarchy that ideally would separate green balls from magenta balls.

In this section, we will focus on agglomerative hierarchical clustering and the dendrogram command from MATLAB. Like the Lloyd algorithm for k-means clustering, building the dendrogram proceeds from a simple algorithmic structure based on computing the distance between data points. Although we typically use a Euclidean distance, there are a number of important distance metrics one might consider for different types of data. Some typical distances are given as follows:

Euclidean distance	$\|\mathbf{x}_j - \mathbf{x}_k\|_2,$	(5.14a)
squared Euclidean distance	$\|\mathbf{x}_j - \mathbf{x}_k\|_2^2,$	(5.14b)
Manhattan distance	$\|\mathbf{x}_j - \mathbf{x}_k\|_1,$	(5.14c)
maximum distance	$\|\mathbf{x}_j - \mathbf{x}_k\|_\infty,$	(5.14d)
Mahalanobis distance	$\sqrt{(\mathbf{x}_j - \mathbf{x}_k)^T \mathbf{C}^{-1} (\mathbf{x}_j - \mathbf{x}_k)},$	(5.14e)

where $\mathbf{C}^{-1}$ is the covariance matrix. As already illustrated in the previous chapter, the choice of norm can make a tremendous difference for exposing patterns in the data that can be exploited for clustering and classification.

The dendrogram algorithm is shown in Fig. 5.13. The algorithm is as follows: (i) Compute the distance between all m data points $\mathbf{x}_j$ (Fig. 5.13 illustrates the use of a Euclidian distance). (ii) Merge the closest two data points into a single new data point midway between their original locations. (iii) Repeat the calculation with the new $m - 1$ points. The algorithm continues until the data has been hierarchically merged into a single data point.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

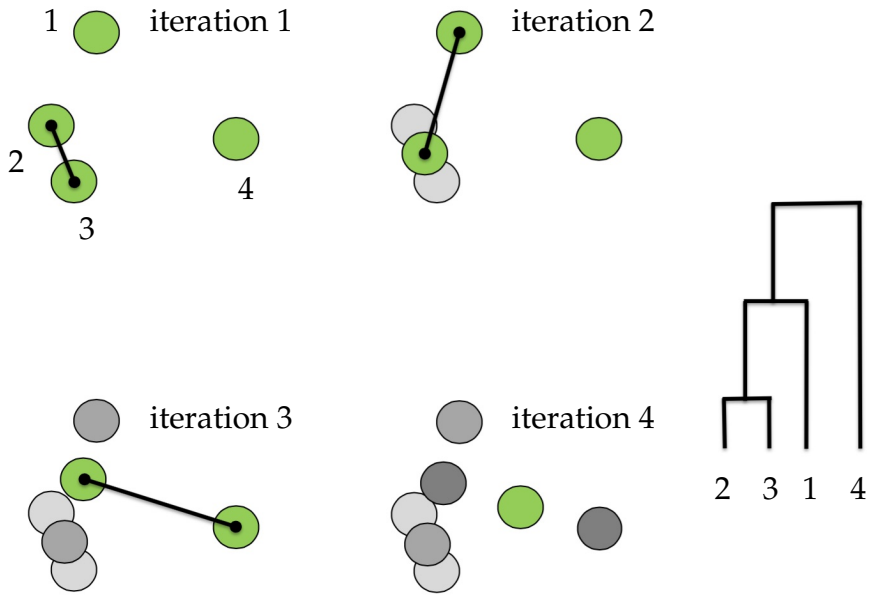


Figure 5.13: Illustration of the agglomerative hierarchical clustering scheme applied to four data points. In the algorithm, the distances between the four data points are computed. Initially, the Euclidean distance between points 2 and 3 is the least. Points 2 and 3 are thus merged into a point midway between them, and the distances are once again computed. The dendrogram on the right shows how the process generates a summary (dendrogram) of the hierarchical clustering. Note that the length of the branches of the dendrogram tree are directly related to the distance between the merged points.

The following code performs a hierarchical clustering using the **dendrogram** command from MATLAB. The example we use is the same as that considered for k-means clustering. Figure 5.12 shows the data under consideration. Visual inspection shows two clear clusters that are easily discernible. As with k-means, our goal is to see how well a dendrogram can extract the two clusters.

Code 5.7: [MATLAB] Dendrogram for unsupervised clustering.

```
Y3=[X1(1:50,:); X2(1:50,:)];
Y2 = pdist(Y3,'euclidean');
Z = linkage(Y2,'average');
thresh=0.85*max(Z(:,3));
[H,T,O]=dendrogram(Z,100,'ColorThreshold',thresh);
```

Code 5.7: [Python] Dendrogram for unsupervised clustering.

```
Y2 = pdist(Y3,metric='euclidean')
Z = hierarchy.linkage(Y2,method='average')
thresh = 0.85*np.max(Z[:,2])
dn = hierarchy.dendrogram(Z,p=100,color_threshold=thresh)
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

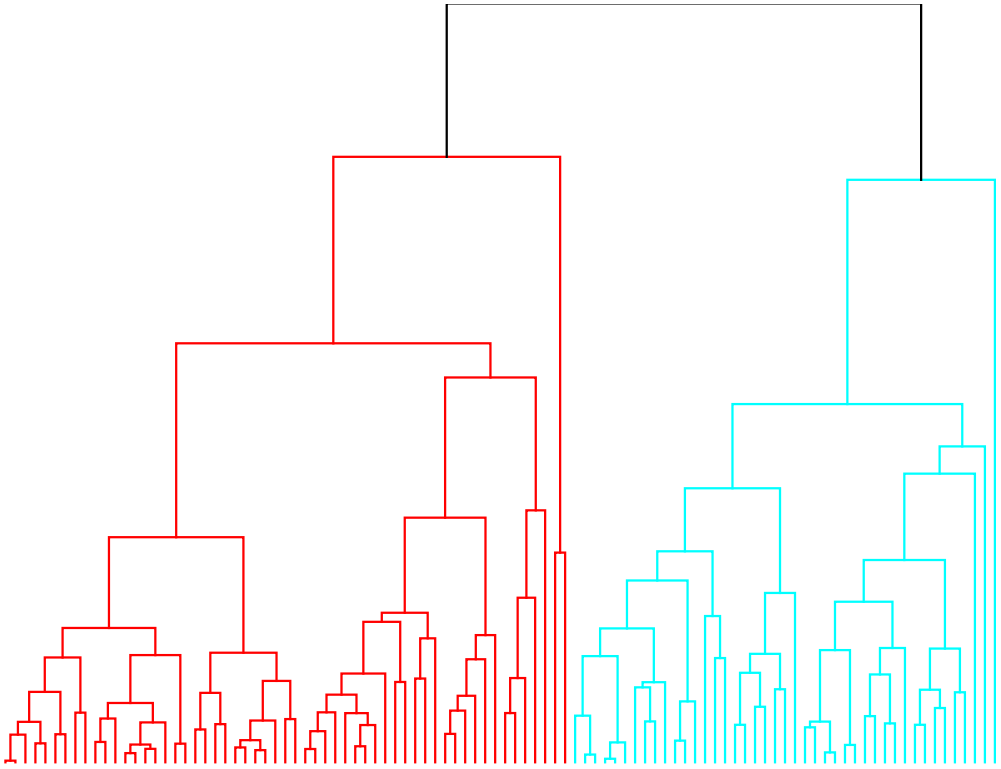


Figure 5.14: Dendrogram structure produced from the data in Fig. 5.12. The dendrogram shows which points are merged as well as the distance between points. Two clusters are generated for this level of threshold.

Figure 5.14 shows the dendrogram associated with the data in Fig. 5.12. The structure of the algorithm shows which points are merged as well as the distance between points. The threshold command is important in labeling where each point belongs in the hierarchical scheme. By setting the threshold at different levels, there can be more or fewer clusters in the dendrogram. The output of the dendrogram is used to show how the data was labeled. Recall that the first 50 data points are from the green cluster and the second 50 data points are from the magenta cluster.

Figure 5.15 shows how the data was clustered in the dendrogram. If perfect clustering had been achieved, then the first 50 points would have been below the horizontal dotted red line while the second 50 points would have been above the horizontal dotted red line. The vertical dotted red line is the line separating the green dots on the left from the magenta dots on the right.

A greater number of clusters are generated by adjusting the threshold in the **dendrogram** command. This is equivalent to setting the number of clusters in k-means to something greater than two. Recall that one rarely has a ground truth to compare with when doing unsupervised clustering, so tuning the threshold becomes important.

Figure 5.16 shows a new dendrogram with a different threshold. Note that

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

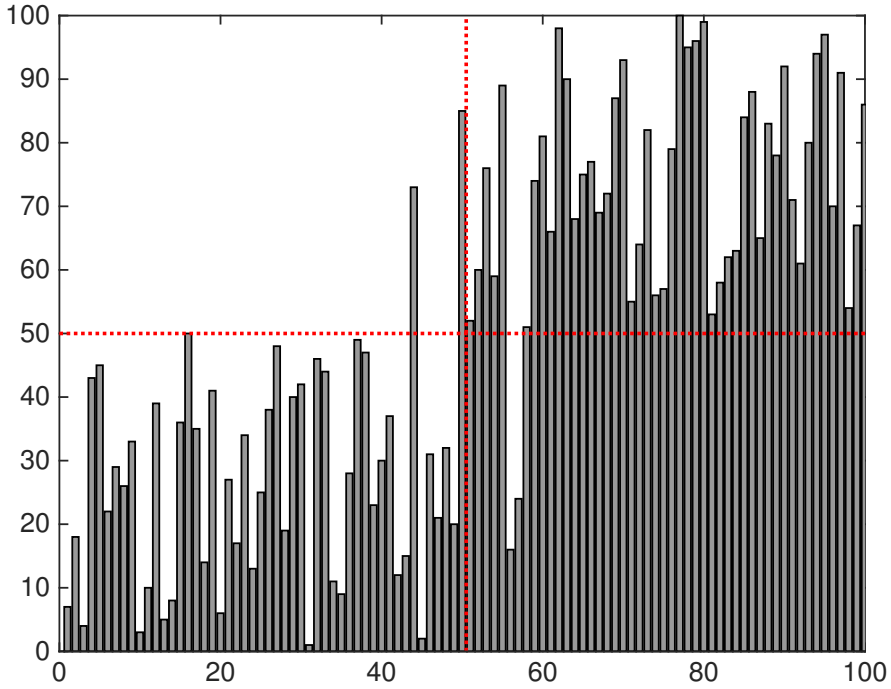


Figure 5.15: Clustering outcome from dendrogram routine. This is a summary of Fig. 5.14, showing how each of the points was clustered through the distance metric. The horizontal red dotted line shows where the ideal separation should occur. The first 50 points (green dots of Fig. 5.12) should be grouped so that they are below the red horizontal line in the lower left quadrant. The second 50 points (magenta dots of Fig. 5.12) should be grouped above the red horizontal line in the upper right quadrant. In summary, the dendrogram only misclassified two green points and two magenta points.

in this case, the hierarchical clustering produces more than a dozen clusters. The tuning parameter can be seen to be critical for unsupervised clustering, much like choosing the number of clusters in k-means. In summary, both k-means and hierarchical clustering provide a method whereby data can be parsed automatically into clusters. This provides a starting point for interpretations and analysis in data mining.

5.5 Mixture Models and the Expectation-Maximization Algorithm

The third unsupervised method we consider is known as *finite mixture models*. Often the models are assumed to be Gaussian distributions, in which case this method is known as *Gaussian mixture models* (GMM). The basic assumption in this method is that data observations $\mathbf{x}_j$ are a mixture of a set of k processes

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

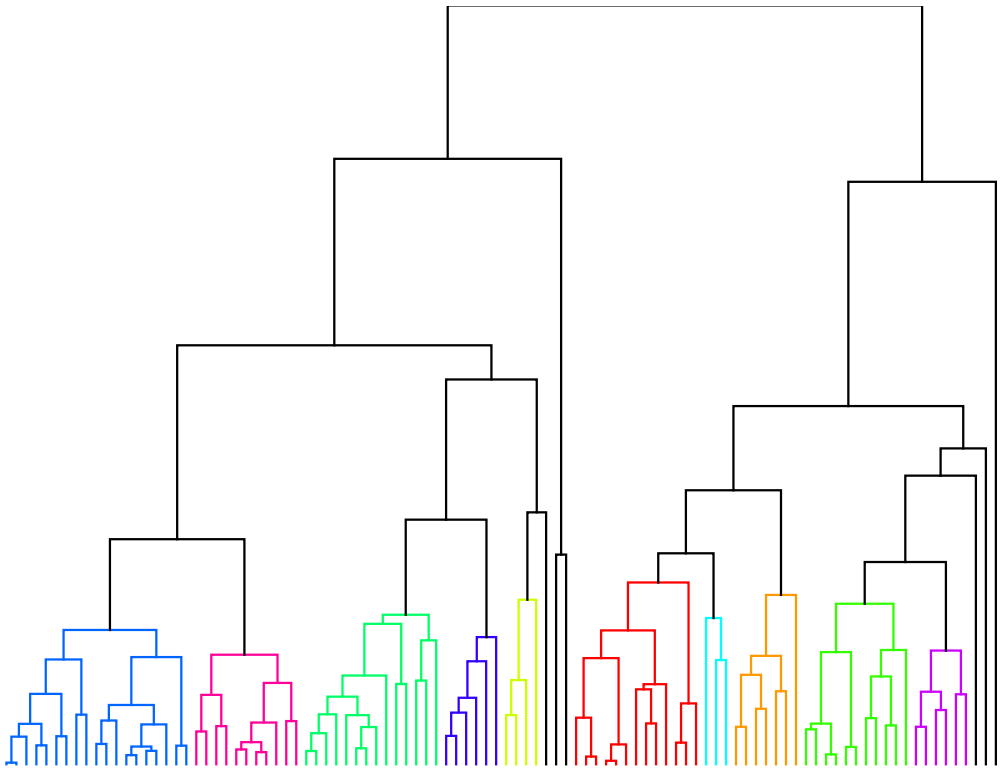


Figure 5.16: Dendrogram structure produced from the data in Fig. 5.12 with a different threshold used than in Fig. 5.14. The dendrogram shows which points are merged as well as the distance between points. In this case, more than a dozen clusters are generated.

that combine to form the measurement. Like k-means and hierarchical clustering, the GMM model we fit to the data requires that we specify the number of mixtures k and the individual statistical properties of each mixture that best fit the data. GMMs are especially useful since the assumption that each mixture model has a Gaussian distribution implies that it can be completely characterized by two parameters: the mean and the variance.

The algorithm that enables the GMM computes the maximum-likelihood using the famous *expectation-maximization* (EM) algorithm of Dempster, Laird, and Rubin [200]. The EM algorithm is designed to find maximum-*likelihood* parameters of statistical models. Likelihood is a fundamental concept of statistics and probability theory [91, 453]. Although not covered here, it provides the mathematical construct for the EM algorithm and GMM. Generally, the iterative structure of the algorithm finds a local maximum likelihood, which estimates the true parameters that cannot be directly solved for. As with most data, the observed data involves many latent or unmeasured variables and unknown parameters. Regardless, the alternating and iterative construction of the algo-

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

rithm recursively estimates the best parameters possible from an initial guess. The EM algorithm proceeds like the k-means algorithm in that initial guesses for the mean and variance are given for the assumed k-distributions. The algorithm then recursively updates the weights of the mixtures versus the parameters of each mixture. One alternates between these two until convergence is achieved.

In any such iteration scheme, it is not obvious that the solution will converge, or that the solution is good, since it typically falls into a local value of the maximum likelihood. But it can be proven that in this context it does converge, and that the derivative of the likelihood is arbitrarily close to zero at that point, which in turn means that the point is either a maximum or a saddle point [763]. In general, multiple maxima may occur, with no guarantee that the global maximum will be found. Some likelihoods also have singularities, i.e., nonsensical maxima. For example, one of the solutions that may be found by EM in a mixture model involves setting one of the components to have zero variance and the mean equal to one of the data points. Cross-validation can often alleviate some of the common pitfalls that can occur by initializing the algorithm with some bad initial guesses.

The fundamental assumption of the mixture model is that the probability density function (PDF) for observations of data $\mathbf{x}_j$ is a weighted linear sum of a set of unknown distributions,

$$f(\mathbf{x}_j, \boldsymbol{\Theta}) = \sum_{p=1}^{k} \alpha_p f_p(\mathbf{x}_j, \boldsymbol{\Theta}_p), \tag{5.15}$$

where $f(\cdot)$ is the measured PDF, $f_p(\cdot)$ is the PDF of the mixture p, and k is the total number of mixtures. Each of the PDFs $f_p(\cdot)$ is weighted by α_p (with $\alpha_1 + \alpha_2 + \cdots + \alpha_k = 1$) and parameterized by an unknown vector of parameters $\boldsymbol{\Theta}_p$. To state the objective of mixture models more precisely then: *Given the observed PDF $f(\mathbf{x}_j, \boldsymbol{\Theta})$, estimate the mixture weights α_p and the parameters of the distribution $\boldsymbol{\Theta}_p$*. Note that $\boldsymbol{\Theta}$ is a vector containing all the parameters $\boldsymbol{\Theta}_p$. Making this task somewhat easier is the fact that we assume the form of the PDF distribution $f_p(\cdot)$.

For GMM, the parameters in the vector $\boldsymbol{\Theta}_p$ are known to include only two variables: the mean μ_p and variance σ_p. Moreover, the distribution $f_p(\cdot)$ is normally distributed, so that (5.15) becomes

$$f(\mathbf{x}_j, \boldsymbol{\Theta}) = \sum_{p=1}^{k} \alpha_p \mathcal{N}_p(\mathbf{x}_j, \mu_p, \sigma_p). \tag{5.16}$$

This gives a much more tractable framework since there is now a limited set of parameters. Thus, once one assumes a number of mixtures k, then the task

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

is to determine α_p along with μ_p and σ_p for each mixture. It should be noted that there are many other distributions besides Gaussian that can be imposed, but GMM are common since, without prior knowledge, an assumption of a Gaussian distribution is typically assumed.

An estimate of the parameter vector $\boldsymbol{\Theta}$ can be computed using the *maximum-likelihood estimate* (MLE) of Fisher. The MLE computes the value of $\boldsymbol{\Theta}$ from the roots of

$$\frac{\partial L(\boldsymbol{\Theta})}{\partial \boldsymbol{\Theta}} = 0, \tag{5.17}$$

where the log-likelihood function L is

$$L(\boldsymbol{\Theta}) = \sum_{j=1}^{n} \log f(\mathbf{x}_j|\boldsymbol{\Theta}) \tag{5.18}$$

and the sum is over all the n data vectors $\mathbf{x}_j$. The solution to this optimization problem, i.e., when the derivative is zero, produces a local maximizer. This maximizer can be computed using the EM algorithm since derivatives cannot be explicitly computed without an analytic form.

The EM algorithm starts by assuming an initial estimate (guess) of the parameter vector $\boldsymbol{\Theta}$. This estimate can be used to estimate

$$\tau_p(\mathbf{x}_j, \boldsymbol{\Theta}) = \frac{\alpha_p f_p(\mathbf{x}_j, \boldsymbol{\Theta}_p)}{f(\mathbf{x}_j, \boldsymbol{\Theta})}, \tag{5.19}$$

which is the posterior probability of component membership of $\mathbf{x}_j$ in the pth distribution. In other words, does $\mathbf{x}_j$ belong to the pth mixture? The E step of the EM algorithm uses this posterior to compute memberships. For GMM, the algorithm proceeds as follows: Given an initial parameterization of $\boldsymbol{\Theta}$ and α_p, compute

$$\tau_p^{(k)}(\mathbf{x}_j) = \frac{\alpha_p^{(k)} \mathcal{N}_p(\mathbf{x}_j, \mu_p^{(k)}, \sigma_p^{(k)})}{f(\mathbf{x}_j, \boldsymbol{\Theta}^{(k)})}. \tag{5.20}$$

With an estimated posterior probability, the M step of the algorithm then updates the parameters and mixture weights,

$$\alpha_p^{(k+1)} = \frac{1}{n} \sum_{j=1}^{n} \tau_p^{(k)}(\mathbf{x}_j), \tag{5.21a}$$

$$\mu_p^{(k+1)} = \frac{\sum_{j=1}^{n} \mathbf{x}_j \tau_p^{(k)}(\mathbf{x}_j)}{\sum_{j=1}^{n} \tau_p^{(k)}(\mathbf{x}_j)}, \tag{5.21b}$$

$$\Sigma_p^{(k+1)} = \frac{\sum_{j=1}^{n} \tau_p^{(k)}(\mathbf{x}_j)(\mathbf{x}_j - \mu_p^{(k+1)})(\mathbf{x}_j - \mu_p^{(k+1)})^T}{\sum_{j=1}^{n} \tau_p^{(k)}(\mathbf{x}_j)}, \tag{5.21c}$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

where the matrix $\Sigma_p^{(k+1)}$ is the covariance matrix containing the variance parameters. The E and M steps are alternated until convergence within a specified tolerance. Recall that, to initialize the algorithm, the number of mixture models k must be specified and initial parameterization (guesses) of the distributions given. This is similar to the k-means algorithm, where the number of clusters k is prescribed and an initial guess for the cluster centers is specified.

The GMM is popular since it simply fits k Gaussian distributions to data, which is reasonable for unsupervised learning. The GMM algorithm also has a stronger theoretical base than most unsupervised methods, as both k-means and hierarchical clustering are simply defined as algorithms. The primary assumption in GMM is the number of clusters and the form of the distribution $f(\cdot)$.

The following code executes a GMM model on the second and fourth principal components of the dogs and cats wavelet image data introduced previously in Figs. 5.4–5.6. Thus the features are the second and fourth columns of the right singular vector of the SVD. The **fitgmdist** command is used to extract the mixture model.

Code 5.8: [MATLAB] Gaussian mixture model for cats versus dogs.

```
dogcat=v(:,2:2:4);
GMModel=fitgmdist(dogcat,2)
AIC= GMModel.AIC
```

Code 5.8: [Python] Gaussian mixture model for cats versus dogs.

```
GMModel = GaussianMixture(n_components=2).fit(dogcat)
AIC = GMModel.aic(dogcat)
```

The results of the algorithm can be plotted for visual inspection, and the parameters associated with each Gaussian are given: specifically, the mixing proportion of each model along with the mean in each of the two dimensions of the feature space. The following is displayed to the screen.

```
Component 1:
Mixing proportion: 0.355535
Mean:   -0.0290   -0.0753

Component 2:
Mixing proportion: 0.644465
Mean:    0.0758    0.0076

AIC =

 -792.8105
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

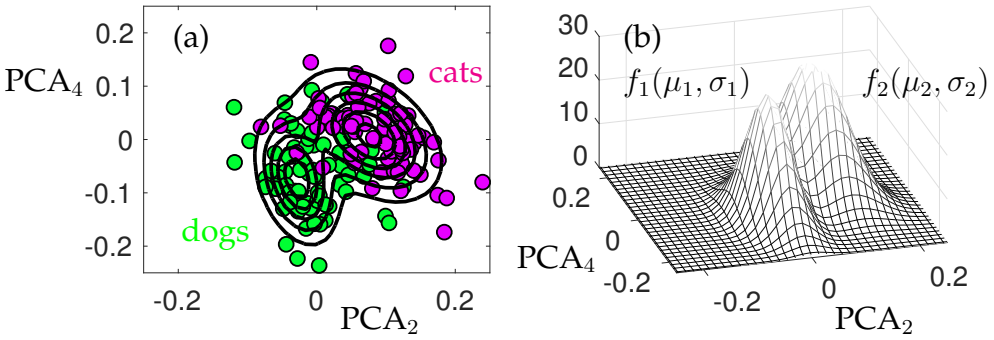


Figure 5.17: GMM fit of the second and fourth principal components of the dogs and cats wavelet image data. The two Gaussians are well placed over the distinct dogs and cats features as shown in (a). The PDF of the Gaussian models extracted are highlighted in (b) in arbitrary units.

The code can also produce an AIC score for how well the mixture of Gaussians explains the data. This gives a principled method for cross-validating in order to determine the number of mixtures required to describe the data.

Figure 5.17 shows the results of the GMM fitting procedure along with the original data of cats and dogs. The Gaussians produced from the fitting procedure are also illustrated. The **fitgmdist** command can also be used with **cluster** to label new data from the feature separation discovered by GMM.

5.6 Supervised Learning and Linear Discriminants

We now turn our attention to supervised learning methods. One of the earliest supervised methods for classification of data was developed by Fisher in 1936 in the context of taxonomy [245]. His *linear discriminant analysis* (LDA) is still one of the standard techniques for classification. It was generalized by C. R. Rao for multi-class data in 1948 [586]. The goal of these algorithms is to find a linear combination of features that characterizes or separates two or more classes of objects or events in the data. Importantly, for this supervised technique we have labeled data that guides the classification algorithm. Figure 5.18 illustrates the concept of finding an optimal low-dimensional embedding of the data for classification. The LDA algorithm aims to solve an optimization problem to find a subspace whereby the different labeled data have clear separation between their distributions of points. This then makes classification easier because an optimal feature space has been selected.

The supervised learning architecture includes a training set and a withhold set of data. The withhold set is never used to train the classifier. However, the training data can be partitioned into k folds, for instance, to help build a better classification model. The last chapter details how cross-validation should

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

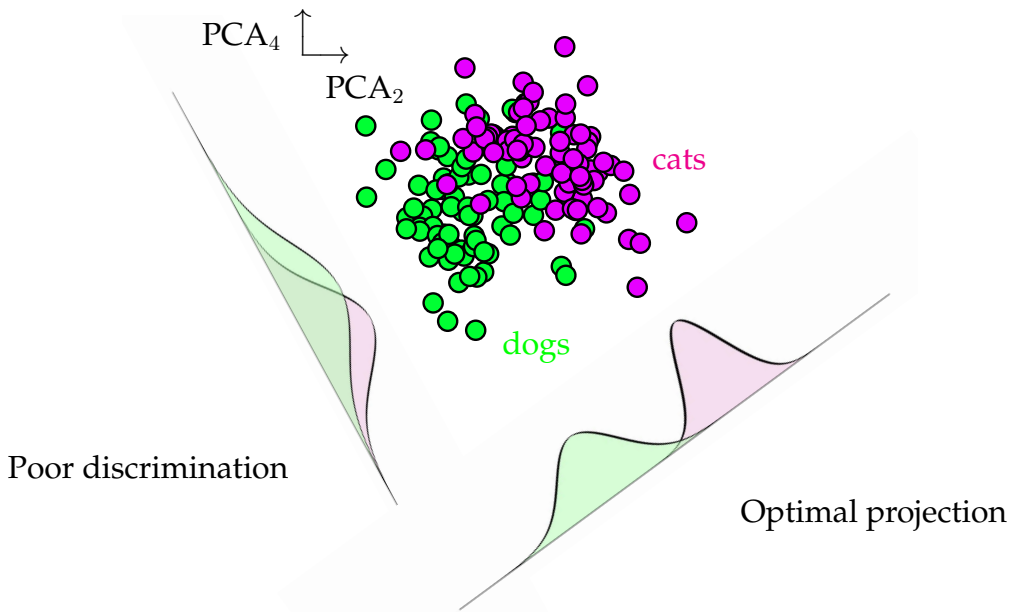


Figure 5.18: Illustration of linear discriminant analysis (LDA). The LDA optimization method produces an optimal dimensionality reduction to a decision line for classification. The figure illustrates the projection of data onto the second and fourth principal component modes of the dogs and cats wavelet data considered in Fig. 5.4. Without optimization, a general projection can lead to very poor discrimination between the data. However, the LDA separates the probability density functions in an optimal way.

be appropriately used. The goal here is to train an algorithm that uses feature space to make a decision about how to classify data. Figure 5.18 gives a cartoon of the key idea involved in LDA. In our example, two data sets are considered and projected onto new bases. On the left-hand side, the projection shows that the data is completely mixed, making it difficult to separate the data. On the right-hand side, which is the ideal caricature for LDA, the data are well separated, with the means μ_1 and μ_2 being well apart when projected onto the chosen subspace. Thus the goal of LDA is two-fold: *find a suitable projection that maximizes the distance between the inter-class data while minimizing the intra-class data*.

For a two-class LDA, this results in the following mathematical formulation. Construct a projection $\mathbf{w}$ such that

$$\mathbf{w} = \arg\max_{\mathbf{w}} \frac{\mathbf{w}^T \mathbf{S}_B \mathbf{w}}{\mathbf{w}^T \mathbf{S}_W \mathbf{w}}, \tag{5.22}$$

where the scatter matrices for between-class $\mathbf{S}_B$ and within-class $\mathbf{S}_W$ data are

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

given by

$$\mathbf{S}_B = (\mu_2 - \mu_1)(\mu_2 - \mu_1)^T, \tag{5.23}$$

$$\mathbf{S}_W = \sum_{j=1}^{2} \sum_{\mathbf{x} \in \mathcal{D}_j} (\mathbf{x} - \mu_j)(\mathbf{x} - \mu_j)^T. \tag{5.24}$$

These quantities essentially measure the variance of the data sets as well as the variance of the difference in the means. The criterion in (5.22) is commonly known as the generalized Rayleigh quotient, whose solution can be found via the generalized eigenvalue problem

$$\mathbf{S}_B \mathbf{w} = \lambda \mathbf{S}_W \mathbf{w}, \tag{5.25}$$

where the maximum eigenvalue λ and its associated eigenvector give the quantity of interest and the projection basis. Thus, once the scatter matrices are constructed, the generalized eigenvectors can be constructed with MATLAB.

Performing an LDA analysis in MATLAB is simple. One needs only to organize the data into a training set with labels, which can then be applied to a test data set. Given a set of data $\mathbf{x}_j$ for $j = 1, 2, \ldots, m$ with corresponding labels $\mathbf{y}_j$, the algorithm will find an optimal classification space as shown in Fig. 5.18. New data $\mathbf{x}_k$ with $k = m+1, m+2, \ldots, m+n$ can then be evaluated and labeled. We illustrate the classification of data using the dogs and cats data set introduced in the feature section of this chapter. Specifically, we consider the dogs and cats images in the wavelet domain and label them so that $\mathbf{y}_j \in \{\pm 1\}$ (where $\mathbf{y}_j = 1$ is a dog and $\mathbf{y}_j = -1$ is a cat). The following code trains on the first 60 images of dogs and cats, and then tests the classifier on the remaining 20 dogs and cats images. For simplicity, we train on the second and fourth principal components, as these show good discrimination between dogs and cats (see Fig. 5.5).

Code 5.9: [MATLAB] LDA analysis of dogs versus cats.

```
class=classify(test,xtrain,label);
```

Code 5.9: [Python] LDA analysis of dogs versus cats.

```
lda = LinearDiscriminantAnalysis()
test_class = lda.fit(xtrain, label).predict(test)
```

Note that the **classify** command in MATLAB takes in the three matrices of interest: the training data, the test data, and the labels for the training data. What is produced are the labels for the test set. One can also extract from this command the decision line for online use. Figure 5.19 shows the results of the classification on the 40 test data samples. Recall that this classification is performed using only the second and fourth PCA modes, which cluster as shown

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

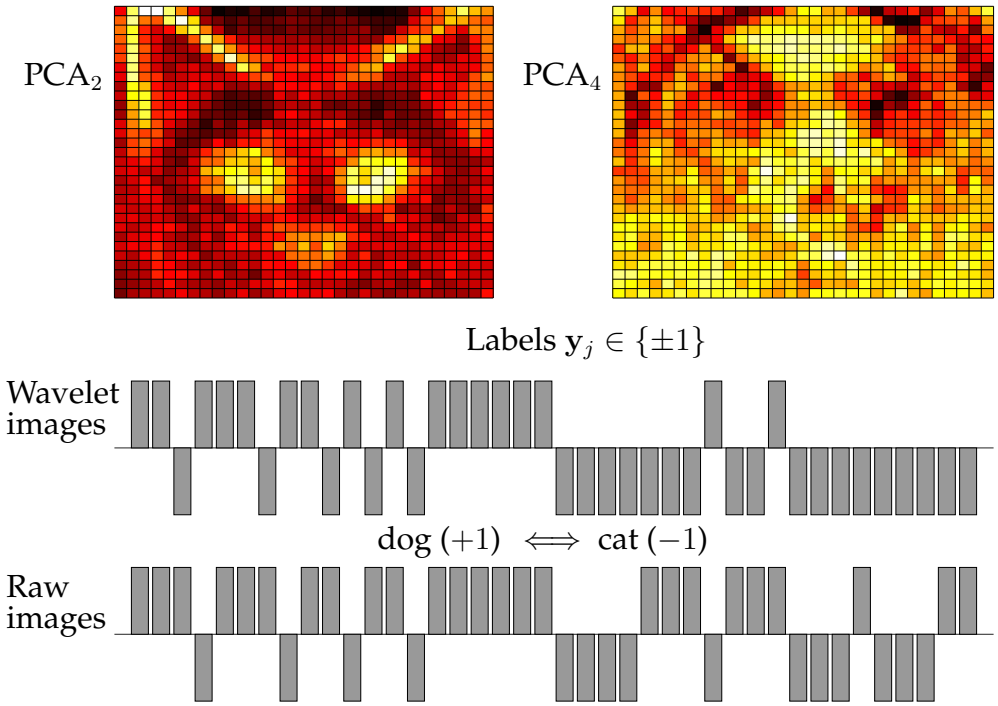


Figure 5.19: Depiction of the performance achieved for classification using the second and fourth principal component modes. The top two panels are PCA modes (features) used to build a classifier. The labels returned are either $\mathbf{y}_j \in \{\pm 1\}$. The ground-truth answer in this case should produce a vector of 20 ones followed by 20 negative ones.

in Fig. 5.18. The returned labels are either ± 1 depending on whether a cat or a dog is labeled. The ground-truth labels for the test data should return a $+1$ (dogs) for the first 20 test sets and a -1 (cats) for the second test set. The accuracy of classification for this realization is 82.5% (2/20 cats are mislabeled while 5/20 dogs are mislabeled). Comparing the wavelet images to the raw images, we see that the feature selection in the raw images is not as good. In particular, for the same two principal components, 9/20 cats are mislabeled and 4/20 dogs are mislabeled. Of course, the data is fairly limited and cross-validation should always be performed to evaluate the classifier. We run 100 trials of the **classify** command where 60 dogs and cats images are randomly selected and tested against the remaining 20 images.

Figure 5.20 shows the results of the cross-validation over 100 trials. Note the variability that can occur from trial to trial. Specifically, the performance can achieve 100%, but can also be as low as 40%, which is worse than a coin flip. The average classification score (red dotted line) is around 70%. Cross-validation, as already highlighted in the regression chapter, is critical for testing and robustifying the model. Recall that the methods for producing a classifier are based

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

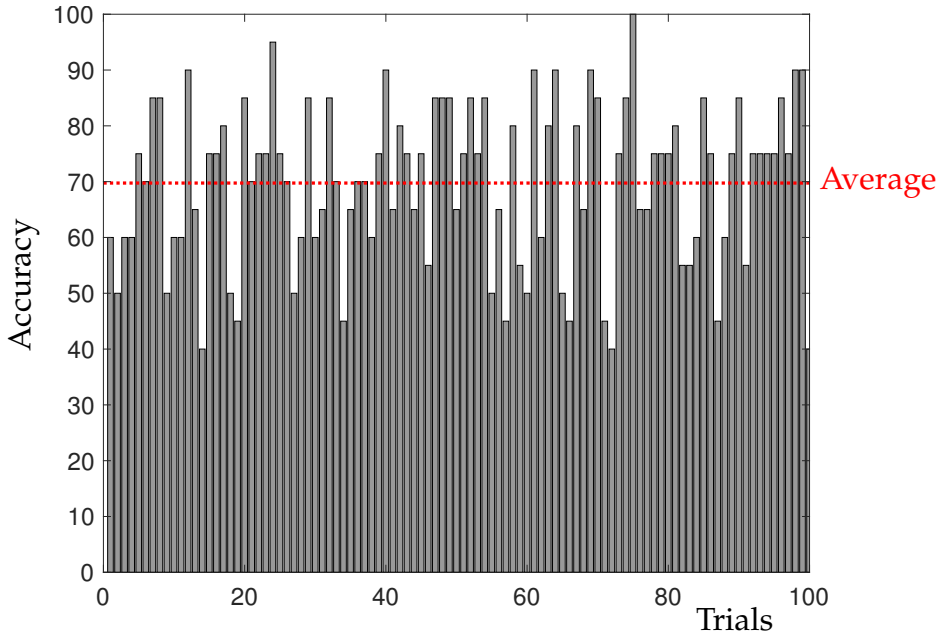


Figure 5.20: Performance of the LDA over 100 trials. Note the variability that can occur in the classifier depending on which data is selected for training and testing. This highlights the importance of cross-validation for building a robust classifier.

on optimization and regression, so that all the cross-validation methods can be ported to the clustering and classification problem.

In addition to a linear discriminant line, a quadratic discriminant line can be found to separate the data. Indeed, the **classify** command in MATLAB allows one to not only produce the classifier, but also extract the line of separation between the data.

Figure 5.21 shows the dogs and cats data along with the linear and quadratic lines separating them. This linear or quadratic fit is found in the structured variable **coeff** which is returned with classify. The quadratic line of separation can often offer a little more flexibility when trying to fit boundaries separating data. A major advantage of LDA-based methods is that they are easily interpretable and easy to compute. Thus, they are widely used across many branches of the sciences for classification of data.

5.7 Support Vector Machines (SVM)

One of the most successful data-mining methods developed to date is the *support vector machine* (SVM). It is a core machine learning tool that is used widely in industry and science, often providing results that are better than competing methods. Along with the *random forest* algorithm, they have been pillars of ma-

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

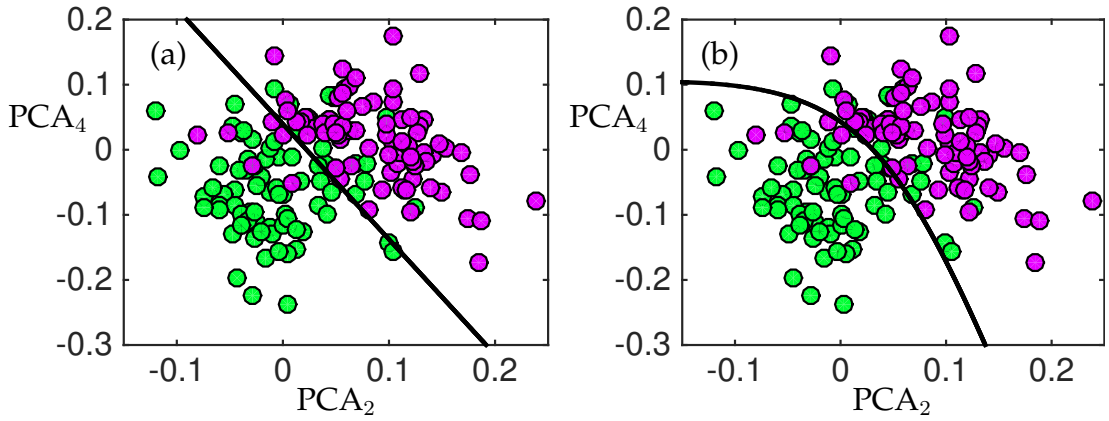


Figure 5.21: Classification line for (a) linear discriminant analysis (LDA) and (b) quadratic discriminant analysis (QDA) for dog (green dots) versus cat (magenta dots) data projected onto the second and fourth principal components. This two-dimensional feature space allows for a good discrimination in the data. The two lines represent the best line and parabola for separating the data for a given training sample.

chine learning in the last few decades. With enough training data, the SVM can now be replaced with deep neural nets. But otherwise, SVM and random forest are frequently used algorithms for applications where the best classification scores are required.

The original SVM algorithm by Vapnik and Chervonenkis evolved out of the statistical learning literature in 1963, where hyperplanes are optimized to split the data into distinct clusters. Nearly three decades later, Boser, Guyon and Vapnik created nonlinear classifiers by applying the kernel trick to maximum-margin hyperplanes [98]. The current standard incarnation (soft margin) was proposed by Cortes and Vapnik in the mid-1990s [184].

Linear SVM

The key idea of the linear SVM method is to construct a hyperplane

$$\mathbf{w} \cdot \mathbf{x} + b = 0, \tag{5.26}$$

where the vector $\mathbf{w}$ and constant b parameterize the hyperplane. Figure 5.22 shows two potential hyperplanes splitting a set of data. Each has a different value of $\mathbf{w}$ and constant b. The optimization problem associated with SVM is not only to optimize a decision line that makes the fewest labeling errors for the data, but also to optimize the largest margin between the data, shown in the shaded regions of Fig. 5.22. The vectors that determine the boundaries of the margin, i.e., the vectors touching the edge of the shaded regions, are termed the *support vectors*. Given the hyperplane (5.26), a new data point $\mathbf{x}_j$ can be clas-

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

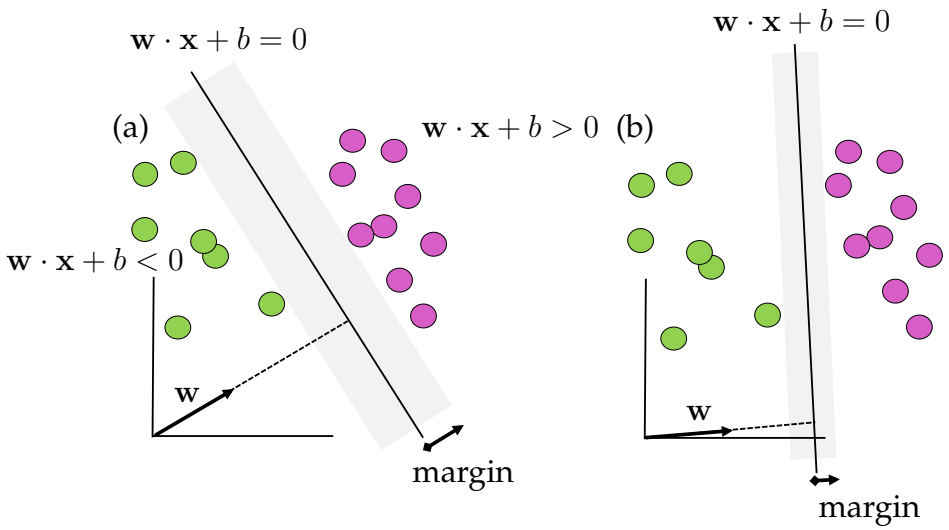


Figure 5.22: The SVM classification scheme constructs a hyperplane $\mathbf{w}\cdot\mathbf{x}+b=0$ that optimally separates the labeled data. The area of the margin separating the labeled data is maximal in (a) and much less in (b). Determining the vector $\mathbf{w}$ and parameter b is the goal of the SVM optimization. Note that for data to the right of the hyperplane $\mathbf{w}\cdot\mathbf{x}+b>0$, while for data to the left $\mathbf{w}\cdot\mathbf{x}+b<0$. Thus the classification labels $\mathbf{y}_j\in\{\pm 1\}$ for the data to the left or right of the hyperplane is given by $\mathbf{y}_j(\mathbf{w}\cdot\mathbf{x}_j+b)=\text{sign}(\mathbf{w}\cdot\mathbf{x}_j+b)$. So only the sign of $\mathbf{w}\cdot\mathbf{x}+b$ needs to be determined in order to label the data. The vectors touching the edge of the shaded regions are termed the *support vectors*.

sified by simply computing the sign of $(\mathbf{w}\cdot\mathbf{x}_j+b)$. Specifically, for classification labels $\mathbf{y}_j\in\{\pm 1\}$, the data to the left or right of the hyperplane is given by

$$\mathbf{y}_j(\mathbf{w}\cdot\mathbf{x}_j+b)=\text{sign}(\mathbf{w}\cdot\mathbf{x}_j+b)=\begin{cases}+1 & \text{magenta ball,}\\ -1 & \text{green ball.}\end{cases} \tag{5.27}$$

Thus the classifier $\mathbf{y}_j$ is explicitly dependent on the position of $\mathbf{x}_j$.

Critical to the success of the SVM is determining $\mathbf{w}$ and b in a principled way. As with all machine learning methods, an appropriate optimization must be formulated. The optimization is aimed at both minimizing the number of misclassified data points as well as creating the largest margin possible. To construct the optimization objective function, we define a loss function

$$\ell(\mathbf{y}_j,\bar{\mathbf{y}}_j)=\ell(\mathbf{y}_j,\text{sign}(\mathbf{w}\cdot\mathbf{x}_j+b))=\begin{cases}0 & \text{if } \mathbf{y}_j=\text{sign}(\mathbf{w}\cdot\mathbf{x}_j+b),\\ +1 & \text{if } \mathbf{y}_j\neq\text{sign}(\mathbf{w}\cdot\mathbf{x}_j+b).\end{cases} \tag{5.28}$$

Stated more simply,

$$\ell(\mathbf{y}_j,\bar{\mathbf{y}}_j)=\begin{cases}0 & \text{if data is correctly labeled,}\\ +1 & \text{if data is incorrectly labeled.}\end{cases} \tag{5.29}$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Thus each mislabeled point produces a loss of unity. The training error over m data points is the sum of the loss functions $\ell(\mathbf{y}_j, \bar{\mathbf{y}}_j)$.

In addition to minimizing the loss function, the goal is also to make the margin as large as possible. We can then frame the linear SVM optimization problem as

$$\underset{\mathbf{w},b}{\operatorname{argmin}} \sum_{j=1}^{m} \ell(\mathbf{y}_j, \bar{\mathbf{y}}_j) + \frac{1}{2}\|\mathbf{w}\|^2 \quad \text{subject to} \quad \min_j |\mathbf{x}_j \cdot \mathbf{w}| = 1. \tag{5.30}$$

Although this is a concise statement of the optimization problem, the fact that the loss function is discrete and constructed from ones and zeros makes it very difficult to actually optimize. Most optimization algorithms are based on some form of gradient descent, which requires smooth objective functions in order to compute derivatives or gradients to update the solution. A more common formulation then is given by

$$\underset{\mathbf{w},b}{\operatorname{argmin}} \sum_{j=1}^{m} H(\mathbf{y}_j, \bar{\mathbf{y}}_j) + \frac{1}{2}\|\mathbf{w}\|^2 \quad \text{subject to} \quad \min_j |\mathbf{x}_j \cdot \mathbf{w}| = 1, \tag{5.31}$$

where $H(\mathbf{y}_j, \bar{\mathbf{y}}_j) = \max(0,\ 1 - \mathbf{y}_j \cdot \bar{\mathbf{y}}_j)$ is called a Hinge loss function. This is a smooth function that counts the number of errors in a linear way and that allows for piecewise differentiation so that standard optimization routines can be employed.

Nonlinear SVM

Although easily interpretable, linear classifiers are of limited value. They are simply too restrictive for data embedded in a high-dimensional space and which may have the structured separation as illustrated in Fig. 5.8. To build more sophisticated classification curves, the feature space for SVM must be enriched. SVM does this by including nonlinear features and then building hyperplanes in this new space. To do this, one simply maps the data into a nonlinear, higher-dimensional space

$$\mathbf{x} \mapsto \mathbf{\Phi}(\mathbf{x}). \tag{5.32}$$

We can call the $\mathbf{\Phi}(\mathbf{x})$ new *observables* of the data. The SVM algorithm now learns the hyperplanes that optimally split the data into distinct clusters in a new space. Thus one now considers the hyperplane function

$$f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{\Phi}(\mathbf{x}) + b, \tag{5.33}$$

with corresponding labels $\mathbf{y}_j \in \{\pm 1\}$ for each point $f(\mathbf{x}_j)$.

This simple idea, of enriching feature space by defining new functions of the data $\mathbf{x}$, is exceptionally powerful for clustering and classification. As a simple

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

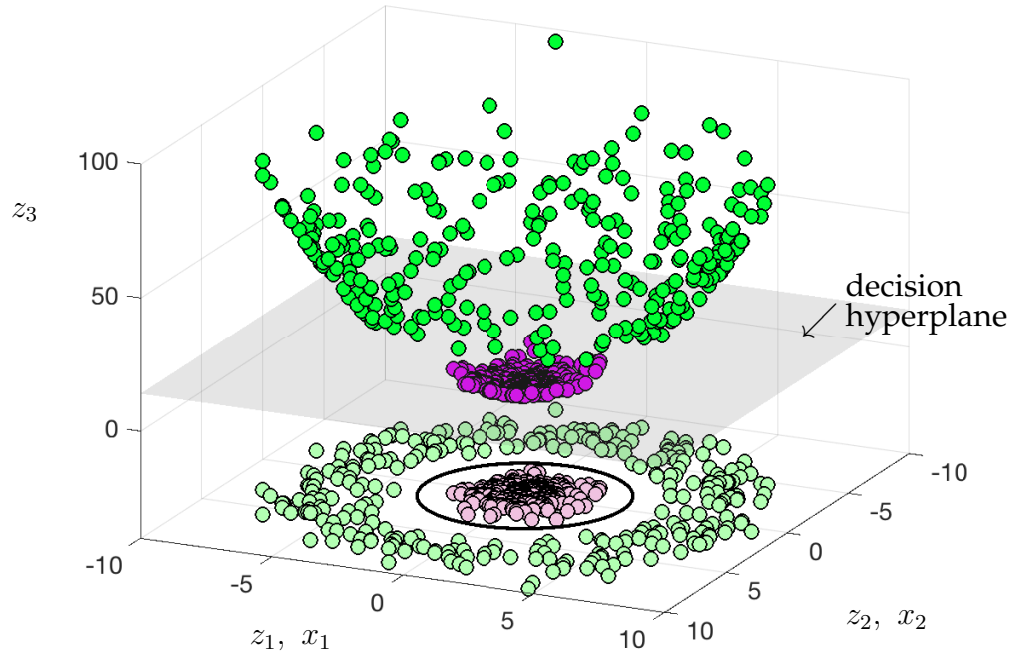


Figure 5.23: The nonlinear embedding of Fig. 5.8(b) using the variables $(x_1, x_2) \mapsto (z_1, z_2, z_3) := (x_1, x_2, x_1^2 + x_2^2)$ in (5.34). A hyperplane can now easily separate the green from magenta balls, showing that linear classification can be accomplished simply by enriching the measurement space of the data. Visual inspection alone suggests that nearly optimal separation can be achieved with the plane $z_3 \approx 14$ (shaded gray plane). In the original coordinate system, this gives a circular classification line (black line on the plane x_1 versus x_2) with radius $r = \sqrt{z_3} = \sqrt{x_1^2 + x_2^2} \approx \sqrt{14}$. This example makes it obvious how a hyperplane in higher dimensions can produce curved classification lines in the original data space.

example, consider two-dimensional data $\mathbf{x} = (x_1, x_2)$. One can easily enrich the space by considering polynomials of the data:

$$(x_1, x_2) \mapsto (z_1, z_2, z_3) := (x_1, x_2, x_1^2 + x_2^2). \tag{5.34}$$

This gives a new set of polynomial coordinates in x_1 and x_2 that can be used to embed the data. This philosophy is simple: by embedding the data in a higher-dimensional space, it is much more likely to be separable by hyperplanes. As a simple example, consider the data illustrated in Fig. 5.8(b). A linear classifier (or hyperplane) in the x_1–x_2 plane will clearly not be able to separate the data. However, the embedding (5.34) projects into a three-dimensional space, which can be easily separated by a hyperplane as illustrated in Fig. 5.23.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

The ability of SVM to embed in higher-dimensional nonlinear spaces makes it one of the most successful machine learning algorithms developed. The underlying optimization algorithm (5.31) remains unchanged, except that the previous labeling function $\bar{\mathbf{y}}_j = \text{sign}(\mathbf{w} \cdot \mathbf{x}_j + b)$ is now

$$\bar{\mathbf{y}}_j = \text{sign}(\mathbf{w} \cdot \mathbf{\Phi}(\mathbf{x}_j) + b). \tag{5.35}$$

The function $\mathbf{\Phi}(\mathbf{x})$ specifies the enriched space of observables. As a general rule, more features are better for classification.

Kernel Methods for SVM

Despite its promise, the SVM method of building nonlinear classifiers by enriching in higher dimensions leads to a computationally intractable optimization. Specifically, the large number of additional features leads to the *curse of dimensionality*. Thus computing the vectors $\mathbf{w}$ is prohibitively expensive and may not even be represented explicitly in memory. The *kernel trick* solves this problem. In this scenario, the $\mathbf{w}$ vector is represented as

$$\mathbf{w} = \sum_{j=1}^{m} \alpha_j \mathbf{\Phi}(\mathbf{x}_j), \tag{5.36}$$

where α_j are parameters that weight the different nonlinear observable functions $\mathbf{\Phi}(\mathbf{x}_j)$. Thus the vector $\mathbf{w}$ is expanded in the observable set of functions. We can then generalize (5.33) to the following:

$$f(\mathbf{x}) = \sum_{j=1}^{m} \alpha_j \mathbf{\Phi}(\mathbf{x}_j) \cdot \mathbf{\Phi}(\mathbf{x}) + b. \tag{5.37}$$

The *kernel function* [643] is then defined as

$$K(\mathbf{x}_j, \mathbf{x}) = \mathbf{\Phi}(\mathbf{x}_j) \cdot \mathbf{\Phi}(\mathbf{x}). \tag{5.38}$$

With this new definition of $\mathbf{w}$, the optimization problem (5.31) becomes

$$\underset{\boldsymbol{\alpha}, b}{\text{argmin}} \sum_{j=1}^{m} H(\mathbf{y}_j, \bar{\mathbf{y}}_j) + \frac{1}{2} \left\| \sum_{j=1}^{m} \alpha_j \mathbf{\Phi}(\mathbf{x}_j) \right\|^2 \quad \text{subject to} \quad \min_j |\mathbf{x}_j \cdot \mathbf{w}| = 1, \tag{5.39}$$

where $\boldsymbol{\alpha}$ is the vector of α_j coefficients that must be determined in the minimization process. There are different conventions for representing the minimization. However, in this formulation, the minimization is now over $\boldsymbol{\alpha}$ instead of $\mathbf{w}$.

In this formulation, the kernel function $K(\mathbf{x}_j, \mathbf{x})$ essentially allows us to represent Taylor series expansions of a large (infinite) number of observables in

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

a compact way [643]. The kernel function enables one to operate in a high-dimensional, implicit feature space without ever computing the coordinates of the data in that space, but rather by simply computing the inner products between all pairs of data in the feature space. For instance, two of the most commonly used kernel functions are

$$\text{radial basis function (RBF):} \quad K(\mathbf{x}_j, \mathbf{x}) = \exp(-\gamma \|\mathbf{x}_j - \mathbf{x}\|^2), \tag{5.40a}$$

$$\text{polynomial kernel:} \quad K(\mathbf{x}_j, \mathbf{x}) = (\mathbf{x}_j \cdot \mathbf{x} + 1)^N, \tag{5.40b}$$

where N is the degree of polynomials to be considered, which is exceptionally large to evaluate without using the kernel trick, and γ is the width of the Gaussian kernel measuring the distance between individual data points $\mathbf{x}_j$ and the classification line. These functions can be differentiated in order to optimize (5.39).

This represents the major theoretical underpinning of the SVM method. It allows us to construct higher-dimensional spaces using observables generated by kernel functions. Moreover, it results in a computationally tractable optimization. The following code shows the basic workings of the kernel method on the example of dogs and cats classification data. In the first example, a standard linear SVM is used, while in the second, the RBF is executed as an option.

Code 5.10: [MATLAB] SVM classification.

```
Mdl = fitcsvm(xtrain,label);
test_labels = predict(Mdl,test);

Mdl = fitcsvm(xtrain,label,'KernelFunction','RBF');
test_labels = predict(Mdl,test);
CMdl = crossval(Mdl);          % cross-validate the model
classLoss = kfoldLoss(CMdl)   % compute class loss
```

Code 5.10: [Python] SVM classification.

```
Mdl = svm.SVC(kernel='rbf',gamma='auto').fit(xtrain,label)
test_labels = Mdl.predict(test)

CMdl = cross_val_score(Mdl, xtrain, label, cv=10) #cross-
    validate the model
classLoss = 1-np.mean(CMdl) # average error over all cross-
    validation iterations
```

Note that in this code we have demonstrated some of the diagnostic features of the SVM method in MATLAB, including the cross-validation and class loss scores that are associated with training. This is a superficial treatment of the SVM. Overall, SVM is one of the most sophisticated machine learning tools in MATLAB, and there are many options that can be executed in order to tune performance and extract accuracy/cross-validation metrics.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

5.8 Classification Trees and Random Forest

Decision trees are common in business. They establish an algorithmic flow chart for making decisions based on criteria that are deemed important and related to a desired outcome. Often the decision trees are constructed by experts with knowledge of the workflow involved in the decision making process. *Decision tree learning* provides a principled method based on data for creating a predictive model for classification and/or regression. Along with SVM, classification and regression trees are core machine learning and data-mining algorithms used in industry, given their demonstrated success. The work of Leo Breiman and co-workers [110] established many of the theoretical foundations exploited today for data mining.

The decision tree is a hierarchical construct that looks for optimal ways to split the data in order to provide a robust classification and regression. It is the opposite of the unsupervised dendrogram hierarchical clustering previously demonstrated. In this case, our goal is not to move from bottom up in the clustering process, but from top down in order to create the best splits possible for classification. The fact that it is a supervised algorithm, which uses labeled data, allows us to split the data accordingly.

There are significant advantages in developing decision trees for classification and regression: (i) they often produce interpretable results that can be graphically displayed, making them easy to interpret even for non-experts; (ii) they can handle numerical or categorical data equally well; (iii) they can be statistically validated so that the reliability of the model can be assessed; (iv) they perform well with large data sets at scale; and (v) the algorithms mirror human decision making, again making them more interpretable and useful.

As one might expect, the success of decision tree learning has produced a large number of innovations and algorithms for how to best split the data. The coverage here will be limited, but we will highlight the basic architecture for data splitting and tree construction. Recall that we have the following:

$$\text{data} \quad \{\mathbf{x}_j \in \mathbb{R}^n,\ j \in Z := \{1, 2, \ldots, m\}\}, \tag{5.41a}$$

$$\text{labels} \quad \{\mathbf{y}_j \in \{\pm 1\},\ j \in Z' \subset Z\}. \tag{5.41b}$$

The basic decision tree algorithm is fairly simple: (i) Scan through each component (feature) x_k (with $k = 1, 2, \ldots, n$) of the vector $\mathbf{x}_j$ to identify the value of x_j that gives the best labeling prediction for $\mathbf{y}_j$. (ii) Compare the prediction accuracy for each split on the feature x_j. The feature giving the best segmentation of the data is selected as the split for the tree. (iii) With the two new branches of the tree created, this process is repeated on each branch. The algorithm terminates once each individual data point is a unique cluster, known as a *leaf*, on a new branch of the tree. This is essentially the inverse of the dendrogram.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

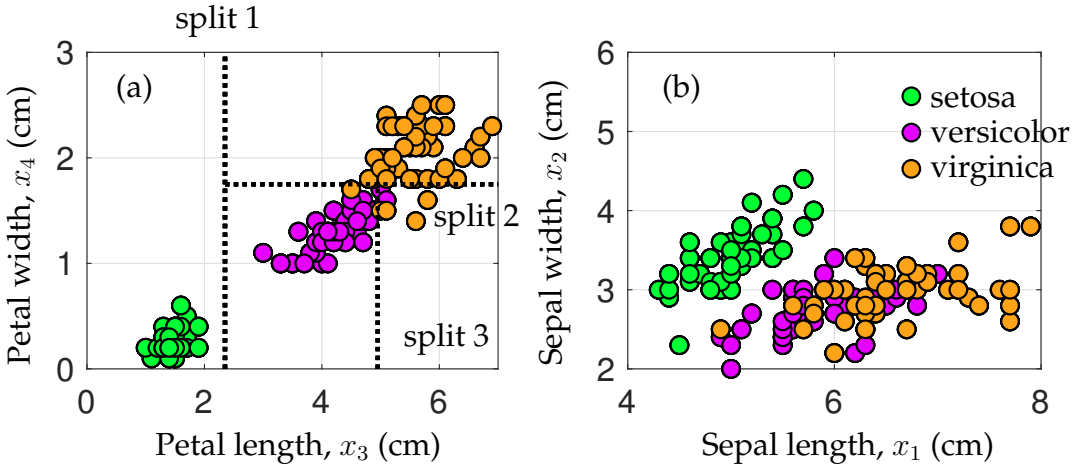


Figure 5.24: Illustration of the splitting procedure for decision tree learning performed on the Fisher iris data set. Each variable x_1 through x_4 is scanned over to determine the best split of data which retains the best correct classification of the labeled data in the split. The variable $x_3 = 2.35$ provides the first split in the data for building a classification tree. This is followed by a second split at $x_4 = 1.75$ and a third split at $x_3 = 4.95$. Only three splits are shown. The classification tree after three splits is shown in Fig. 5.25. Note that, although the setosa data in the x_1 and x_2 direction seems to be well separated along a diagonal line, the decision tree can only split along horizontal and vertical lines.

As a specific example, consider the Fisher iris data set from Fig. 5.1. For this data, each flower had four features (petal width and length, sepal width and length), and three labels (setosa, versicolor, and virginica). There were 50 flowers of each variety for a total of 150 data points. Thus for this data the vector $\mathbf{x}_j$ has the four components

$$x_1 = \text{sepal length}, \tag{5.42a}$$
$$x_2 = \text{sepal width}, \tag{5.42b}$$
$$x_3 = \text{petal length}, \tag{5.42c}$$
$$x_4 = \text{petal width}. \tag{5.42d}$$

The decision tree algorithm scans over these four features in order to decide how to best split the data. Figure 5.24 shows the splitting process in the space of the four variables x_1 through x_4. Illustrated are two data planes containing x_1 versus x_2 (panel (b)) and x_3 versus x_4 (panel (a)). By visual inspection, one can see that the x_3 (petal length) variable maximally separates the data. In fact, the decision tree performs the first split of the data at $x_3 = 2.35$. No further splitting is required to predict setosa, as this first split is sufficient. The variable x_4 then provides the next most promising split at $x_4 = 1.75$. Finally, a third split is performed at $x_3 = 4.95$. Only three splits are shown. This process shows that the splitting procedure has an intuitive appeal, as the data splits optimally

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

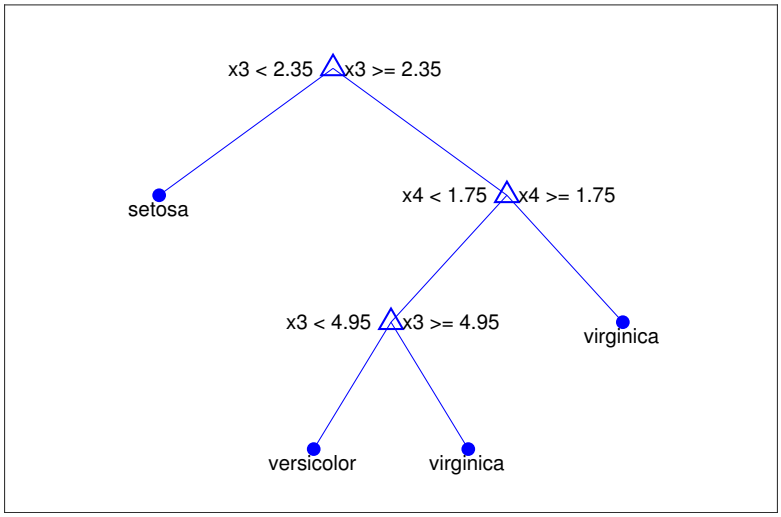


Figure 5.25: Tree structure generated by the MATLAB **fitctree** command. Note that only three splits are conducted, creating a classification tree that produces a class error of 4.67%.

separating the data are clearly visible. Moreover, the splitting does not occur on the x_1 and x_2 (sepal width and length) variables as they do not provide a clear separation of the data. Figure 5.25 shows the tree used for Fig. 5.24.

The following code fits a tree to the Fisher iris data. Note that the **fitctree** command allows for many options, including a cross-validation procedure (used in the code) and parameter tuning (not used in the code).

Code 5.11: [MATLAB] Decision tree classification of Fisher iris data.

```
load fisheriris;
tree=fitctree(meas,species,'MaxNumSplits',3,'CrossVal','on')
view(tree.Trained{1},'Mode','graph');
classError = kfoldLoss(tree)
```

Code 5.11: [Python] Decision tree classification of Fisher iris data.

```
decision_tree = tree.DecisionTreeClassifier(max_depth=3).fit
    (meas,species_label)
tree.export_graphviz(decision_tree, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```

The results of the splitting procedure are demonstrated in Fig. 5.25. The **view** command generates an interactive window showing the tree structure. The tree

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

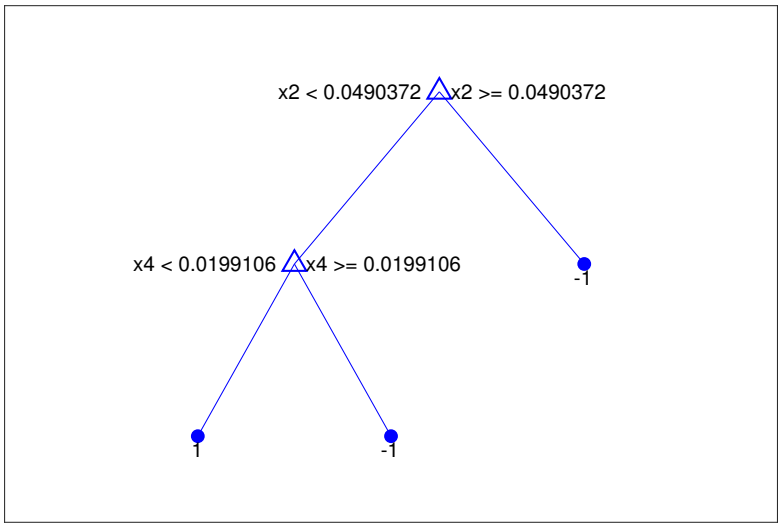


Figure 5.26: Tree structure generated by the MATLAB **fitctree** command for dog versus cat data. Note that only two splits are conducted, creating a classification tree that produces a class error of approximately 16%.

can be pruned and other diagnostics are shown in this interactive graphic format. The class error achieved for the Fisher iris data is 4.67%.

As a second example, we construct a decision tree to classify dogs versus cats using our previously considered wavelet images. Figure 5.26 shows the resulting classification tree. Note that the decision tree learning algorithm identifies the first two splits as occurring along the x_2 and x_4 variables, respectively. These two variables have been considered previously since their histograms show them to be more distinguishable than the other PCA components (see Fig. 5.5). For this splitting, which has been cross-validated, the class error achieved is approximately 16%, which can be compared with the 30% error of LDA.

As a final example, we consider census data that is included in MATLAB. The following code shows some important uses of the classification and regression tree architecture. In particular, the variables included can be used to make associations between relationships. In this case, the various data is used to predict the salary data. Thus, salary is the outcome of the classification. Moreover, the importance of each variable and its relation to salary can be computed, as shown in Fig. 5.27. The following code highlights some of the functionality of the tree architecture.

Code 5.12: [MATLAB] Decision tree classification of census data.

```
load census1994
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

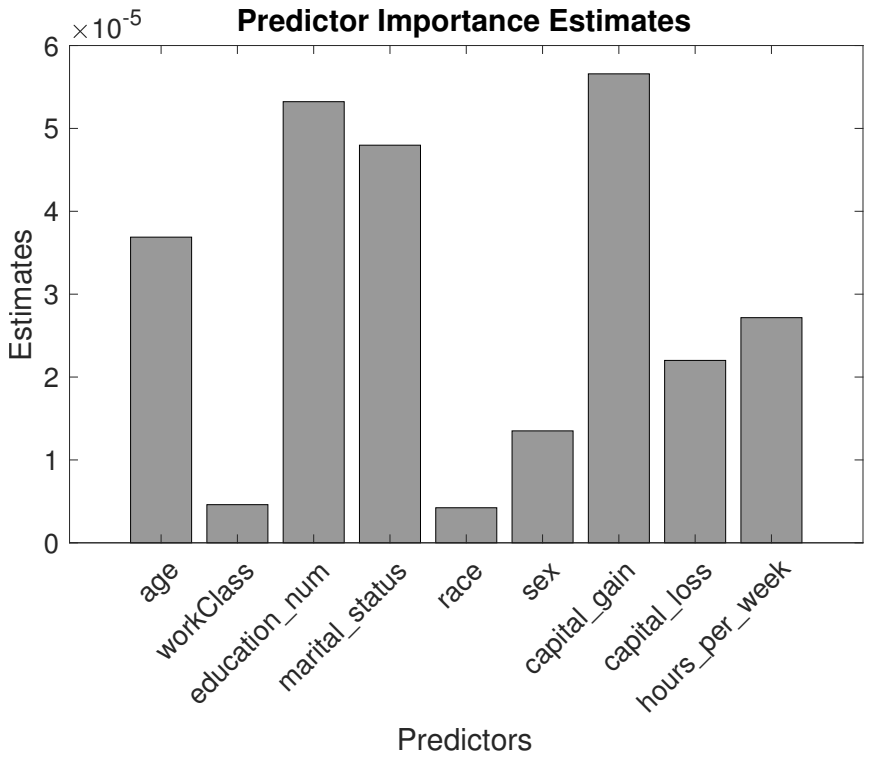


Figure 5.27: Importance of variables for prediction of salary data for the US census of 1994. The classification tree architecture allows for sophisticated treatment of data, including understanding how each variable contributes statistically to predicting a classification outcome.

```
X = adultdata(:,{'age','workClass','education_num','
   marital_status','race','sex','capital_gain',...
    'capital_loss','hours_per_week','salary'});

Mdl = fitctree(X,'salary','PredictorSelection','curvature','
   Surrogate','on');

imp = predictorImportance(Mdl);

bar(imp,'FaceColor',[.6 .6 .6],'EdgeColor','k');
title('Predictor Importance Estimates');
ylabel('Estimates'); xlabel('Predictors'); h = gca;
h.XTickLabel = Mdl.PredictorNames;
h.XTickLabelRotation = 45;
```

Code 5.12: [Python] Decision tree classification of census data.

```
Mdl = tree.DecisionTreeClassifier(max_features=10).fit(
   adultdata_input,adultdata_salary)
imp = Mdl.feature_importances_
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

```
infeatures = ['age','workClass','education_num', '
    marital_status', 'race', 'sex', 'capital_gain','
    capital_loss','hours_per_week']
```

As with the SVM algorithm, there exists a wide variety of tuning parameters for classification trees, and this is a superficial treatment. Overall, such trees are one of the most sophisticated machine learning tools in MATLAB and there are many options that can be executed to tune performance and extract accuracy/cross-validation metrics.

Random Forest Algorithms

Before closing this section, it is important to mention Breiman's *random forest* [108] innovations for decision learning trees. Random forests, or random decision forests, are an ensemble learning method for classification and regression. This is an important innovation, since the decision trees created by splitting are generally not robust to different samples of the data. Thus one can generate two significantly different classification trees with two subsamples of the data. This presents significant challenges for cross-validation. In ensemble learning, a multitude of decision trees are constructed in the training process. The random decision forests correct for a decision trees' habit of overfitting to their training set, thus providing a more robust framework for classification.

There are many variants of the random forest architecture, including variants with *boosting* and *bagging*. These will not be considered here except to mention that the MATLAB **figctree** exploits many of these techniques through its options. One way to think about ensemble learning is that it allows for robust classification trees. It often does this by focusing its training efforts on hard-to-classify data instead of easy-to-classify data. Random forests, bagging, and boosting are all extensive subjects in their own right, but have already been incorporated into leading software offerings that build decision learning trees.

5.9 Top 10 Algorithms of Data Mining circa 2008 (Before the Deep Learning Revolution)

This chapter has illustrated the tremendous diversity of supervised and unsupervised methods available for the analysis of data. Although the algorithms are now easily accessible through many commercial and open-source software packages, the difficulty is now evaluating which method(s) should be used on a given problem. In December 2006, various machine learning experts attending the IEEE International Conference on Data Mining (ICDM) identified the top 10 algorithms for data mining [764]. The identified algorithms were the following: C4.5, k-Means, SVM, Apriori, EM, PageRank, AdaBoost, kNN, Naive

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Bayes, and CART. These top 10 algorithms were identified at the time as being among the most influential data-mining algorithms in the research community. In the summary article, each algorithm was briefly described, along with its impact and potential future directions of research. The 10 algorithms covered classification, clustering, statistical learning, association analysis, and link mining, which are all among the most important topics in data-mining research and development. Interestingly, deep learning and neural networks, which are the topic of the next chapter, are not mentioned in the article. The landscape of data science would change significantly in 2012 with the ImageNet data set, and deep convolutional neural networks (CNN) began to dominate almost any meaningful metric for classification and regression accuracy.

In this section, we highlight their identified top 10 algorithms and the basic mathematical structure of each. Many of them have already been covered in this chapter. This list is not exhaustive, nor does it rank them beyond their inclusion in the top 10 list. Our objective is simply to highlight what was considered by the community as the state-of-the-art data-mining tools in 2008. We begin with those algorithms already considered previously in this chapter.

k-Means

This is one of the workhorse unsupervised algorithms. As already demonstrated, the goal of k-means is simply to cluster by proximity to a set of k points. By updating the locations of the k points according to the mean of the points closest to them, the algorithm iterates to the k-means. The **kmeans** command takes in data $\mathbf{X}$ and the number of prescribed clusters k. It returns labels for each point, **labels**, along with their location, **centers**.

EM (Mixture Models)

Mixture models are the second workhorse algorithm for unsupervised learning. The assumption underlying the mixture models is that the observed data is produced by a mixture of different probability density functions whose weightings are unknown. Moreover, the parameters must be estimated, thus requiring the expectation-maximization (EM) algorithm, where fitting produces Gaussian mixtures to the data $\mathbf{X}$ in k clusters. The **Model** output is a structured variable containing information on the probability distributions (mean, variance, etc.) along with the goodness-of-fit.

Support Vector Machine (SVM)

One of the most powerful and flexible supervised learning algorithms used for most of the 1990s and 2000s, the SVM is an exceptional off-the-shelf method for

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

classification and regression. The main idea is to project the data into higher dimensions and split the data with hyperplanes. Critical to making this work in practice was the kernel trick for efficiently evaluating inner products of functions in higher-dimensional space, where the algorithm takes in labeled training data denoted by **train** and **label**, and produces a structured output, **Model**. The structured output can be used along with the **predict** command to take test data, **test**, and produce labels (**test_labels**). There exist many options and tuning parameters for **fitcsvm**, making it one of the best off-the-shelf methods.

CART (Classification and Regression Tree)

This was the subject of the last section and was demonstrated to provide another powerful technique of supervised learning. The underlying idea was to split the data in a principled and informed way so as to produce an interpretable clustering of the data. The data splitting occurs along a single variable at a time to produce branches of the tree structure, where the algorithm takes in labeled training data denoted by **train** and **label**, and produces a structured output, **tree**. There are many options and tuning parameters for **fitctree**, making it one of the best off-the-shelf methods.

k-Nearest Neighbors (kNN)

This is perhaps the simplest supervised algorithm to understand. It is highly interpretable and easy to execute. Given a new data point $\mathbf{x}_k$ which does not have a label, simply find the k nearest neighbors $\mathbf{x}_j$ with labels $\mathbf{y}_j$. The label of the new point $\mathbf{x}_k$ is determined by a majority vote of the k nearest neighbors. Given a model for the data, the **knnsearch** uses the **Mdl** to label the test data, **test**.

Naive Bayes

The naive Bayes algorithm provides an intuitive framework for supervised learning. It is simple to construct and does not require any complicated parameter estimation, similar to SVM and/or classification trees. It further gives highly interpretable results that are remarkably good in practice. The method is based upon Bayes's theorem and the computation of conditional probabilities. Thus one can estimate the label of a new data point based on the prior probability distributions of the labeled data. The **fitcNativeBayes** command takes in labeled training data denoted by **train** and **label**, and produces a structured output, **Model**. The structured output can be used with the **predict** command to label test data, **test**.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

AdaBoost (Ensemble Learning and Boosting)

AdaBoost is an example of an *ensemble learning* algorithm [251]. Broadly speaking, AdaBoost is a form of random forest [108] which takes into account an ensemble of decision tree models. The way all boosting algorithms work is to first consider an equal weighting for all training data $\mathbf{x}_j$. Boosting re-weights the importance of the data according to how difficult they are to classify. Thus the algorithm focuses on harder-to-classify data. A family of weak learners can be trained to yield a strong learner by boosting the importance of hard-to-classify data [629]. This concept and its usefulness are based upon a seminal theoretical contribution by Kearns and Valiant [378]. The **fitcensemble** command is a general ensemble learner that can do many more things than AdaBoost, including robust boosting and gradient boosting. Gradient boosting is one of the most powerful techniques [252].

C4.5 (Ensemble Learning of Decision Trees)

This algorithm is another variant of decision tree learning developed by J. R. Quinlan [579, 580]. At its core, the algorithm splits the data according to an information entropy score. In its latest versions, it supports boosting as well as many other well-known functionalities to improve performance. Broadly, we can think of this as a strong-performing version of CART. The **fitcensemble** algorithm highlighted with AdaBoost gives a generic ensemble learning architecture that can incorporate decision trees, allowing for a C4.5-like algorithm.

Apriori Algorithm

The last two methods highlighted here tend to focus on different aspects of data mining. In the Apriori algorithm, the goal is to find frequent item sets from data. Although this may sound trivial, it is not, since data sets tend to be very large and can easily produce NP-hard computations because of the combinatorial nature of the algorithms. The Apriori algorithm provides an efficient algorithm for finding frequent item sets using a candidate generation architecture [6]. This algorithm can then be used for fast learning of associate rules in the data.

PageRank

The founding of Google by Sergey Brin and Larry Page revolved around the PageRank algorithm [114]. PageRank produces a static ranking of variables, such as web pages, by computing an offline value for each variable that does not depend on search queries. The PageRank is associated with graph theory, as it originally interpreted a hyperlink from one page to another as a vote. From

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

this, and various modifications of the original algorithm, one can then compute an importance score for each variable and provide an ordered rank list. The number of enhancements for this algorithm is quite large. Producing accurate orderings of variables (web pages) and their importance remains an active topic of research.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Suggested Reading

Texts

(1) Machine learning: A probabilistic perspective, by K. P. Murphy, 2012 [518].

(2) Pattern recognition and machine learning, by C. M. Bishop, 2006 [91].

(3) Pattern classification, by R. O. Duda, P. E. Hart, and D. G. Stork, 2000 [218].

(4) An introduction to statistical learning, by G. James, D. Witten, T. Hastie, and R. Tibshirani, 2013 [348].

(5) Learning with kernels: Support vector machines, regularization, optimization, and beyond, by B. Schölkopf and A. J. Smola, 2002 [643].

(6) Classification and regression trees, by L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen, 1984 [110].

(7) Random forests, by L. Breiman, 2001 [108].

Papers and reviews

(1) Top 10 algorithms in data mining, by X. Wu et al., *Knowledge and Information Systems*, 2008 [764].

(2) The strength of weak learnability, by R. E. Schapire, *Machine Learning*, 1990 [629].

(3) Greedy function approximation: A gradient boosting machine, by J. H. Friedman, *Annals of Statistics*, 2001 [252].

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Homework

Exercise 5-1. Download the MNIST data set (both training and test sets and labels) from `http://yann.lecun.com/exdb/mnist/`. Perform the following analysis:

(a) Do an SVD analysis of the digit images. You will need to reshape each image into a column vector, and each column of your data matrix is a different image.

(b) What does the singular value spectrum look like, and how many modes are necessary for good image reconstruction? (That is, what is the rank r of the digit space?)

(c) What is the interpretation of the $\mathbf{U}$, $\mathbf{\Sigma}$, and $\mathbf{V}$ matrices?

(d) On a 3D plot, project onto three selected $\mathbf{V}$ modes (columns) colored by their digit label, for example, columns 2, 3, and 5.

Once you have performed the above and have your data projected into PCA space, you will build a classifier to identify individual digits in the training set.

(e) Pick two digits. See if you can build a linear classifier (LDA) that can reasonable identify them.

(f) Pick three digits. Try to build a linear classifier to identify these three now.

(g) Which two digits in the data set appear to be the most difficult to separate? Quantify the accuracy of the separation with LDA on the test data.

(h) Which two digits in the data set are most easy to separate? Quantify the accuracy of the separation with LDA on the test data.

(i) SVM (support vector machines) and decision tree classifiers were the state of the art until about 2014. How well do these separate between all 10 digits?

(j) Compare the performance between LDA, SVM, and decision trees on the hardest and easiest pair of digits to separate (from above).

Make sure to discuss the performance of your classifier on both the training and test sets.

Exercise 5-2. Download the two data sets (ORIGINAL IMAGE and CROPPED IMAGES) from Yale Faces B. Your job is to perform an analysis of these data sets. Start with the cropped images and perform the following analysis.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

(a) Do an SVD analysis of the images (where each image is reshaped into a column vector and each column is a new image).

(b) What is the interpretation of the $\mathbf{U}$, $\mathbf{\Sigma}$, and $\mathbf{V}$ matrices?

(c) What does the singular value spectrum look like and how many modes are necessary for good image reconstructions? (That is, what is the rank r of the face space?)

(d) Compare the difference between the cropped (and aligned) versus uncropped images.

Face identification: see if you can build a classifier to identify individuals in the training set.

(e) (Test 1) face classification: Consider the various faces and see if you can build a classifier that can reasonably identify an individual face.

(f) (Test 2) gender classification: Can you build an algorithm capable of recognizing men from women?

(g) (Test 3) unsupervised algorithms: In an unsupervised way, can you develop algorithms that automatically find patters in the faces that naturally cluster?

(Note: You can use any (and hopefully all) of the different clustering and classification methods discussed. Be sure to compare them against each other in these tasks.)

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Chapter 6

Neural Networks and Deep Learning

Neural networks (NNs) were inspired by the Nobel Prize winning work of Hubel and Wiesel on the primary visual cortex of cats [342]. Their seminal experiments showed that neuronal networks were organized in hierarchical layers of cells for processing visual stimuli. The first mathematical model of the NN, termed the Neocognitron in 1980 [260], had many of the characteristic features of today's deep convolutional neural networks (DCNNs), including a multi-layer structure, convolution, max pooling, and nonlinear dynamical nodes. The recent success of DCNNs in computer vision has been enabled by two critical components: (i) the continued growth of computational power, and (ii) exceptionally large labeled data sets which take advantage of the power of a *deep* multi-layer architecture. Indeed, although the theoretical inception of NNs has an almost four-decade history, the analysis of the ImageNet data set in 2012 [414] provided a watershed moment for NNs and deep learning [432]. Prior to this data set, there were a number of data sets available with approximately tens of thousands of labeled images. ImageNet provided over 15 million labeled, high-resolution images with over 22 000 categories. DCNNs, which are only one potential category of NNs, have since transformed the field of computer vision by dominating the performance metrics in almost every meaningful computer vision task intended for classification and identification.

Although ImageNet has been critically enabling for the field, NNs were textbook material in the early 1990s, with a focus typically on a small number of layers. Critical machine learning tasks such as principal component analysis (PCA) were shown to be intimately connected with networks that included backpropagation. Importantly, there were a number of critical innovations which established multi-layer feedforward networks as a class of universal approximators [338]. The past decade has seen tremendous advances in NN architectures, many designed and tailored for specific application areas. Innovations have come from algorithmic modifications that have led to significant performance gains in a variety of fields. These innovations include pre-training, dropout, inception modules, data augmentation with virtual examples, batch

normalization, and/or residual learning (see Goodfellow et al. [290] for a detailed exposition of NNs). This is only a partial list of potential algorithmic innovations, thus highlighting the continuing and rapid pace of progress in the field. Remarkably, NNs were not even listed as one of the top 10 algorithms of data mining in 2008 [764]. But a decade later, the undeniable and growing list of successes of NNs on challenge data sets make them perhaps the most important data-mining tool for our emerging generation of scientists and engineers.

As already shown in the last two chapters, all of machine learning revolves fundamentally around optimization. NNs specifically optimize over a compositional function

$$\underset{\mathbf{A}_j}{\operatorname{argmin}}(f_M(\mathbf{A}_M, \ldots, f_2(\mathbf{A}_2, f_1(\mathbf{A}_1, \mathbf{x}))\ldots) + \lambda g(\mathbf{A}_j)), \tag{6.1}$$

which is often solved using stochastic gradient descent and backpropagation algorithms. Each matrix $\mathbf{A}_k$ denotes the weights connecting the neural network from the kth to the $(k+1)$th layer. It is a massively under-determined system which is regularized by $g(\mathbf{A}_j)$. Composition and regularization are critical for generating expressive representations of the data and preventing overfitting, respectively. The notation used in (6.1) is motivated from solving linear systems $\mathbf{A}\mathbf{x} = \mathbf{b}$ through regression. This will be highlighted in the first few sections of this chapter. We will then move to a broader framework of mapping input data $\mathbf{X}$ to output data $\mathbf{Y}$ using a model $\mathbf{f}(\cdot)$. Thus we will represent (6.1) in deep learning models as

$$\underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}), \tag{6.2}$$

where $\boldsymbol{\theta}$ are the neural network weights and $\mathbf{f}(\cdot)$ characterizes the network (number of layers, structure, regularizers). Thus we will move to this notation in the second half of this chapter as a generic representation of a neural net. This general optimization framework is at the center of deep learning algorithms, and its solution will be considered in this chapter. Importantly, NNs have significant potential for overfitting of data so that cross-validation must be carefully considered. Recall that: *if you do not cross-validate, you is dumb.*

6.1 Neural Networks: Single-Layer Networks

The generic architecture of a multi-layer NN is shown in Fig. 6.1. For classification tasks, the goal of the NN is to map a set of input data to a classification. Specifically, we train the NN to accurately map the data $\mathbf{x}_j$ to their correct label $\mathbf{y}_j$. As shown in Fig. 6.1, the input space has the dimension of the raw data $\mathbf{x}_j \in \mathbb{R}^n$. The output layer has the dimension of the designed classification space. Constructing the output layer will be discussed further in the following.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

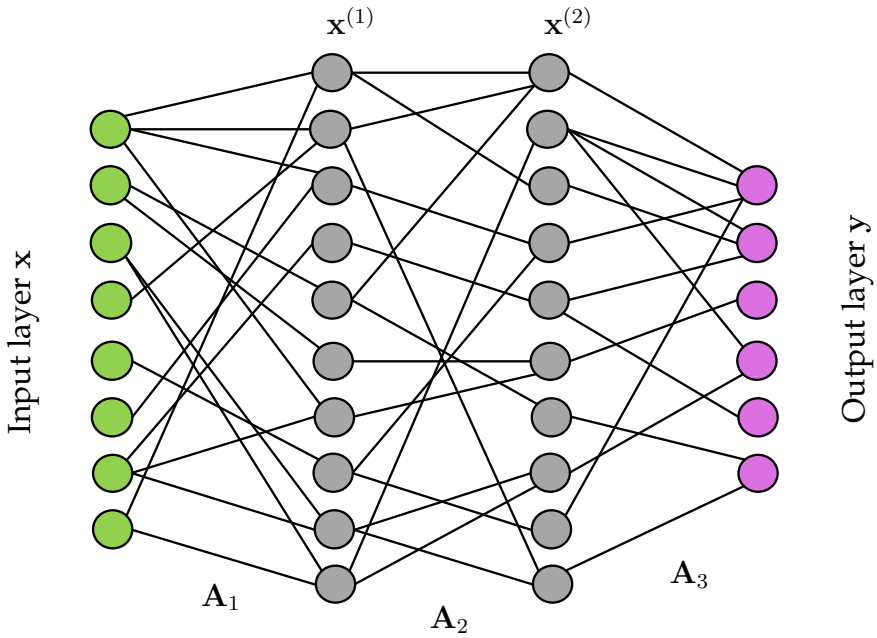


Figure 6.1: Illustration of a neural net architecture mapping an input layer $\mathbf{x}$ to an output layer $\mathbf{y}$. The middle (hidden) layers are denoted $\mathbf{x}^{(j)}$ where j determines their sequential ordering. The matrices $\mathbf{A}_j$ contain the coefficients that map each variable from one layer to the next. Although the dimensionality of the input layer $\mathbf{x} \in \mathbb{R}^n$ is known, there is great flexibility in choosing the dimension of the inner layers as well as how to structure the output layer. The number of layers and how to map between layers is also selected by the user. This flexible architecture gives great freedom in building a good classifier.

Immediately, one can see that there are a great number of design questions regarding NNs. How many layers should be used? What should be the dimension of the layers? How should the output layer be designed? Should one use all-to-all or sparsified connections between layers? How should the mapping between layers be performed: a *linear mapping* or a *nonlinear mapping*? Much like the tuning options on SVM and classification trees, NNs have a significant number of design options that can be tuned to improve performance.

Initially, we consider the mapping between layers of Fig. 6.1. We denote the various layers between input and output as $\mathbf{x}^{(k)}$, where k is the layer number. For a linear mapping between layers, the following relations hold

$$\mathbf{x}^{(1)} = \mathbf{A}_1\mathbf{x}, \tag{6.3a}$$
$$\mathbf{x}^{(2)} = \mathbf{A}_2\mathbf{x}^{(1)}, \tag{6.3b}$$
$$\mathbf{y} = \mathbf{A}_3\mathbf{x}^{(2)}. \tag{6.3c}$$

This forms a compositional structure so that the mapping between input and

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

output can be represented as

$$\mathbf{y} = \mathbf{A}_3\mathbf{A}_2\mathbf{A}_1\mathbf{x}. \tag{6.4}$$

This basic architecture can scale to M layers, so that a general representation between input data and the output layer for a linear NN is given by

$$\mathbf{y} = \mathbf{A}_M\mathbf{A}_{M-1}\cdots\mathbf{A}_2\mathbf{A}_1\mathbf{x}. \tag{6.5}$$

This is generally a highly under-determined system that requires some constraints on the solution in order to select a unique solution. One constraint is immediately obvious: the mapping must generate M distinct matrices that give the best mapping. It should be noted that linear mappings, even with a compositional structure, can only produce a limited range of functional responses due to the limitations of the linearity.

Nonlinear mappings are also possible, and generally used, in constructing the NN. Indeed, nonlinear activation functions allow for a richer set of functional responses than their linear counterparts. In this case, the connections between layers are given by

$$\mathbf{x}^{(1)} = f_1(\mathbf{A}_1, \mathbf{x}), \tag{6.6a}$$
$$\mathbf{x}^{(2)} = f_2(\mathbf{A}_2, \mathbf{x}^{(1)}), \tag{6.6b}$$
$$\mathbf{y} = f_3(\mathbf{A}_3, \mathbf{x}^{(2)}). \tag{6.6c}$$

Note that we have used different nonlinear functions $f_j(\cdot)$ between layers. Often a single function is used; however, there is no constraint that this is necessary. In terms of mapping the data between input and output over M layers, the following is derived:

$$\mathbf{y} = f_M(\mathbf{A}_M, \ldots, f_2(\mathbf{A}_2, f_1(\mathbf{A}_1, \mathbf{x}))\ldots), \tag{6.7}$$

which can be compared with (6.1) for the general optimization which constructs the NN. As a highly under-determined system, constraints should be imposed in order to extract a desired solution type, as in (6.1). For big data applications such as ImageNet and computer vision tasks, the optimization associated with this compositional framework is expensive given the number of variables that must be determined. However, for moderate-sized networks, it can be performed on workstation and laptop computers. Modern stochastic gradient descent and backpropagation algorithms enable this optimization, and both are covered in later sections.

A Single-Layer Network

To gain insight into how an NN might be constructed, we will consider a single-layer network that is optimized to build a classifier between dogs and cats.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

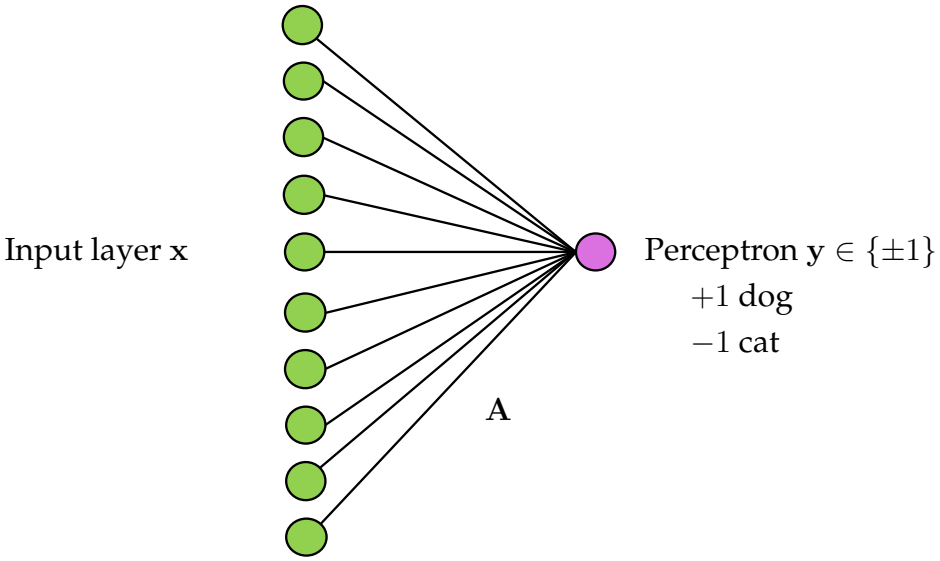


Figure 6.2: Single-layer network for binary classification between dogs and cats. The output layer for this case is a perceptron with $\mathbf{y} \in \{\pm 1\}$. A linear mapping between the input image space and output layer can be constructed for training data by solving $\mathbf{A} = \mathbf{Y}\mathbf{X}^\dagger$. This gives a least-squares regression for the matrix $\mathbf{A}$ mapping the images to label space.

The dogs and cats example was considered extensively in the previous chapter. Recall that we were given images of dogs and cats, or a wavelet version of dogs and cats. Figure 6.2 shows our construction. To make this as simple as possible, we consider the simple NN output

$$\mathbf{y} = \{\text{dog}, \text{cat}\} = \{+1, -1\}, \tag{6.8}$$

which labels each data vector with an output $\mathbf{y} \in \{\pm 1\}$. In this case the output layer is a single node. As in previous supervised learning algorithms, the goal is to determine a mapping so that each data vector $\mathbf{x}_j$ is labeled correctly by $\mathbf{y}_j$.

The easiest mapping is a linear mapping between the input images $\mathbf{x}_j \in \mathbb{R}^n$ and the output layer. This gives a linear system $\mathbf{AX} = \mathbf{Y}$ of the form

$$\mathbf{AX} = \mathbf{Y} \quad \Longrightarrow$$
$$[a_1\ a_2\ \cdots\ a_n] \begin{bmatrix} | & | & & | \\ \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_p \\ | & | & & | \end{bmatrix} = [+1\ +1\ \cdots\ -1\ -1], \tag{6.9}$$

where each column of the matrix $\mathbf{X}$ is a dog or a cat image and the columns of $\mathbf{Y}$ are its corresponding labels. Since the output layer is a single node, both $\mathbf{A}$ and $\mathbf{Y}$ reduce to vectors. In this case, our goal is to determine the matrix (vector) $\mathbf{A}$ with components a_j. The simplest solution is to take the pseudo-inverse of the data matrix $\mathbf{X}$:

$$\mathbf{A} = \mathbf{Y}\mathbf{X}^\dagger. \tag{6.10}$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Thus a single output layer allows us to build a NN using least-squares fitting. Of course, we could also solve this linear system in a variety of other ways, including with sparsity-promoting methods. The following code solves this problem through both least-squares fitting (**pinv**) and the LASSO.

Code 6.1: [MATLAB] Single-layer, linear neural network.

```
test=[dog_wave(:,61:80) cat_wave(:,61:80)];
label=[ones(60,1); -1*ones(60,1)].';

A=label*pinv(train); test_labels=sign(A*test);
A=lasso(train.',label.','Lambda',0.1).';
test_labels=sign(A*test);
```

Code 6.1: [Python] Single-layer, linear neural network.

```
train = np.concatenate((dog_wave[:,:60],cat_wave[:,:60]),
   axis=1)
test = np.concatenate((dog_wave[:,60:80],cat_wave[:,60:80]),
   axis=1)
label = np.repeat(np.array([1,-1]),60)

A = label @ np.linalg.pinv(train)
test_labels = np.sign(A@test)

lasso = linear_model.Lasso().fit(train.T,label)
A_lasso = lasso.coef_
test_labels_lasso = np.sign(A_lasso@test)
```

Figures 6.3 and 6.4 show the results of this linear single-layer NN with single-node output layer. Specifically, the four rows of Fig. 6.3 show the output layer on the withheld test data for both the pseudo-inverse and LASSO methods along with a bar graph of the 32×32 (1024 pixels) weightings of the matrix $\mathbf{A}$. Note that all matrix elements are non-zero in the pseudo-inverse solution, while the LASSO highlights a small number of pixels that can classify the pictures as well as using all pixels. Figure 6.4 shows the matrix $\mathbf{A}$ for the two solution strategies reshaped into 32×32 images. Note that, for the pseudo-inverse, the weightings of the matrix elements $\mathbf{A}$ show many features of the cat and dog faces. For the LASSO method, only a few pixels are required that are clustered near the eyes and ears. Thus for this single-layer network, interpretable results are achieved by looking at the weights generated in the matrix $\mathbf{A}$.

6.2 Multi-Layer Networks and Activation Functions

The previous section constructed what is perhaps the simplest NN possible. It was linear, had a single layer, and a single output layer neuron. The potential

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

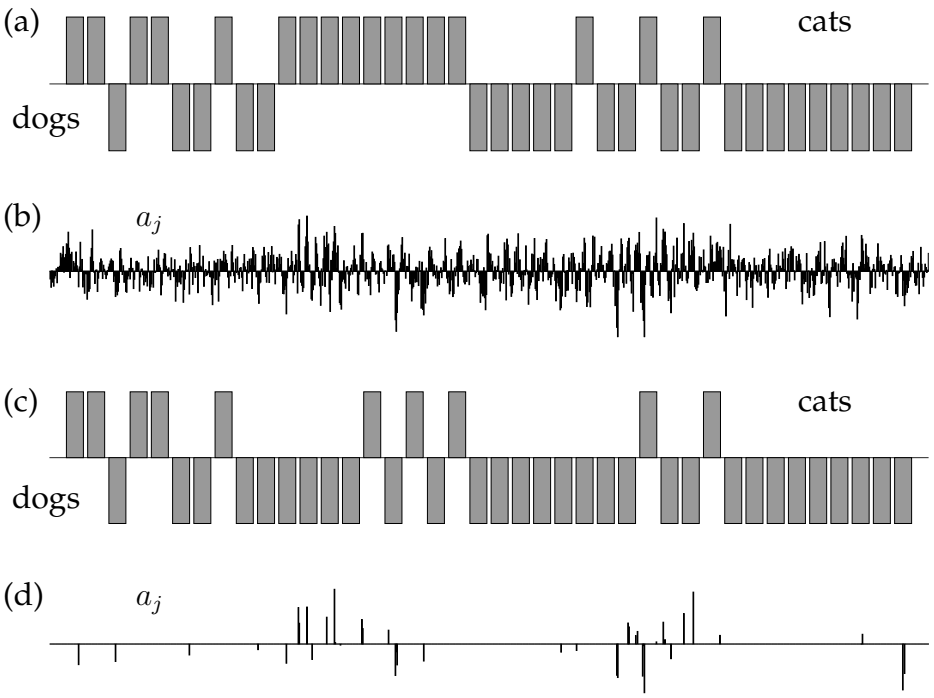


Figure 6.3: Classification of withheld data tested on a trained, single-layer network with linear mapping between inputs (pixel space) and a single output. Panels (a) and (c) are the bar graphs of the output layer score $\mathbf{y} \in \{\pm 1\}$ achieved for the withheld data using a pseudo-inverse for training and the LASSO for training, respectively. The results show in both cases that dogs are more often misclassified than cats are misclassified. Panels (b) and (d) show the coefficients of the matrix $\mathbf{A}$ for the pseudo-inverse and LASSO, respectively. Note that the LASSO has only a small number of non-zero elements, thus suggesting that the NN is highly sparse.

generalizations are endless, but we will focus on two simple extensions of the NN in this section. The first extension concerns the assumption of linearity in which we assumed that there is a linear transform from the image space to the output layer: $\mathbf{Ax} = \mathbf{y}$ in (6.9). We highlight here common nonlinear transformations from input to output space represented by

$$\mathbf{y} = f(\mathbf{A}, \mathbf{x}) \tag{6.11}$$

where $f(\cdot)$ is a specified *activation function* (transfer function) for our mapping.

The linear mapping used previously, although simple, does not offer the flexibility and performance that other mappings offer. Some standard activa-

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

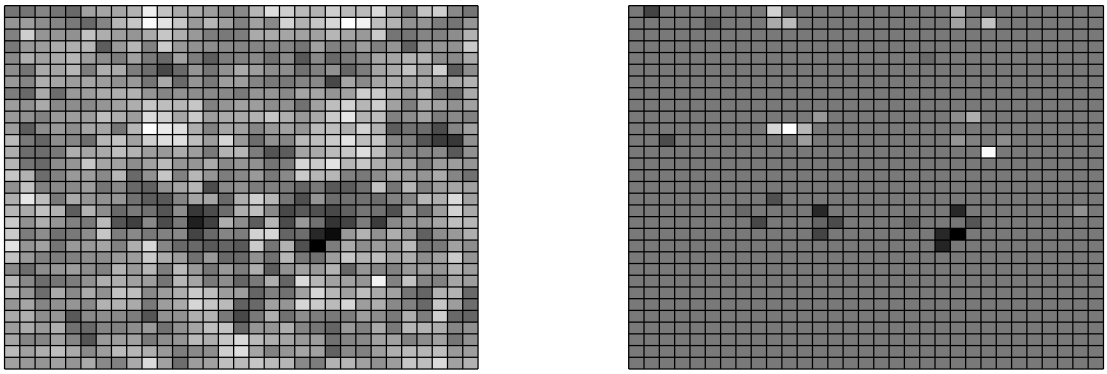


Figure 6.4: Weightings of the matrix $\mathbf{A}$ reshaped into 32×32 arrays. The left matrix shows the matrix $\mathbf{A}$ computed by least-squares regression (the pseudo-inverse) while the right matrix shows the matrix $\mathbf{A}$ computed by LASSO. Both matrices provide similar classification scores on withheld data. They further provide interpretability in the sense that the results from the pseudo-inverse show many of the features of dogs and cats while the LASSO shows that measuring near the eyes and ears alone can give the features required for distinguishing between dogs and cats.

tion functions are given by

$$f(x) = x, \qquad \text{linear}, \tag{6.12a}$$

$$f(x) = \begin{cases} 0 & \text{for } x \leq 0, \\ 1 & \text{for } x > 0, \end{cases} \qquad \text{binary step}, \tag{6.12b}$$

$$f(x) = \frac{1}{1+\exp(-x)}, \qquad \text{logistic (soft step)}, \tag{6.12c}$$

$$f(x) = \tanh(x), \qquad \text{tanh}, \tag{6.12d}$$

$$f(x) = \begin{cases} 0 & \text{for } x \leq 0, \\ x & \text{for } x > 0, \end{cases} \qquad \text{rectified linear unit (ReLU)}. \tag{6.12e}$$

There are other possibilities, but these are perhaps the most commonly considered in practice and they will serve for our purposes. Importantly, the chosen function $f(x)$ will be differentiated in order to be used in gradient descent algorithms for optimization. Each of the functions above is either differentiable or piecewise differentiable. Perhaps the most commonly used activation function is currently the ReLU, which we denote $f(x) = \text{ReLU}(x)$.

With a nonlinear activation function $f(x)$, or if there is more than one layer, then standard linear optimization routines such as the pseudo-inverse and LASSO can no longer be used. Although this may not seem immediately significant, recall that we are optimizing in a high-dimensional space where each entry of the matrix $\mathbf{A}$ needs to be found through optimization. Even moderate to

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

small problems can be computationally expensive to solve without using specialty optimization methods. Fortunately, the two dominant optimization components for training NNs, stochastic gradient descent (SGD) and backpropagation (backprop), are included with the neural network function calls in MATLAB. As these methods are critically enabling, both of them are considered in detail in the next two sections of this chapter.

Multiple layers can also be considered as shown in (6.5) and (6.6c). In this case, the optimization must simultaneously identify multiple connectivity matrices $\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_M$, in contrast to the linear case, where only a single matrix is determined, $\bar{\mathbf{A}} = \mathbf{A}_M \cdots \mathbf{A}_2\mathbf{A}_1$. The multiple-layer structure significantly increases the size of the optimization problem, as each matrix element of the M matrices must be determined. Even for a single-layer structure, an optimization routine such as **fminsearch** will be severely challenged when considering a nonlinear transfer function, and one needs to move to a gradient descent-based algorithm.

MATLAB's neural network toolbox, much like TensorFlow in Python, has a wide range of features, which makes it exceptionally powerful and convenient for building NNs. In the following code, we will train a NN to classify between dogs and cats as in the previous example. However, in this case, we allow the single layer to have a nonlinear transfer function that maps the input to the output layer. The output layer for this example will be modified to the following:

$$\mathbf{y} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \{\text{dog}\} \quad \text{and} \quad \mathbf{y} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \{\text{cat}\}. \tag{6.13}$$

Half of the data is extracted for training, while the other half is used for testing the results. The following code builds a network using the **train** command to classify between our images.

Code 6.2: [MATLAB] Neural network with nonlinear transfer functions.

```
net = patternnet(2,'trainscg');
net.layers{1}.transferFcn = 'tansig';

net = train(net,x,label);
```

In the code above, the **patternnet** command builds a classification network with two outputs (6.13). It also optimizes with the option **trainscg** which is a *scaled conjugate gradient backpropagation*. The **net.layers** also allows us to specify the transfer function, in this case hyperbolic tangent functions (6.12ed). The **view(net)** command produces a diagnostic tool shown in Fig. 6.5 that summarizes the optimization and NN.

The results of the classification for a cross-validated training set as well as a withhold set are shown in Fig. 6.6. Specifically, the desired outputs are given by the vectors (6.13). For both the training and withhold sets, the two components

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

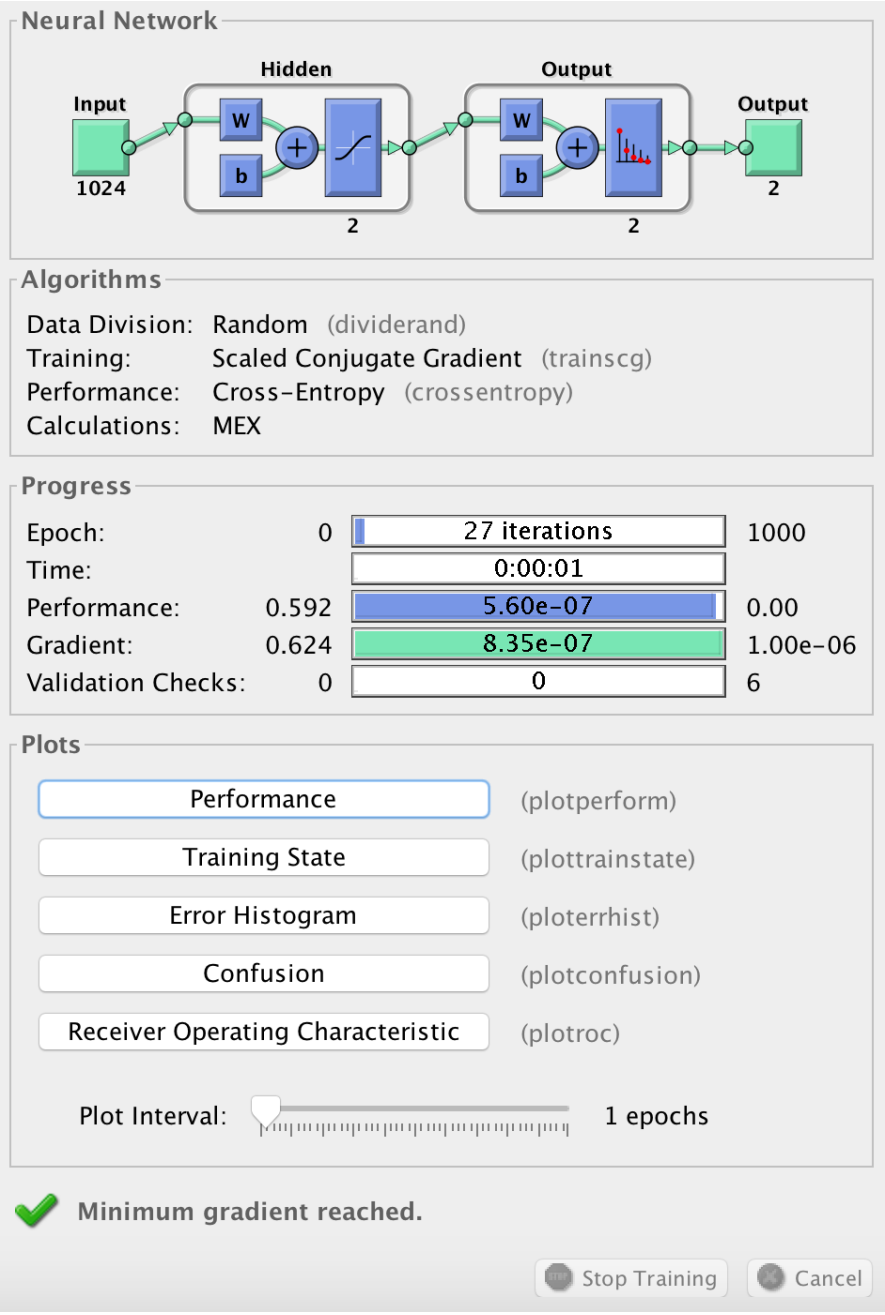


Figure 6.5: MATLAB neural network visualization tool. The number of iterations along with the performance can all be accessed from the interactive graphical tool. The performance, error histogram, and confusion buttons produce Figs. 6.7–6.9, respectively.

of the vector are shown for the 80 training images (40 cats and 40 dogs) and the 80 withheld images (40 cats and 40 dogs). The training set produces a perfect classifier using a single-layer network with a hyperbolic tangent transfer func-

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

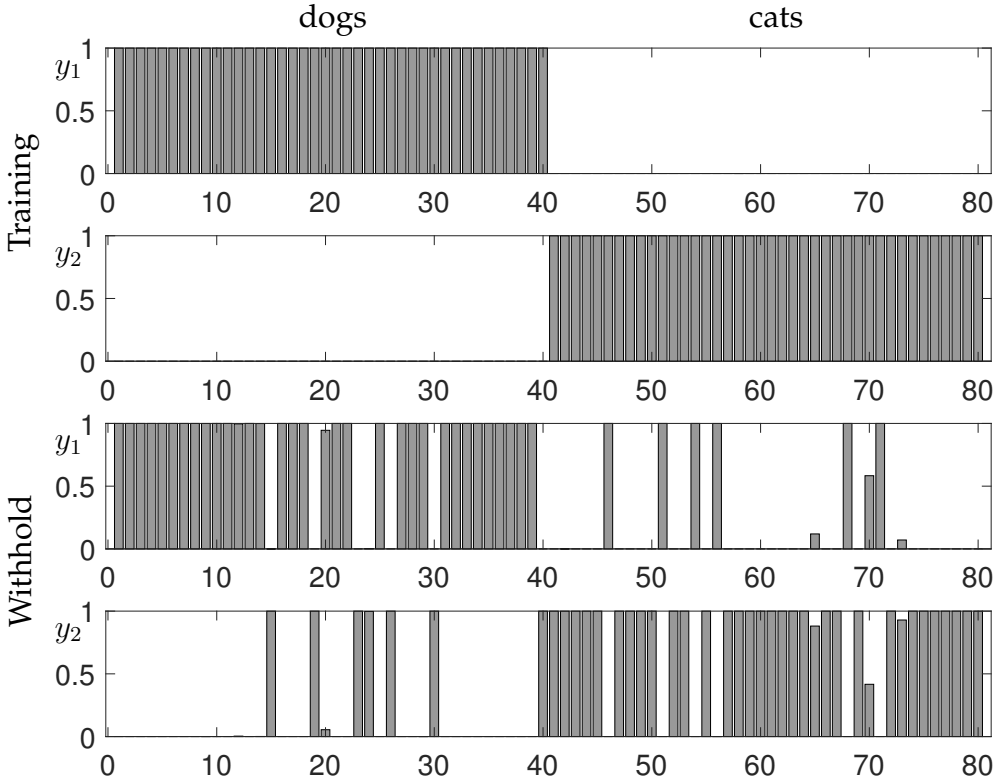


Figure 6.6: Comparison of the output vectors $\mathbf{y} = [y_1 \ y_2]^T$, which are ideally (6.13) for the dogs and cats considered here. The NN training stage produces a cross-validated classifier that achieves 100% accuracy in classifying the training data (top two panels for 40 dogs and 40 cats). When applied to a withheld set, 85% accuracy is achieved (bottom two panels for 40 dogs and 40 cats).

tion (6.12ed). On the withheld data, it incorrectly identifies six of 40 dogs and cats, yielding an accuracy of $\approx 85\%$ on new data.

The diagnostic tool shown in Fig. 6.5 allows access to a number of features critical for evaluating the NN. Figure 6.7 is a summary of the performance achieved by the NN training tool. In this figure, the training algorithm automatically breaks the data into a training, validation, and test set. The backpropagation-enabled, stochastic gradient descent optimization algorithm then iterates through a number of training epochs until the cross-validated error achieves a minimum. In this case, 22 epochs are sufficient to achieve a minimum. The error on the test set is significantly higher than what is achieved for cross-validation. For this case, only a limited amount of data is used for training (40 dogs and 40 cats), thus making it difficult to achieve great performance. Regardless, as already shown, once the algorithm has been trained, it can be used to evaluate new data as shown in Fig. 6.6.

There are two other features easily available with the NN diagnostic tool of Fig. 6.5. Figure 6.8 shows an error histogram associated with the trained net-

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

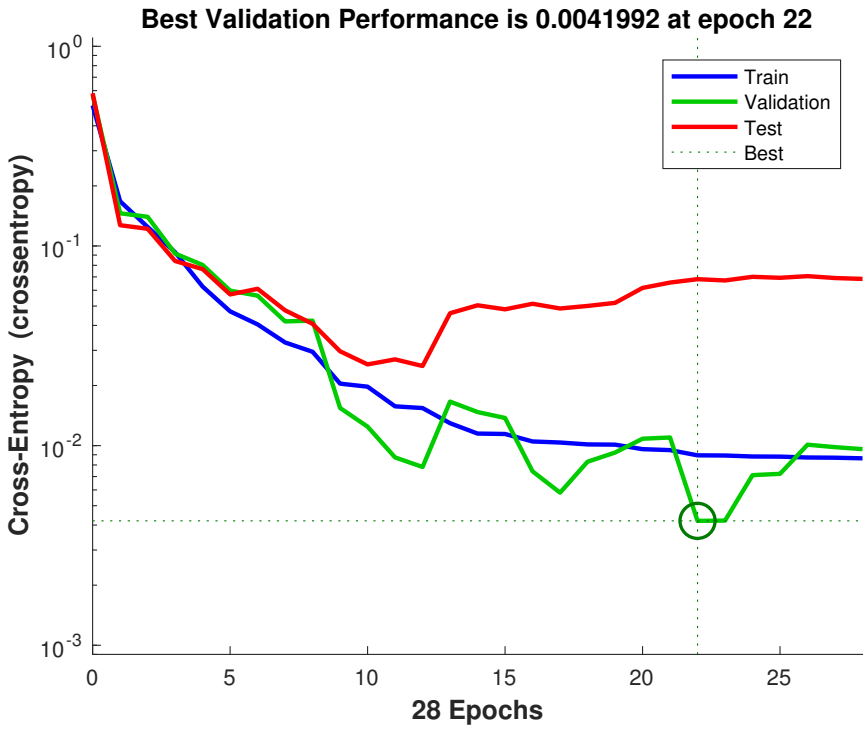


Figure 6.7: Summary of training of the NN over a number of epochs. The NN architecture automatically separates the data into training, validation, and test sets. The training continues (with a maximum of 1000 epochs) until the validation error curve hits a minimum. The training then stops and the trained algorithm is then used on the test set to evaluate performance. The NN trained here has only a limited amount of data (40 dogs and 40 cats), thus limiting the performance. This figure is accessed with the **performance** button on the NN interactive tool of Fig. 6.6.

work. As with Fig. 6.7, the data is divided into training, validation, and test sets. This provides an overall assessment of the classification quality that can be achieved by the NN training algorithm. Another view of the performance can be seen in the confusion matrices for the training, validation, and test data. This is shown in Fig. 6.9. Overall, between Figs. 6.7 to 6.9, high-quality diagnostic tools are available to evaluate how well the NN is able to achieve its classification task. The performance limits are easily seen in these figures.

6.3 The Backpropagation Algorithm

As was shown for the NNs of the last two sections, training data is required to determine the weights of the network. Specifically, the network weights are determined so as to best classify dog versus cat images. In the single-layer

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

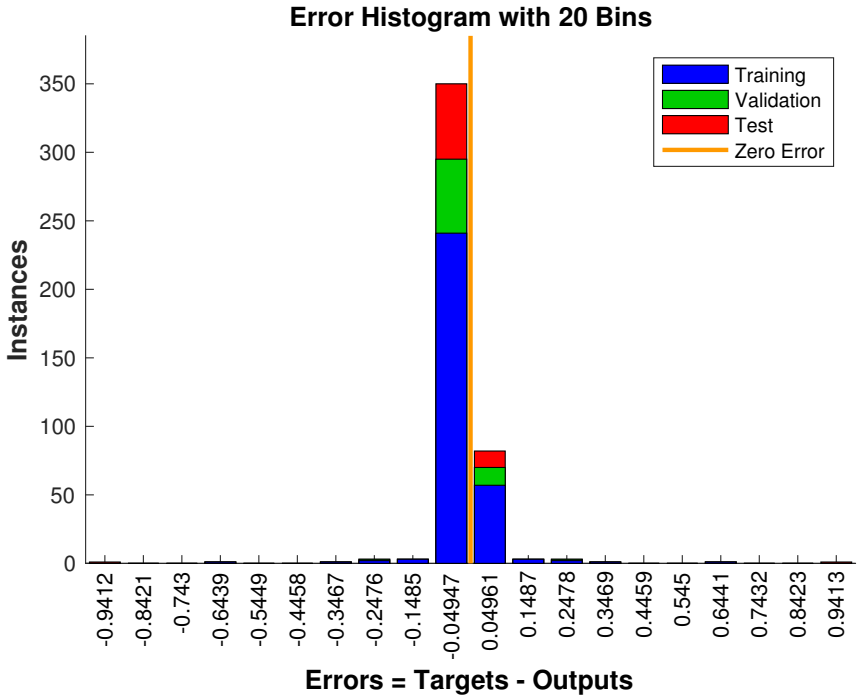


Figure 6.8: Summary of the error performance of the NN architecture for training, validation, and test sets. This figure is accessed with the **errorhistogram** button on the NN interactive tool of Fig. 6.6.

network, this was done using both least-squares regression and LASSO. This shows that, at its core, an optimization routine and objective function are required to determine the weights. The objective function should minimize a measure of the misclassified images. The optimization, however, can be modified by imposing a regularizer or constraints, such as the ℓ_1 penalization in LASSO.

In practice, the objective function chosen for optimization is not the true objective function desired, but rather a proxy for it. Proxies are chosen largely due to the ability to differentiate the objective function in a computationally tractable manner. There are also many different objective functions for different tasks. Instead, one often considers a suitably chosen loss function so as to approximate the true objective. Ultimately, computational tractability is critical for training NNs.

The backpropagation algorithm (backprop) exploits the compositional nature of NNs in order to frame an optimization problem for determining the weights of the network. Specifically, it produces a formulation amenable to standard gradient descent optimization (see Section 4.2). Specifically, backprop calculates the gradient of the error, which is then used for gradient descent. Backprop relies on a simple mathematical principle: the chain rule for differentiation. Moreover, it can be proven that the computational time required to

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.



Figure 6.9: Summary of the error performance through confusion matrices of the NN architecture for training, validation, and test sets. This figure is accessed with the **confusion** button on the NN interactive tool of Fig. 6.6.

evaluate the gradient is within a factor of 5 of the time required for computing the actual function itself [59]. This is known as the Baur–Strassen theorem. Figure 6.10 gives the simplest example of backprop and how the gradient descent is to be performed. The input-to-output relationship for this single-node, one-hidden-layer network is given by

$$y = g(z, b) = g(f(x, a), b). \tag{6.14}$$

Thus, given functions $f(\cdot)$ and $g(\cdot)$ with weighting constants a and b, the output error produced by the network can be computed against the ground truth as

$$E = \frac{1}{2}(y_0 - y)^2, \tag{6.15}$$

where y_0 is the correct output and y is the NN approximation to the output. The goal is to find a and b to minimize the error. The minimization requires

$$\frac{\partial E}{\partial a} = -(y_0 - y)\frac{\mathrm{d}y}{\mathrm{d}z}\frac{\mathrm{d}z}{\mathrm{d}a} = 0. \tag{6.16}$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

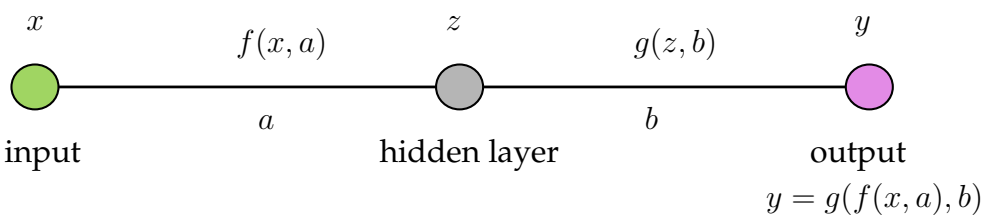


Figure 6.10: Illustration of the backpropagation algorithm on a one-node, one-hidden-layer network. The compositional nature of the network gives the input–output relationship $y = g(z, b) = g(f(x, a), b)$. By minimizing the error between the output y and its desired output y_0, the composition along with the chain rule produces an explicit formula (6.16) for updating the values of the weights. Note that the chain rule backpropagates the error all the way through the network. Thus, by minimizing the output, the chain rule acts on the compositional function to produce a product of derivative terms that advance backward through the network.

A critical observation is that the compositional nature of the network along with the chain rule forces the optimization to backpropagate error through the network. In particular, the terms $(\mathrm{d}y/\mathrm{d}z)(\mathrm{d}z/\mathrm{d}a)$ show how this backprop occurs. Given functions $f(\cdot)$ and $g(\cdot)$, the chain rule can be explicitly computed.

Backprop results in an iterative, gradient descent update rule:

$$a_{k+1} = a_k - \delta \frac{\partial E}{\partial a_k}, \tag{6.17a}$$

$$b_{k+1} = b_k - \delta \frac{\partial E}{\partial b_k}, \tag{6.17b}$$

where δ is the so-called learning rate and $\partial E/\partial a$ along with $\partial E/\partial b$ can be explicitly computed using (6.16). The iteration algorithm is executed to convergence. As with all iterative optimization, a good initial guess is critical to achieve a good solution in a reasonable amount of computational time.

Backprop proceeds as follows: (i) A NN is specified along with a labeled training set. (ii) The initial weights of the network are set to random values. Importantly, one must not initialize the weights to zero, similar to what may be done in other machine learning algorithms. If weights are initialized to zero, after each update, the outgoing weights of each neuron will be identical, because the gradients will be identical. Moreover, NNs often get stuck at local optima where the gradient is zero but that are not global minima, so random weight initialization allows one to have a chance of circumventing this by starting at many different random values. (iii) The training data is run through the network to produce an output $\mathbf{y}$, whose ideal ground-truth output is $\mathbf{y}_0$. The derivatives with respect to each network weight are then computed using backprop formulas (6.16). (iv) For a given learning rate δ, the network weights are

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

updated as in (6.17). (v) We return to step (iii) and continue iterating until a maximum number of iterations is reached or convergence is achieved.

As a simple example, consider the linear activation function

$$f(\xi, \alpha) = g(\xi, \alpha) = \alpha\xi. \tag{6.18}$$

In this case we have in Fig. 6.10:

$$z = ax, \tag{6.19a}$$
$$y = bz. \tag{6.19b}$$

We can now explicitly compute the gradients such as (6.16). This gives

$$\frac{\partial E}{\partial a} = -(y_0 - y)\frac{\mathrm{d}y}{\mathrm{d}z}\frac{\mathrm{d}z}{\mathrm{d}a} = -(y_0 - y) \cdot b \cdot x, \tag{6.20a}$$
$$\frac{\partial E}{\partial b} = -(y_0 - y)\frac{\mathrm{d}y}{\mathrm{d}b} = -(y_0 - y)z = -(y_0 - y) \cdot a \cdot x. \tag{6.20b}$$

Thus, with the current values of a and b, along with the input–output pair x and y and target truth y_0, each derivative can be evaluated. This provides the required information to perform the update (6.17).

The backprop for a deeper net follows in a similar fashion. Consider a network with M hidden layers labeled z_1 to z_m, with the first connection weight a between x and z_1. The generalization of Fig. 6.10 and (6.16) is given by

$$\frac{\partial E}{\partial a} = -(y_0 - y)\frac{\mathrm{d}y}{\mathrm{d}z_m}\frac{\mathrm{d}z_m}{\mathrm{d}z_{m-1}} \cdots \frac{\mathrm{d}z_2}{\mathrm{d}z_1}\frac{\mathrm{d}z_1}{\mathrm{d}a}. \tag{6.21}$$

The cascade of derivatives induced by the composition and chain rule highlights the backpropagation of errors that occurs when minimizing the classification error.

A full generalization of backprop involves multiple layers as well multiple nodes per layer. The general situation is illustrated in Fig. 6.1. The objective is to determine the matrix elements of each matrix $\mathbf{A}_j$. Thus a significant number of network parameters need to be updated in gradient descent. Indeed, training a network can often be computationally infeasible even though the update rules for individual weights are not difficult. NNs can thus suffer from the curse of dimensionality, as each matrix from one layer to another requires updating n^2 coefficients for an n-dimensional input, assuming the two connected layers are both n-dimensional.

Denoting all the weights to be updated by the vector $\mathbf{w}$, where $\mathbf{w}$ contains all the elements of the matrices $\mathbf{A}_j$ illustrated in Fig. 6.1, then

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \delta \nabla E, \tag{6.22}$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

where the gradient of the error ∇E, through the composition and chain rule, produces the backpropagation algorithm for updating the weights and reducing the error. Expressed in a component-by-component way:

$$w_{k+1}^j = w_k^j - \delta \frac{\partial E}{\partial w_k^j}, \tag{6.23}$$

where this equation holds for the jth component of the vector $\mathbf{w}$. The term $\partial E/\partial w^j$ produces the backpropagation through the chain rule, i.e., it produces the sequential set of functions to evaluate as in (6.21). Methods for solving this optimization more quickly, or even simply enabling the computation to be tractable, remain of active research interest. Perhaps the most important method is stochastic gradient descent, which is considered in the next section.

6.4 The Stochastic Gradient Descent Algorithm

Training neural networks is computationally expensive due to the size of the NNs being trained. Even NNs of modest size can become prohibitively expensive if the optimization routines used for training are not well informed. Two algorithms have been especially critical for enabling the training of NNs: *stochastic gradient descent* (SGD) and backprop. Backprop allows for an efficient computation of the objective function's gradient, while SGD provides a more rapid evaluation of the optimal network weights. Although alternative optimization methods for training NNs continue to provide computational improvements, backprop and SGD are both considered here in detail so as to give the reader an idea of the core architecture for building NNs.

Gradient descent was considered in Section 4.2. Recall that this algorithm was developed for nonlinear regression where the data fit takes the general form

$$f(x) = f(x, \boldsymbol{\theta}), \tag{6.24}$$

where $\boldsymbol{\theta}$ are fitting coefficients used to minimize the error. In NNs, the parameters $\boldsymbol{\theta}$ are the network weights; thus we can rewrite this in the form

$$f(\mathbf{x}) = f(\mathbf{x}, \mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_M), \tag{6.25}$$

where the $\mathbf{A}_j$ are the connectivity matrices from one layer to the next in the NN. Thus $\mathbf{A}_1$ connects the first and second layers, and there are M hidden layers.

The goal of training the NN is to minimize the error between the network and the data. The standard root-mean-square error for this case is defined as

$$\underset{\mathbf{A}_j}{\operatorname{argmin}} E(\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_M) = \underset{\mathbf{A}_j}{\operatorname{argmin}} \sum_{k=1}^{n} (f(\mathbf{x}_k, \mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_M) - \mathbf{y}_k)^2, \tag{6.26}$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

which can be minimized by setting the partial derivative with respect to each matrix component to zero, i.e., we require $\partial E/\partial(a_{ij})_k = 0$, where $(a_{ij})_k$ is the ith row and jth column of the kth matrix ($k = 1, 2, \ldots M$). Recall that the zero derivative is a minimum, since there is no maximum error. This gives the gradient $\nabla f(\mathbf{x})$ of the function with respect to the NN parameters. Note further that $f(\cdot)$ is the function evaluated at each of the n data points.

As was shown in Section 4.2, this leads to a Newton–Raphson iteration scheme for finding the minima,

$$\mathbf{x}_{j+1}(\delta) = \mathbf{x}_j - \delta \nabla f(\mathbf{x}_j), \tag{6.27}$$

where δ is a parameter determining how far a step should be taken along the gradient direction. In NNs, this parameter is called the *learning rate*. Unlike standard gradient descent, it can be computationally prohibitive to compute an optimal learning rate.

Although the optimization formulation is easily constructed, evaluating (6.26) is often computationally intractable for NNs. This is due to two reasons: (i) the number of matrix weighting parameters for each $\mathbf{A}_j$ is quite large, and (ii) the number of data points n is generally also large.

To render the computation (6.26) potentially tractable, SGD does not estimate the gradient in (6.27) using all n data points. Rather, a single, randomly chosen data point, or a subset for *batch gradient descent*, is used to approximate the gradient at each step of the iteration. In this case, we can reformulate the least-squares fitting of (6.26) so that

$$E(\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_M) = \sum_{k=1}^{n} E_k(\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_M) \tag{6.28}$$

and

$$E_k(\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_M) = (f_k(\mathbf{x}_k, \mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_M) - \mathbf{y}_k)^2, \tag{6.29}$$

where $f_k(\cdot)$ is now the fitting function for each data point, and the entries of the matrices $\mathbf{A}_j$ are determined from the optimization process.

The gradient descent iteration algorithm (6.27) is now updated as follows:

$$\mathbf{w}_{j+1}(\delta) = \mathbf{w}_j - \delta \nabla E_k(\mathbf{w}_j), \tag{6.30}$$

where $\mathbf{w}_j$ is the vector of all the network weights from $\mathbf{A}_j$ ($j = 1, 2, \ldots, M$) at the jth iteration, and the gradient is computed using only the kth data point and $f_k(\cdot)$. Thus, instead of computing the gradient with all n points, only a single data point is randomly selected and used. At the next iteration, another randomly selected point is used to compute the gradient and update the solution. The algorithm may require multiple passes through all the data to converge, but each step is now easy to evaluate versus the expensive computation of the

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

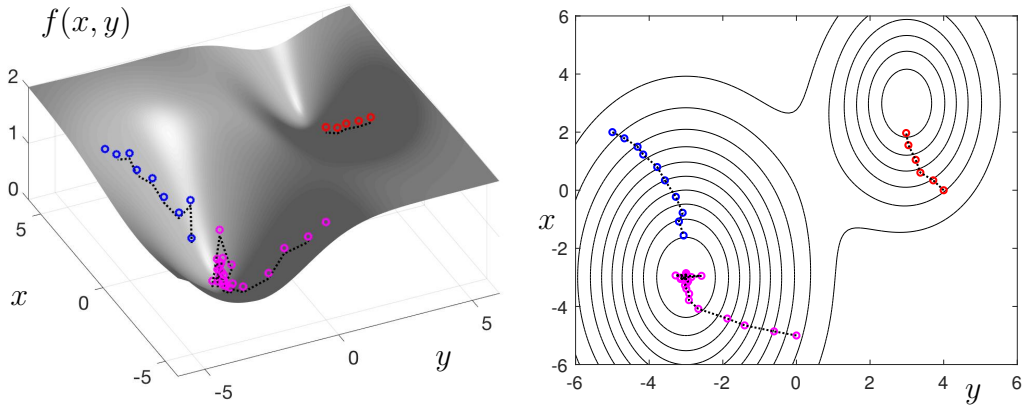


Figure 6.11: Stochastic gradient descent applied to the function featured in Fig. 4.3(b). The convergence can be compared to a full gradient descent algorithm as shown in Fig. 4.6. Each step of the stochastic (batch) gradient descent selects 100 data points for approximating the gradient, instead of the 10^4 data points of the data. Three initial conditions are shown: $(x_0, y_0) = \{(0, 4), (-5, 0), (2, -5)\}$. The first of these (red circles) gets stuck in a local minimum, while the other two initial conditions (blue and magenta) find the global minimum. Interpolation of the gradient functions of Fig. 4.5 is used to update the solutions.

Jacobian that is required for the gradient. If, instead of a single point, a subset of points is used, then we have the following batch gradient descent algorithm:

$$\mathbf{w}_{j+1}(\delta) = \mathbf{w}_j - \delta \nabla E_K(\mathbf{w}_j), \tag{6.31}$$

where $K \in [k_1, k_2, \ldots, k_p]$ denotes the p randomly selected data points k_j used to approximate the gradient.

Code from Section 4.2 can be modified for the stochastic gradient descent. The modification here involves taking a significant subsampling of the data to approximate the gradient. Specifically, a batch gradient descent is illustrated with a fixed learning rate of $\delta = 2$. Ten points are used to approximate the gradient of the function at each step.

Figure 6.11 shows the convergence of SGD for three initial conditions. As with gradient descent, the algorithm can get stuck in local minima. However, the SGD now approximates the gradient with only 100 points instead of the full 10^4 points, thus allowing for a computation that is three orders of magnitude smaller. Importantly, the SGD is a scalable algorithm, allowing for significant computational savings even as the data grows to be high-dimensional. For this reason, SGD has become a critically enabling part of NN training. Note that the learning rate, batch size, and data sampling play an important role in the convergence of the method.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

6.5 Deep Convolutional Neural Networks

With the basics of the NN architecture in hand, along with an understanding of how to formulate an optimization framework powered by SGD and backprop, we are ready to construct *deep convolution neural nets* (DCNNs), which are the fundamental building blocks of *deep learning* methods. Indeed, today when practitioners generally talk about NNs for practical use, they are typically talking about DCNNs. Of course, *natural language processing* (NLP) is another important class powered by recurrent neural networks (RNNs). But as much as we would like to have a principled approach to building DCNNs, there remains a great deal of artistry and expert intuition for producing the highest-performing networks. Moreover, DCNNs are especially prone to overtraining, thus requiring special care to cross-validate the results. The recent textbook on deep learning by Goodfellow et al. [290] provides a detailed and extensive account of the state of the art in DCNNs. It is especially useful for highlighting many rules of thumb and tricks for training effective DCNNs.

Like SVM and random forest algorithms, the MATLAB package for building NNs has a tremendous number of features and tuning parameters. This flexibility is both advantageous and overwhelming at the same time. As was pointed out at the beginning of this chapter, it is immediately evident that there are a great number of design questions regarding NNs. How many layers should be used? What should be the dimension of the layers? How should the output layer be designed? Should one use all-to-all or sparsified connections between layers? How should the mapping between layers be performed: a *linear mapping* or a *nonlinear mapping*?

The prototypical structure of a DCNN is illustrated in Fig. 6.12. Included in the visualization is a number of commonly used convolutional and pooling layers. Also illustrated is the fact that each layer can be used to build multiple downstream layers, or *feature* spaces, which can be engineered by the choice of activation functions and/or network parameterizations. All of these layers are ultimately combined into the output layer. The number of connections that require updating through backprop and SGD can be extraordinarily high; thus even modest networks and training data may require significant computational resources. A typical DCNN is constructed of a number of layers, with DCNNs typically having 7–10 layers. More recent efforts have considered the advantages of a truly deep network with approximately 100 layers, but the merits of such architectures are still not fully known. The following paragraphs highlight some of the more prominent elements that comprise DCNNs, including convolutional layers, pooling layers, fully connected layers, and dropout.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

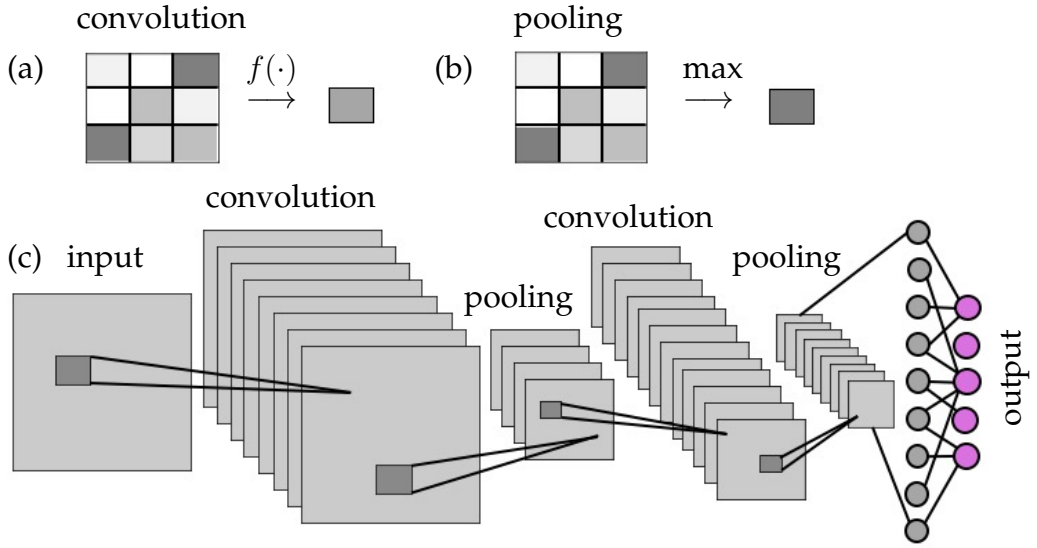


Figure 6.12: Prototypical DCNN architecture which includes commonly used convolutional and pooling layers. The dark gray boxes show the convolutional sampling from layer to layer. Note that, for each layer, many functional transformations can be used to produce a variety of feature spaces. The network ultimately integrates all this information into the output layer.

Convolutional Layers

Convolutional layers are similar to windowed (Gabor) Fourier transforms or wavelets from Chapter 2, in that a small selection of the full high-dimensional input space is extracted and used for feature engineering. Figure 6.12 shows the convolutional windows (dark gray boxes) that are slid across the entire layer (light gray boxes). Each convolutional window transforms the data into a new node through a given activation function, as shown in Fig. 6.12(a). The feature spaces are thus built from the smaller patches of the data. Convolutional layers are especially useful for images, as they can extract important features such as edges. Wavelets are also known to efficiently extract such features, and there are deep mathematical connections between wavelets and DCNNs, as shown by Mallat and co-workers [18, 476]. Note that in Fig. 6.12 the input layer can be used to construct many layers by simply manipulating the activation function $f(\cdot)$ to the next layer as well the size of the convolutional window.

Pooling Layers

It is common to periodically insert a pooling layer between successive convolutional layers in a DCNN architecture. Its function is to progressively reduce the spatial size of the representation in order to reduce the number of parameters and computation in the network. This is an effective strategy (i) to help control overfitting and (ii) to fit the computation in memory. Pooling layers op-

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

erate independently on every depth slice of the input and resize them spatially. Using the max operation, i.e., the maximum value for all the nodes in its convolutional window, is called *max pooling*. In image processing, the most common form of max pooling is a pooling layer with filters of size 2×2 applied with a stride of two downsamples every depth slice in the input by two along both width and height, discarding 75% of the activations. Every max pooling operation would in this case be taking a max over four numbers (a 2×2 region in some depth slice). The depth dimension remains unchanged. An example max pooling operation is shown in Fig. 6.12(b), where a 3×3 convolutional cell is transformed to a single number that is the maximum of the nine numbers.

Fully Connected Layers

Occasionally, fully connected layers are inserted into the DCNN so that different regions can be connected. The pooling and convolutional layers are *local* connections only, while the fully connected layer restores *global* connectivity. This is another commonly used layer in the DCNN architecture, providing a potentially important feature space to improve performance.

Dropout

Overfitting is a serious problem in DCNNs. Indeed, overfitting is at the core of why DCNNs often fail to demonstrate good generalizability properties (see Chapter 4 on regression). Large DCNNs are also slow to use, making it difficult to deal with overfitting by combining the predictions of many different large neural nets for online implementation. Dropout is a technique which helps address this problem. The key idea is to randomly drop nodes in the network (along with their connections) from the DCNN during training, i.e., during SGD/backprop updates of the network weights. This prevents units from co-adapting too much. During training, dropout samples form an exponential number of different "thinned" networks. This idea is similar to the ensemble methods for building random forests. At test time, it is easy to approximate the effect of averaging the predictions of all these thinned networks by simply using a single unthinned network that has smaller weights. This significantly reduces overfitting and has been shown to give major improvements over other regularization methods [672].

There are many other techniques that have been devised for training DCNNs, but the above methods highlight some of the most commonly used. The most successful applications of these techniques tend to be in computer vision tasks where DCNNs offer unparalleled performance in comparison to other machine learning methods. Importantly, the ImageNet data set is what allowed these

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

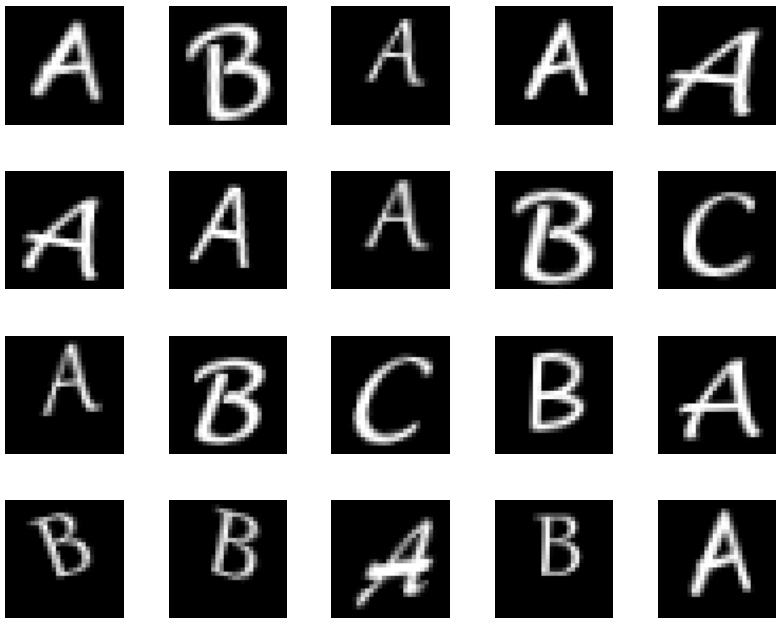


Figure 6.13: Representative images of the alphabet characters A, B, and C. There are a total of 1500 28×28 grayscale images (XTrain) of the letters that are labeled (TTrain).

DCNN layers to be maximally leveraged for human-level recognition performance.

To illustrate how to train and execute a DCNN, we use data from MATLAB. Specifically, we use a data set that has a training and test set with the alphabet characters A, B, and C (see Fig. 6.13). The training data, XTrain, contains 1500 28×28 grayscale images of the letters A, B, and C in a four-dimensional array. There are equal numbers of each letter in the data set. The variable TTrain contains the categorical array of the letter labels, i.e., the truth labels. The following code constructs and trains a DCNN.

Code 6.3: [MATLAB] Train a DCNN.

```
layers = [imageInputLayer([28 28 1]);
          convolution2dLayer(5,16);
          reluLayer();
          maxPooling2dLayer(2,'Stride',2);
          fullyConnectedLayer(3);
          softmaxLayer();
          classificationLayer()];
options = trainingOptions('sgdm');
rng('default') % For reproducibility
net = trainNetwork(XTrain,TTrain,layers,options);
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Code 6.3: [Python] Train a DCNN.

```
model = Sequential()
model.add(Conv2D(filters=16, kernel_size=5, activation='relu
    ', input_shape=(28,28,1)))
model.add(MaxPool2D(pool_size=2, strides=2))
model.add(Flatten())
model.add(Dense(len(classes), activation='softmax'))

sgd_optimizer = optimizers.SGD(momentum=0.9)
model.compile(optimizer=sgd_optimizer, loss='
    categorical_crossentropy')
model.fit(XTrain, y_train, epochs=30)
```

Note the simplicity in how diverse network layers are easily put together. In addition, a ReLU activation layer is specified along with the training method of stochastic gradient descent (**sgdm**). The **trainNetwork** command integrates the options and layer specifications to build the best classifier possible. The resulting trained network can now be used on a test data set.

Code 6.4: [MATLAB] Test the DCNN performance.

```
YTest = classify(net,XTest);
```

Code 6.4: [Python] Test the DCNN performance.

```
YPredict = np.argmax(model.predict(XTest),axis=1)
```

The resulting classification performance is approximately 93%. One can see by this code structure that modifying the network architecture and specifications is trivial. Indeed, one can probably easily engineer a network to outperform the illustrated DCNN. As already mentioned, artistry and expert intuition are critical for producing the highest-performing networks.

6.6 Neural Networks for Dynamical Systems

Neural networks offer an amazingly flexible architecture for performing a diverse set of mathematical tasks. To return to Mallat et al.: *Supervised learning is a high-dimensional interpolation problem* [476]. Thus, if sufficiently rich data can be acquired, NNs offer the ability to interrogate the data for a variety of tasks centered on classification and prediction. To this point, the tasks demonstrated have primarily been concerned with computer vision. However, NNs can also be used for future state predictions of dynamical systems (see Chapter 7).

To demonstrate the usefulness of NNs for applications in dynamical sys-

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

tems, we will consider the Lorenz system of differential equations [460]

$$\dot{x} = \sigma(y - x), \tag{6.32a}$$
$$\dot{y} = x(\rho - z) - y, \tag{6.32b}$$
$$\dot{z} = xy - \beta z, \tag{6.32c}$$

where the state of the system is given by $\mathbf{x} = [x\ y\ z]^T$ with the parameters $\sigma = 10$, $\rho = 28$, and $\beta = 8/3$. This system will be considered in further detail in the next chapter. For the present, we will simulate this nonlinear system and use it as a demonstration of how NNs can be trained to characterize dynamical systems. Specifically, the goal of this section is to demonstrate that we can train a NN to learn an update rule which advances the state space from $\mathbf{x}_k$ to $\mathbf{x}_{k+1}$, where k denotes the state of the system at time t_k. Accurately advancing the solution in time requires a nonlinear transfer function, since the Lorenz system itself is nonlinear.

The training data required for the NN is constructed from high-accuracy simulations of the Lorenz system. The following code generates a diverse set of initial conditions. One hundred initial conditions are considered in order to generate 100 trajectories. The sampling time is fixed at $\Delta t = 0.01$. Note that the sampling time is not the same as the time-steps taken by the fourth-order Runge–Kutta method [420]. The time-steps are adaptively chosen to meet the stringent tolerances of accuracy chosen for this example.

Code 6.5: [MATLAB] Create training data of Lorenz trajectories.

```
dt=0.01; T=8; t=0:dt:T;
b=8/3; sig=10; r=28;

Lorenz = @(t,x)([ sig * (x(2) - x(1))        ; ...
                  r * x(1)-x(1) * x(3) - x(2) ; ...
                  x(1) * x(2) - b*x(3)         ]);
ode_options = odeset('RelTol',1e-10, 'AbsTol',1e-11);

input=[]; output=[];
for j=1:100  % training trajectories
    x0=30*(rand(3,1)-0.5);
    [t,y] = ode45(Lorenz,t,x0);
    input=[input; y(1:end-1,:)];
    output=[output; y(2:end,:)];
end
```

Code 6.5: [Python] Create training data of Lorenz trajectories.

```
dt = 0.01; T = 8; t = np.arange(0,T+dt,dt)
beta = 8/3; sigma = 10; rho = 28
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

```
nn_input = np.zeros((100*(len(t)-1),3))
nn_output = np.zeros_like(nn_input)
def lorenz_deriv(x_y_z,t0,sigma=sigma,beta=beta,rho=rho):
    x, y, z = x_y_z
    return [sigma*(y-x), x*(rho-z)-y, x*y-beta*z]

x0 = -15 + 30 * np.random.random((100, 3))
x_t = np.asarray([integrate.odeint(lorenz_deriv, x0_j, t)
                  for x0_j in x0])

for j in range(100):
    nn_input[j*(len(t)-1):(j+1)*(len(t)-1),:] = x_t[j,:-1,:]
    nn_output[j*(len(t)-1):(j+1)*(len(t)-1),:] = x_t[j,1:,:]
```

The simulation of the Lorenz system produces two key matrices: **input** and **output**. The former is a matrix of the system at $\mathbf{x}_k$, while the latter is the corresponding state of the system $\mathbf{x}_{k+1}$ advanced $\Delta t = 0.01$.

The NN must learn the nonlinear mapping from $\mathbf{x}_k$ to $\mathbf{x}_{k+1}$. Figure 6.14 shows the various trajectories used to train the NN. Note the diversity of initial conditions and the underlying attractor of the Lorenz system.

We now build a NN trained on trajectories of Fig. 6.14 to advance the solution $\Delta t = 0.01$ into the future for an arbitrary initial condition. Here, a three-layer network is constructed with 10 nodes in each layer and a different activation unit for each layer. The choice of activation types, nodes in the layer, and number of layers are arbitrary. It is trivial to make the network deeper and wider and enforce different activation units. The performance of the NN for the arbitrary choices made is quite remarkable and does not require additional tuning. The NN is built with the following few lines of code.

Code 6.6: [MATLAB] Build a neural network for Lorenz system.

```
net = feedforwardnet([10 10 10]);
net.layers{1}.transferFcn = 'logsig';
net.layers{2}.transferFcn = 'radbas';
net.layers{3}.transferFcn = 'purelin';
net = train(net,input.',output.');
```

Code 6.6: [Python] Build a neural network for Lorenz system.

```
net = keras.models.Sequential()
net.add(layers.Dense(10, input_dim=3, activation='sigmoid'))
net.add(layers.Dense(10, activation='relu'))
net.add(layers.Dense(3, activation='linear'))
net.compile(loss='mse', optimizer='adam')
History = net.fit(nn_input, nn_output, epochs=1000)
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

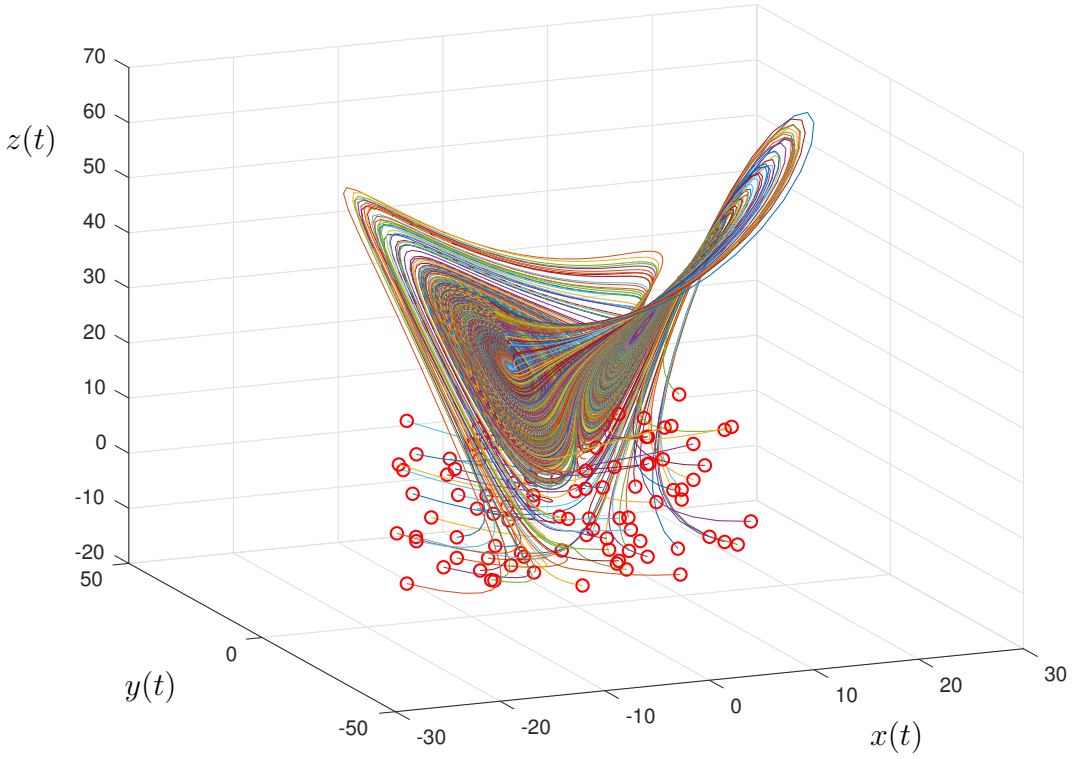


Figure 6.14: Evolution of the Lorenz dynamical equations for 100 randomly chosen initial conditions (red circles). For the parameters $\sigma = 10$, $\rho = 28$, and $\beta = 8/3$, all trajectories collapse to an attractor. These trajectories, generated from a diverse set of initial data, are used to train a neural network to learn the nonlinear mapping from $\mathbf{x}_k$ to $\mathbf{x}_{k+1}$.

The code produces a function **net** which can be used with a new set of data to produce predictions of the future. Specifically, the function **net** gives the nonlinear mapping from $\mathbf{x}_k$ to $\mathbf{x}_{k+1}$. Figure 6.15 shows the structure of the network along with the performance of the training over 1000 epochs of training. The results of the cross-validation are also demonstrated. The NN converges steadily to a network that produces accuracies on the order of 10^{-5}.

Once the NN is trained on the trajectory data, the nonlinear model mapping $\mathbf{x}_k$ to $\mathbf{x}_{k+1}$ can be used to predict the future state of the system from an initial condition. In the following code, the trained function **net** is used to take an initial condition and advance the solution Δt. The output can be reinserted into the **net** function to estimate the solution $2\Delta t$ into the future. This iterative mapping can produce a prediction for the future state as far into the future as desired. In what follows, the mapping is used to predict the Lorenz solutions eight time units into the future from a given initial condition. This can then be compared against the ground-truth simulation of the evolution using a fourth-order Runge–Kutta method. The following iteration scheme gives the

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

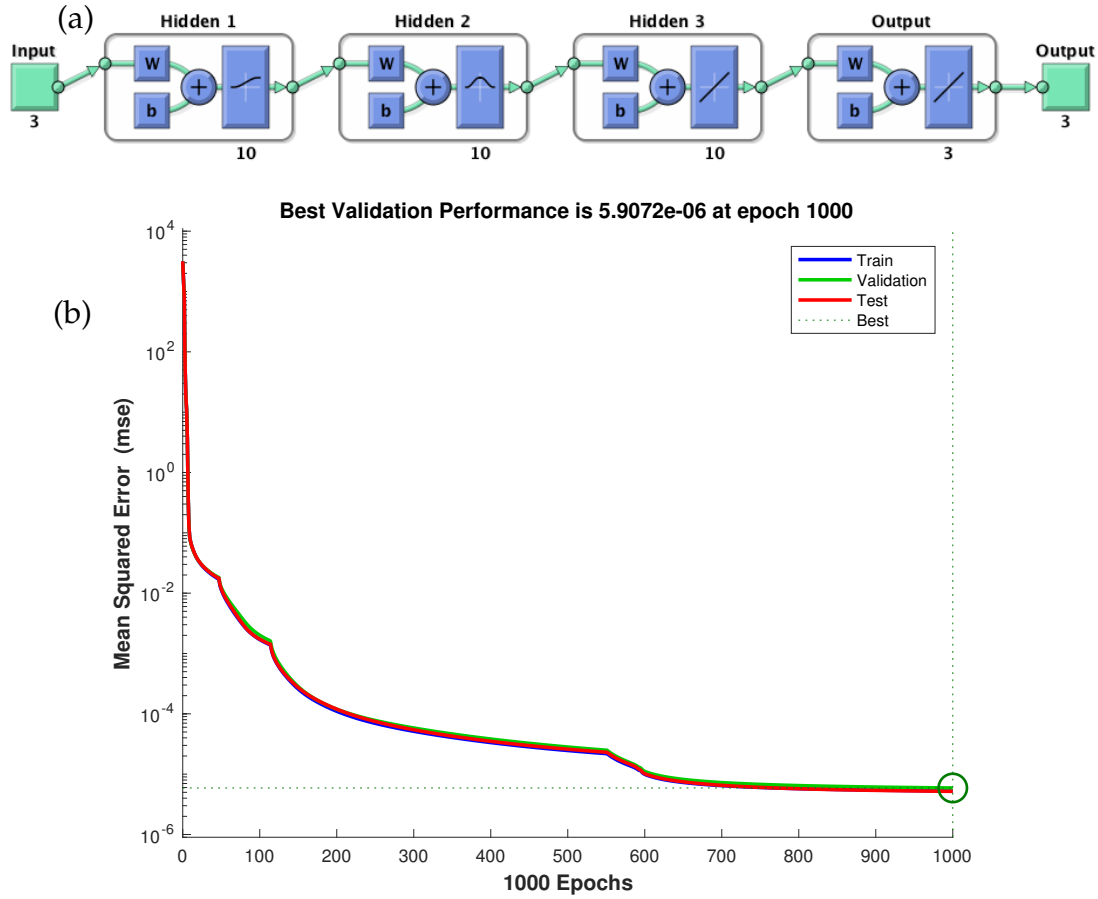


Figure 6.15: (a) Network architecture used to train the NN on the trajectory data of Fig. 6.14. A three-layer network is constructed with 10 nodes in each layer and a different activation unit for each layer. (b) Performance summary of the NN optimization algorithm. Over 1000 epochs of training, accuracies on the order of 10^{-5} are produced. The NN is also cross-validated in the process.

NN approximation to the dynamics.

Code 6.7: [MATLAB] Neural network for prediction.

```
ynn(1,:)=x0;
for jj=2:length(t)
    y0=net(x0);
    ynn(jj,:)=y0.'; x0=y0;
end
```

Code 6.7: [Python] Neural network for prediction.

```
ynn = np.zeros((num_traj, len(t), 3))
ynn[:, 0, :] = -15 + 30 * np.random.random((num_traj, 3))
for jj, tval in enumerate(t[:-1]):
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

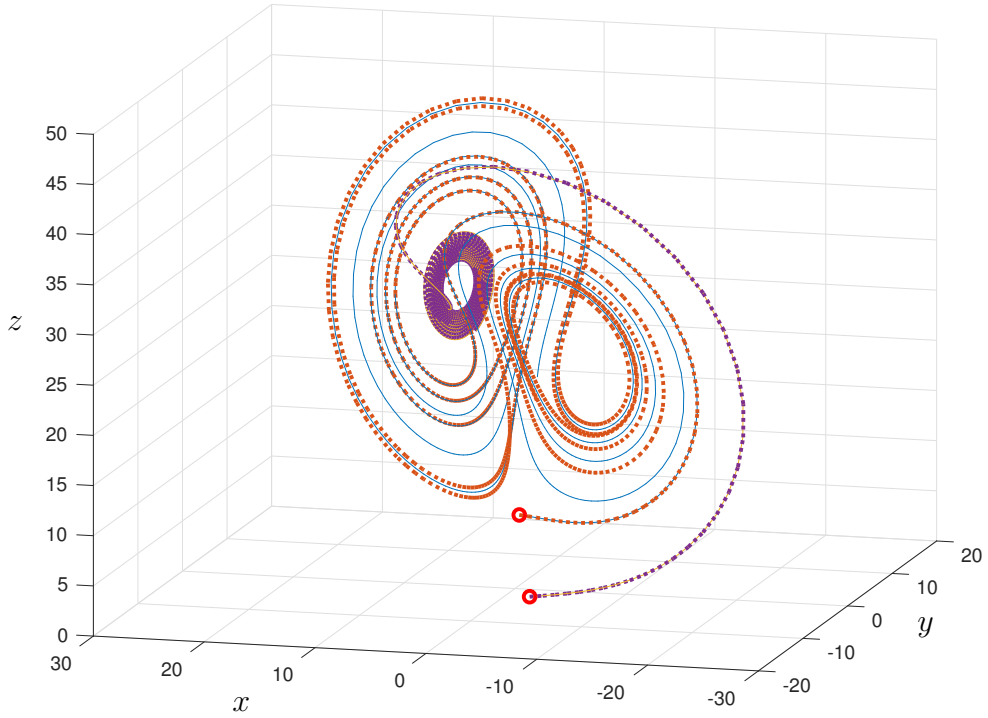


Figure 6.16: Comparison of the time evolution of the Lorenz system (solid line) with the NN prediction (dotted line) for two randomly chosen initial conditions (red dots). The NN prediction stays close to the dynamical trajectory of the Lorenz model. A more detailed comparison is given in Fig. 6.17.

```
ynn[:, jj+1, :] = net.predict(ynn[:, jj, :])
```

Figure 6.16 shows the evolution of two randomly drawn trajectories (solid lines) compared against the NN prediction of the trajectories (dotted lines). The NN prediction is remarkably accurate in producing an approximation to the high-accuracy simulations. This shows that the data used for training is capable of producing a high-quality nonlinear model mapping $\mathbf{x}_k$ to $\mathbf{x}_{k+1}$. The quality of the approximation is more clearly seen in Fig. 6.17 where the time evolution of the individual components of $\mathbf{x}$ are shown against the NN predictions. See Section 7.5 for further details.

In conclusion, the NN can be trained to learn dynamics. More precisely, the NN seems to learn an algorithm which is approximately equivalent to a fourth-order Runge–Kutta scheme for advancing the solution a time-step Δt. Indeed, NNs have been used to model dynamical systems [289] and other physical processes [502] for decades. However, great strides have been made recently in using DNNs to learn Koopman embeddings, resulting in several excellent papers [440, 485, 540, 692, 747, 766]. For example, the VAMPnet architecture [485, 747] uses a time-lagged autoencoder and a custom variational score to identify Koopman coordinates on an impressive protein folding example. In an

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

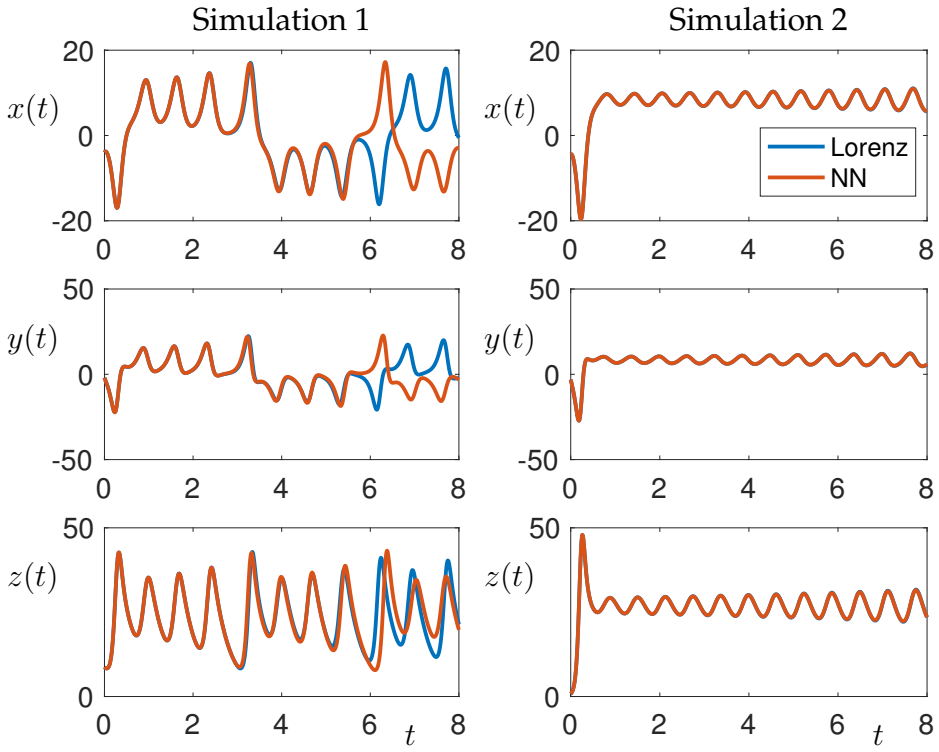


Figure 6.17: Comparison of the time evolution of the Lorenz system for two randomly chosen initial conditions (also shown in Fig. 6.16). The left column shows that the evolution of the Lorenz differential equations and the NN mapping give identical results until $t \approx 5.5$, at which point they diverge. In contrast, the NN prediction stays on the trajectory of the second initial condition for the entire time window.

alternative formulation, variational autoencoders can build low-rank models that are efficient and compact representations of the Koopman operator from data [465]. By construction, the resulting network is both parsimonious and interpretable, retaining the flexibility of neural networks and the physical interpretation of Koopman theory. In all of these recent studies, DNN representations have been shown to be more flexible and exhibit higher accuracy than other leading methods on challenging problems.

6.7 Recurrent Neural Networks

Recurrent neural networks (RNNs) are an important class of neural network architectures that leverage sequential data streams. Sequential data is prevalent in speech recognition, as sentences and phrases have specific temporal structures in order to produce output that is meaningful. RNNs are trained by respecting the time history of a given sequence. Thus, unlike the standard feed-

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

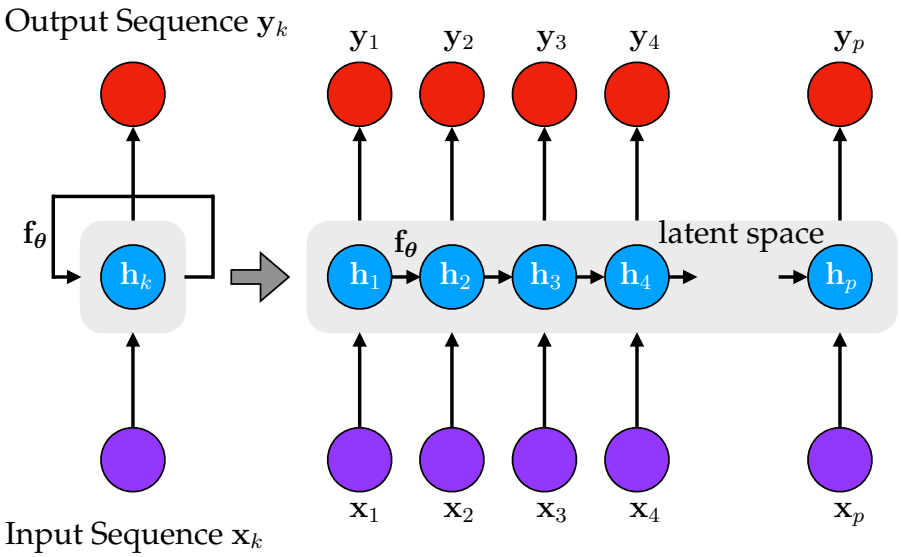


Figure 6.18: Recurrent neural network structure which trains on sequences of input data $\mathbf{x}_k = \mathbf{x}(t_k)$ and output data $\mathbf{y}_k = \mathbf{y}(t_k)$. Unlike a feedforward neural network, the time history of the sequence, or memory, is used to train the neural network. Thus the output of the neural network is fed back into the latent layer $\mathbf{f}_{\boldsymbol{\theta}}$. The left representation of the RNN shows the recurrent structure that is achieved from feeding the output back into the neural network. The right representation is the *unfolding* of the graph, which shows a neural network $\mathbf{f}_{\boldsymbol{\theta}}$ that shares weights across different time points.

forward neural networks of the last section, the time history of the sequences, or memory, is explicitly accounted for. Figure 6.18 shows the neural network architecture of a generic RNN, where a sequence with m snapshots of temporal history is trained to learn a representation of the sequence.

RNNs are structured around key ideas in dynamical systems. Specifically, we often think of dynamics in terms of a flow map

$$\mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \boldsymbol{\theta}), \tag{6.33}$$

which advances a solution forward in time from $\mathbf{x}_k = \mathbf{x}(t_k)$ to $\mathbf{x}_{k+1} = \mathbf{x}(t_{k+1})$. In the last section, we constructed a feedforward neural network to model the flow map

$$\mathbf{x}_{k+1} = \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_k), \tag{6.34}$$

where $\boldsymbol{\theta}$ are the network weights. An RNN trains over a sequence of temporal

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

snapshots, which have an associated output $\mathbf{y}_k$, so that

$$\mathbf{x}_{k+m} = \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{f}_{\boldsymbol{\theta}}(\cdots \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_k)\cdots)). \tag{6.35}$$

This expression is a flow map over m steps from $\mathbf{x}_k = \mathbf{x}(t_k)$ to $\mathbf{x}_{k+m} = \mathbf{x}(t_{k+m})$. This is the *unfolding* of the recursive graph in Fig. 6.18.

It is also often the case that, instead of mapping a sequence in the original input variable $\mathbf{x}_k$, the latent space is used for building a map of the recurrence and to the output sequence $\mathbf{y}_k$. If $\mathbf{h}_k = \mathbf{h}(t_k)$ is the latent representation, then the model becomes

$$\mathbf{h}_{k+1} = \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{h}_k, \mathbf{x}_{k+1}), \tag{6.36}$$

where the neural network is now dependent on the input variable. In either case, a neural network $\mathbf{f}_{\boldsymbol{\theta}}$ is trained to advance the solution in time by training on trajectories from time t_1 to t_m. Thus the big difference between the feedforward neural networks of the last chapter and RNNs is that: RNNs train on trajectories from t_1 to t_m using the entire sequence of time points, whereas feedforward NNs train from t_k to t_{k+1} (or t_k to t_{k+m} as in Section 12.6). Training over trajectories allows for the history (memory) of the solution to shape the neural network model.

The history of RNNs begins in the 1980s with the foundational work of Rumelhart et al. [614] and Hopfield [337]. RNNs in the form of *long short-term memory* (LSTM) networks [331] became especially transformative in speech recognition applications since an LSTM, through its filtering architecture, regularize RNNs to avoid the vanishing gradient problem that is typically encountered in training. Other RNN architectures that have been constructed in order to avoid the vanishing or exploding gradients problem include *gated recurrent units* (GRU) and *echo state networks* (ESN). Thus LSTM, GRU, and ESN, along with their variants, are commonly used with time-series data.

Training on the Lorenz dynamical system model of the last section is similar to building the feedforward network already considered. In this case, an LSTM model is built from portions of the trajectory data. The trained model is compared against the evolution dynamics in Fig. 6.19. The LSTM generates a behavior that mimics the dynamics of Lorenz equations using trajectories of 40 data points. A simple RNN can be constructed instead of an LSTM by uncommenting from the code below.

Code 6.8: [Python] LSTM model for dynamics.

```
sequence_size = 40; train_size = 80; test_size = 20

rnn_input = np.zeros((train_size*(len(t)-sequence_size-1),
    sequence_size, 3))
rnn_output = np.zeros((train_size*(len(t)-sequence_size-1),
    3))
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

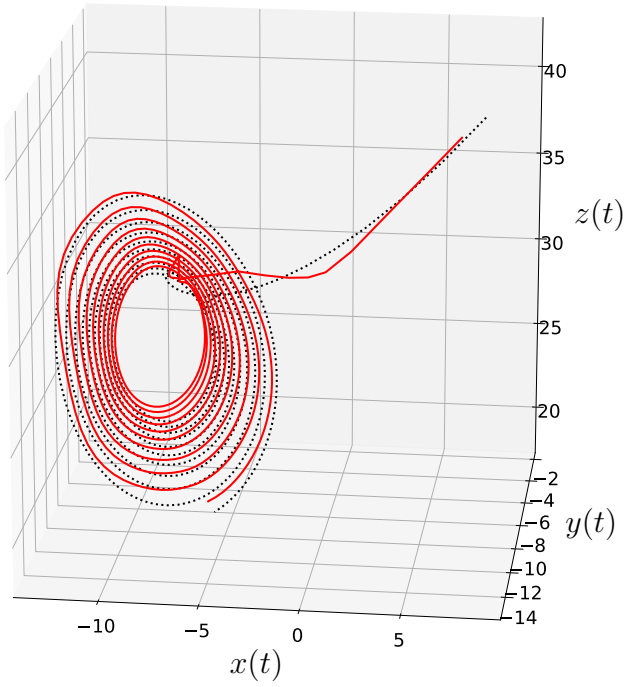


Figure 6.19: Trajectories of the Lorenz dynamical system (black dotted line) versus the trajectories learned by an LSTM (solid red line). The fit can be improved by hyperparameter tuning of the sequence size and training time. The specific training trajectory length selected here was a sequence of 40 time points.

```
for j in range(train_size):
  for k in range(len(t)-sequence_size-1):
    rnn_input[j*(len(t)-sequence_size-1) + k,:] = x_t[j,k:k+
       sequence_size,:]
    rnn_output[j*(len(t)-sequence_size-1) + k,:] = x_t[j,k+
       sequence_size,:]

model = Sequential()
model.add(LSTM(16, input_shape=(None, 3)))
# model.add(SimpleRNN(16, input_shape=(None, 3)))
model.add(Dense(3))

sgd = SGD(0.01)
model.compile(optimizer=sgd, loss='mean_squared_error')
model.fit(rnn_input, rnn_output, epochs=20)
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

6.8 Autoencoders

Autoencoder neural networks are a flexible and advantageous structure for exploiting low-dimensional features in high-dimensional data. They are featured here since many scientific and engineering applications leverage low-dimensional coordinate systems for building parsimonious models characterizing a physical process. The autoencoder generalizes the linear subspace embedding of SVD/PCA to a nonlinear manifold embedding, often of a lower dimension. Specifically, the autoencoder maps the original high-dimensional input vectors $\mathbf{x}_j \in \mathbb{R}^n$ to a low-dimensional latent variable $\mathbf{z}_j \in \mathbb{R}^r$ and then back to the high-dimensional space $\tilde{\mathbf{x}}$, which is technically the output $\mathbf{y}$. The goal of the autoencoder is to map the output back to itself, i.e., $\|\tilde{\mathbf{x}} - \mathbf{x}\|_2 \approx 0$. Typically $r \ll n$ for autoencoding and mathematically

$$\mathbf{Z} = \phi(\mathbf{X}), \tag{6.37}$$

where $\mathbf{Z}$ is the latent space data and $\mathbf{X}$ is the input high-dimensional data. Note that the columns of $\mathbf{Z}$ are $\mathbf{z}_j$ and the columns of $\mathbf{X}$ are $\mathbf{x}_j$. Decoding is represented as

$$\tilde{\mathbf{X}} = \psi(\mathbf{Z}), \tag{6.38}$$

where the neural network weights are optimized so that the output $\tilde{\mathbf{X}}$ is as close as possible to the input,

$$\underset{\boldsymbol{\theta}}{\operatorname{argmin}} \|\mathbf{X} - \tilde{\mathbf{X}}\|_2^2 = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \|\mathbf{X} - \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{X})\|_2^2, \tag{6.39}$$

where $\boldsymbol{\theta}$ are the weights of the autoencoder network $\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}) = \psi(\phi(\mathbf{x}))$. The dimension of the latent space r is often determined by hyperparameter tuning. Thus r is made as small as possible until the autoencoder performance starts to fail. This is often informative, as it can discover the intrinsic dimensionality of the data.

From a more mathematically abstract point of view, the autoencoder provides a mapping, as illustrated in Fig. 6.20, so that

$$\phi : \mathcal{X} \to \mathcal{Z}, \tag{6.40a}$$
$$\psi : \mathcal{Z} \to \mathcal{X}, \tag{6.40b}$$

where the input $\mathbf{x} \in \mathcal{X} \subseteq \mathbb{R}^n$ and output $\mathbf{z} \in \mathcal{Z} \subseteq \mathbb{R}^r$ are defined in high- and low-dimensional spaces, respectively. The resulting neural network optimization is formulated around the loss function

$$\underset{\phi,\psi}{\operatorname{argmin}} \|\mathbf{X} - (\psi \circ \phi)\mathbf{X}\|. \tag{6.41}$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

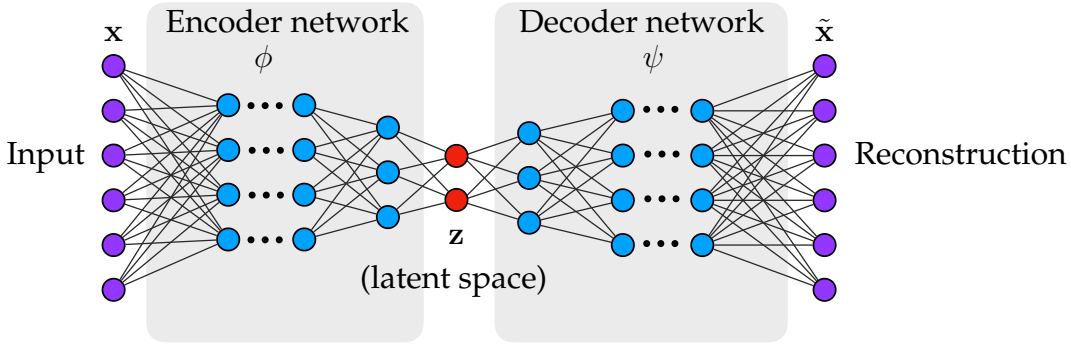


Figure 6.20: Autoencoder network structure which maps the input state space $\mathbf{x}$ to a latent space $\mathbf{z}$. For applications considered here, the latent space if where the dynamical evolution is modeled. The encoder is denoted by ϕ and the decoder by ψ. The autoencoder is trained by minimizing the loss $\|\mathbf{x} - \tilde{\mathbf{x}}\|_2^2$ along with any other regularization that may be applied.

More generally, the autoencoder is often used with a diversity of regularizations in the optimization process, so that a more general formulation is given by

$$\underset{\phi,\psi}{\operatorname{argmin}} \|\mathbf{X} - (\psi \circ \phi)\mathbf{X}\| + \sum_{j=1}^{P} \lambda_j g_j(\mathbf{X}, \phi, \psi), \tag{6.42}$$

where $g_j(\cdot)$ represents a regularizer, weighted by λ_j, and there are total of P additional loss functions added to the optimization. For instance, an elastic net penalty can be added where two additional loss functions would penalize the ℓ_2- and ℓ_1-norm of the network weights. This often can help produce better results in the network. The ℓ_1-norm, for instance, can help deal with outliers and corrupt data.

To demonstrate the ability of the autoencoder to construct a low-dimensional representation of high-dimensional data, the fluid flow around a cylinder is considered. The data is generated from snapshots of the numerical simulation of the incompressible Navier–Stokes equation:

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} + \nabla p - \frac{1}{Re} \nabla^2 \mathbf{u} = 0, \tag{6.43}$$

with the incompressibility constraints $\nabla \cdot \mathbf{u} = 0$. Here $\mathbf{u}(x, y, t)$ represents the 2D velocity, and $p(x, y, t)$ the corresponding pressure field. The boundary conditions dictate a constant flow of $\mathbf{u} = (1, 0)^T$ at $x = -15$, i.e., the entry of the domain. There is also a constant pressure of $p = 0$ at $x = 25$, i.e., the end of the domain, and Neumann boundary conditions, i.e., $\partial \mathbf{u}/\partial \mathbf{n} = 0$, on the boundary of the domain and the cylinder (centered at $(x, y) = (0, 0)$ and of radius unity).

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

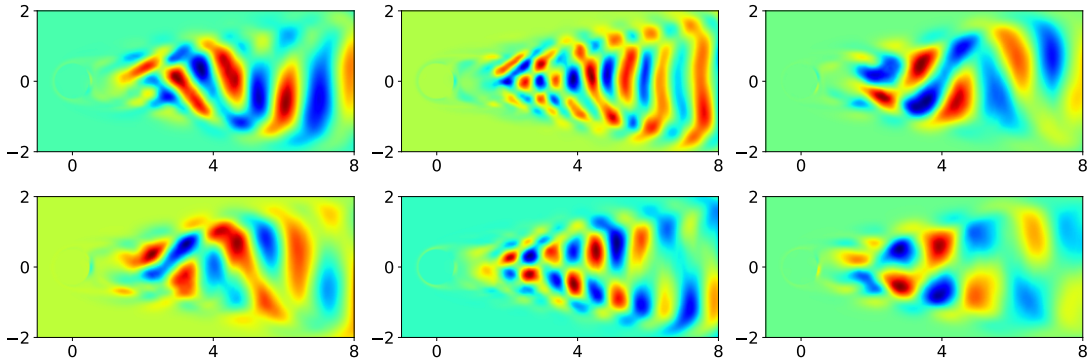


Figure 6.21: First six most dominant modes (top left to bottom right) learned by the autoencoder for the flow around a cylinder. These modes are the latent representation of the flow physics.

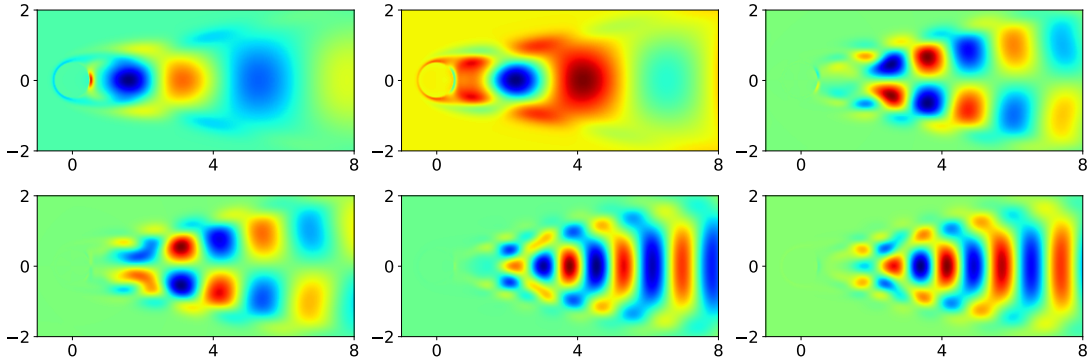


Figure 6.22: First six most dominant modes (top left to bottom right) learned by the autoencoder for the flow around a cylinder using a linear reduction, which is equivalent to $\phi \rightarrow \mathbf{U}^*$ and $\psi \rightarrow \mathbf{U}$. These modes are the SVD (PCA or POD) representation of the flow physics.

The autoencoder is created using the *keras* package with *tensorflow*. The following code creates an autoencoder/decoder structure, whereby the input layer is recursively made half the size of the layers before it. Three layers are constructed on the way to the latent dimension of $r = 10$.

Code 6.9: [Python] Autoencoder structure.

```
class Autoencoder(tf.keras.Model):
    def __init__(self, latent_dim, input_dim, activation='
        sigmoid'):
         super(Autoencoder, self).__init__()
         self.latent_dim = latent_dim
         self.input_dim = input_dim
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

```
        self.activation = activation

        self.encoder = tf.keras.Sequential([
            tf.keras.layers.Dense(int(self.input_dim/2),
                activation=self.activation),
            tf.keras.layers.Dense(int(self.input_dim/4),
                activation=self.activation),
            tf.keras.layers.Dense(int(self.input_dim/8),
                activation=self.activation),
            tf.keras.layers.Dense(self.latent_dim,
                activation='linear'),
        ])
        self.decoder = tf.keras.Sequential([
            tf.keras.layers.Dense(int(self.input_dim/8),
                activation=self.activation),
            tf.keras.layers.Dense(int(self.input_dim/4),
                activation=self.activation),
            tf.keras.layers.Dense(int(self.input_dim/2),
                activation=self.activation),
            tf.keras.layers.Dense(self.input_dim, activation
                ='linear'),
        ])

    def call(self, x):
        encoded = self.encoder(x)
        decoded = self.decoder(encoded)
        return decoded
```

To train the autoencoder model, the following code is used:

Code 6.10: [Python] Autoencoder training.

```
latent_dim = 10 # number of modes
activation = 'elu'
input_dim = x_train.shape[1]

optimizer = 'adam'; epochs = 50
A = Autoencoder(latent_dim, input_dim, activation)
A.compile(optimizer=optimizer, loss=tf.keras.losses.
   MeanSquaredError())
```

The encoding generates a low-dimensional representation of the flow field in the latent space. The first six modes of this nonlinear encoder are highlighted in Fig. 6.21. This should be compared to a linear encoding/decoding, which is achieved using PCA so that the encoder/decoder pair (ϕ, ψ) are given by the first r modes of the SVD $(\mathbf{U}^*, \mathbf{U})$. The linear modes are shown in Fig. 6.22. Note that the linear modes alternate between symmetric and antisymmetric

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

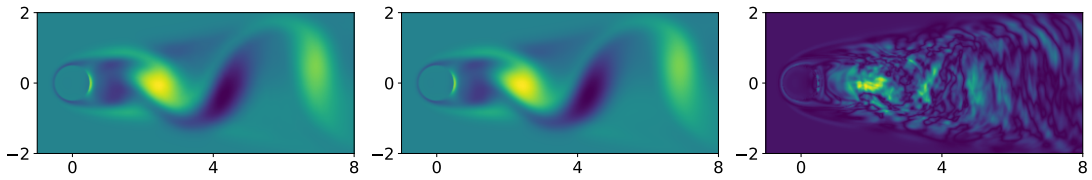


Figure 6.23: Error of the autoencoder representation. The left panel is the original state space $\mathbf{u}$ to be encoded. The middle panel is the reconstructed state space $\tilde{\mathbf{u}}$. And finally the right panel is the error between the original state space and its reconstruction $\mathbf{u} - \tilde{\mathbf{u}}$.

structure whereas the nonlinear encoder produces modes that don't have any clear symmetry. Thus linear and nonlinear encoding can produce quite different patterns. It is shown later in Section 13.7 that there are many advantages to the nonlinear decoder in handling noise. Figure 6.23 compares a snapshot of the flow $\mathbf{x}$ along with its reconstruction $\tilde{\mathbf{u}}$ through the autoencoder/decoder network. The reconstruction error $\mathbf{u} - \tilde{\mathbf{u}}$ is also shown.

6.9 Generative Adversarial Networks (GANs)

Deep learning has also produced success in the generation of synthetic data that is indistinguishable from real data. *Generative adversarial networks* (GANs) learn how to produce synthetic data through an adversarial structure whereby two neural networks are trained simultaneously. One neural network, the discriminator, classifies sample data as real or fake. A second neural network, the generator, produces synthetic data from a latent representation that is run through the discriminator to produce a classification of real or fake. The two neural networks are trained simultaneously so that the generator can produce synthetic data, or fake data, that is indistinguishable from real data.

Goodfellow et al. [291] developed the basic architecture of the GAN as illustrated in Fig. 6.24. Mathematically, the architecture considers a set of real data $\mathbf{X} \in \mathbb{R}^{n \times m}$. Each data sample $\mathbf{x}_k \in \mathbb{R}^n$ is used as input to a neural network that

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

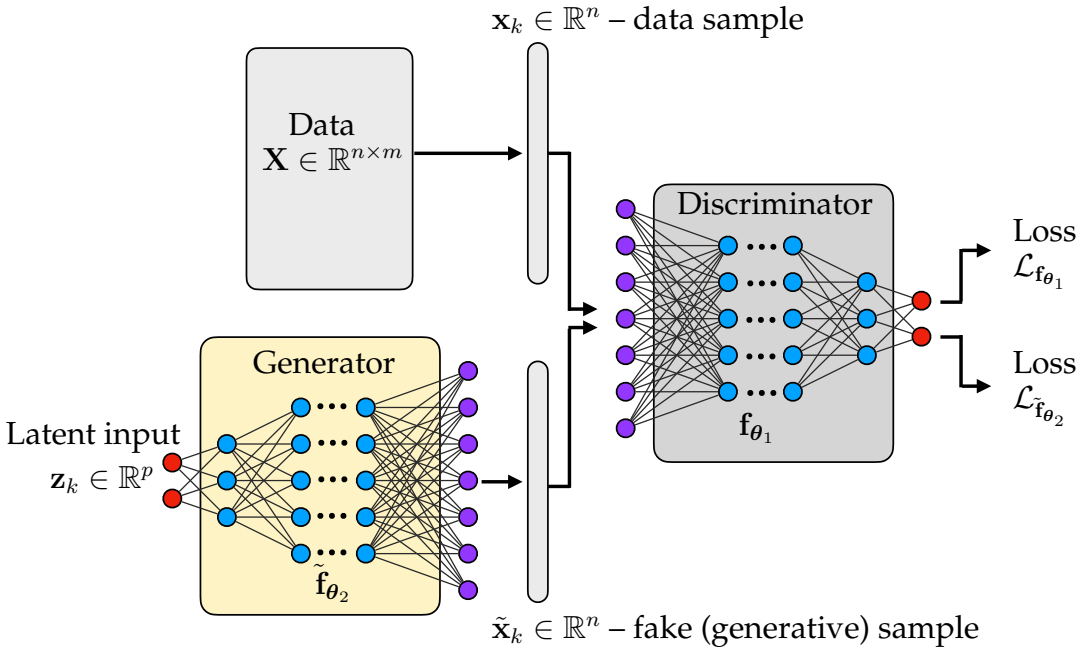


Figure 6.24: Generic GAN architecture that requires the training of two neural networks. Individual samples of data $\mathbf{x}_k \in \mathbb{R}^n$ are used to train a neural network $\mathbf{f}_{\boldsymbol{\theta}_1}$ that maps the input data to a classification label (real or fake) $\mathbf{y}_k \in \mathbb{R}^n$. The generative network $\tilde{\mathbf{f}}_{\boldsymbol{\theta}_2}$ maps a random latent space $\mathbf{z}_k \in \mathbb{R}^p$ to model of the data $\tilde{\mathbf{x}}_k \in \mathbb{R}^n$. The generative data is classified by $\mathbf{f}_{\boldsymbol{\theta}_1}$ as real or fake data by $\tilde{\mathbf{y}}_k \in \mathbb{R}^n$. Backprop is used to update the neural network weights in both networks so that the generator network produces a model of the data $\tilde{\mathbf{x}}_k \in \mathbb{R}^n$ that is indistinguishable from real data $\mathbf{x}_k \in \mathbb{R}^n$.

maps it to a classification $\mathbf{y}_k$ of real or fake:

$$\mathbf{y}_k = \mathbf{f}_{\boldsymbol{\theta}_1}(\mathbf{x}_k). \tag{6.44}$$

Note that the output, for instance, could be given by $\mathbf{y}_k = [1\ 0]^T$ (real) and $\mathbf{y}_k = [0\ 1]^T$ (fake). Without a generative model, this task is trivial, as all the data is, of course, real, so that the label will always be real. A second network is trained to generate synthetic data samples $\tilde{\mathbf{x}}_k$. Specifically, the goal is to make data $\tilde{\mathbf{x}}_k$ indistinguishable from real data $\mathbf{x}_k$. This is done by constructing an input latent space $\mathbf{z}_k$ for the second neural network that generates the synthetic data

$$\tilde{\mathbf{x}}_k = \tilde{\mathbf{f}}_{\boldsymbol{\theta}_2}(\mathbf{z}_k). \tag{6.45}$$

The latent space $\mathbf{z}_k$ is typically a random vector. By training the network, the random vector then produces fake data $\tilde{\mathbf{x}}_k$. The fake data is also then used as

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

an input to the discrimination network

$$\tilde{\mathbf{y}}_k = \mathbf{f}_{\boldsymbol{\theta}_1}(\tilde{\mathbf{x}}_k). \tag{6.46}$$

The output classification vector labels each synthetic data sample as real or fake. Initially, these vectors are largely labeled as fakes. However, by training $\tilde{\mathbf{f}}_{\boldsymbol{\theta}_2}$ and $\mathbf{f}_{\boldsymbol{\theta}_1}$ jointly, the generator network can learn to produce a model that minimizes the number of generated models $\tilde{\mathbf{x}}_k$ that are labeled as fake. Two loss functions are simultaneously considered: (i) $\mathcal{L}_{\mathbf{f}_{\boldsymbol{\theta}_1}}$, which maximizes the probability of assigning the correct label to both training examples $\mathbf{x}_k$ and samples from the generator $\tilde{\mathbf{x}}_k$; and (ii) $\mathcal{L}_{\tilde{\mathbf{f}}_{\boldsymbol{\theta}_2}}$, which minimizes the number of synthetic data labeled as fakes. Thus the second loss function must compute

$$\tilde{\mathbf{y}}_k = \mathbf{f}_{\boldsymbol{\theta}_1}(\tilde{\mathbf{x}}_k) = \mathbf{f}_{\boldsymbol{\theta}_1}(\tilde{\mathbf{f}}_{\boldsymbol{\theta}_2}(\mathbf{z}_k)) \tag{6.47}$$

in order to produce the labels $\tilde{\mathbf{y}}_k$, which are all desired to be considered real. A highly successful outcome would mean that the true data $\mathbf{y}_k$ and synthetic labels $\tilde{\mathbf{y}}_k$ are all classified as real labels.

GANs gained significant popularity due to their ability to generate *deep fakes* (images, video, and speech) that are difficult to distinguish from reality. It is one of the more controversial neural network architectures to have been developed to date. However, it also has advantages when applied to various disciplines in science and engineering. To highlight one of the uses of GAN for scientific purposes, we consider the use of GANs for the super-resolution of turbulent flow physics [202] (see Fig.6.25). In the application of super-resolution, the latent space no longer consists of random inputs. Rather, the latent space $\mathbf{z}_k$ is the low-resolution flow field. Thus the generator network is trained to produce high-resolution flow physics $\tilde{\mathbf{x}}_k$ that is indistinguishable from real data $\mathbf{x}_k$. The generator ($\tilde{\mathbf{f}}_{\boldsymbol{\theta}_2}$) and discriminator ($\mathbf{f}_{\boldsymbol{\theta}_1}$) neural networks are quite sophisticated, being composed of convolutional layers (CONV), parametric ReLU (PReLU), batch normalization (BN), and leaky ReLU (LReLU) layers. Deng et al. [202] show that the GAN is capable of producing high-dimensional reconstructions using low-resolution measurements, thus showing the effectiveness of the method. More broadly, one can imagine using GANs to generate synthetic data that can be useful in various scientific and engineering applications where high-resolution fields are expensive.

6.10 The Diversity of Neural Networks

There are a wide variety of NN architectures, with only a few of the most dominant architectures considered thus far. This chapter (and book) does not attempt to give a comprehensive assessment of the state of the art in neural networks. Rather, our focus is on illustrating some of the key concepts and enabling mathematical architectures that have led NNs to a dominant position in

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

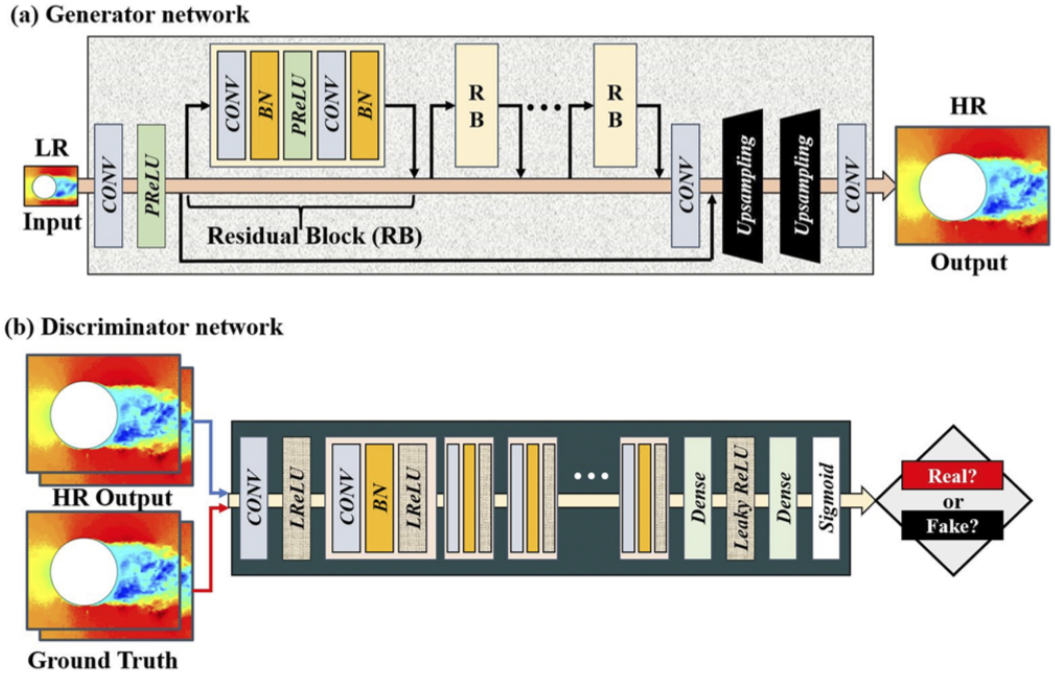


Figure 6.25: Architecture for the super-resolution reconstruction of turbulent velocity fields. In this case, the latent variable space $\mathbf{z}_k$ of Fig. 6.24 is the low-resolution (LR) version of the flow field. The network is trained so that the LR flow field can produce a synthetic version of the high-resolution (HR) flow field $\tilde{\mathbf{x}}_k$ that is indistinguishable from the real data. The generator ($\tilde{\mathbf{f}}_{\boldsymbol{\theta}_2}$) and discriminator ($\mathbf{f}_{\boldsymbol{\theta}_1}$) neural network architectures are composed of convolutional layers (CONV), parametric ReLU (PReLU), batch normalization (BN), and leaky ReLU (LReLU). (From Deng et al. [202]).

modern data science. For a more in-depth review, please see Goodfellow et al. [290]. However, to conclude this chapter, we would like to highlight some of the NN architectures that are used in practice for various data science tasks. This overview is inspired by the *neural network zoo* as highlighted by Fjodor van Veen of the Asimov Institute (`www.asimovinstitute.org`).

The neural network zoo highlights some of the different architectural structures around NNs. Some of the networks highlighted are commonly used across industry, while others serve niche roles for specific applications. Regardless, this demonstrates the tremendous variability and research effort focused on NNs as a core data science tool. Figure 6.26 highlights the prototype structures to be discussed in what follows. Note that the bottom panel has a key to the different type of nodes in the network, including input cells, output cells, and hidden cells. Additionally, the hidden layer NN cells can have memory effects, kernel structures, and/or convolution/pooling. For each NN architecture, a brief

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

description is given along with the original paper proposing the technique.

Perceptron

The first mathematical model of NNs by Fukushima was termed the Neocognitron in 1980 [260]. His model had a single layer with a single output cell called the perceptron, which made a categorial decision based on the sign of the output. Figure 6.2 shows this architecture to classify between dogs and cats. The perceptron is an algorithm for supervised learning of binary classifiers.

Feedforward (FF)

Feedforward networks connect the input layer to the output layer by forming connections between the units so that they do not form a cycle. Figure 6.1 has already shown a version of this architecture where the information simply propagates from left to right in the network. It is often the workhorse of supervised learning where the weights are trained so as to best classify a given set of data. A feedforward network was used in Figs. 6.5 and 6.15 for training a classifier for dogs versus cats and for predicting time-steps of the Lorenz attractor, respectively. An important subclass of feedforward networks is *deep feedforward* (DFF) NNs. DFFs simply put together a larger number of hidden layers, typically 7–10 layers, to form the NN. A second important class of FF is the *radial basis network*, which uses radial basis functions as the activation units [121]. Like any FF network, radial basis function networks have many uses, including function approximation, time-series prediction, classification, and control.

Recurrent Neural Network (RNN)

Illustrated in Fig. 6.26(a), RNNs are characterized by connections between units that form a directed graph along a sequence. This allows an RNN to exhibit dynamic temporal behavior for a time sequence [230]. Unlike feedforward neural networks, RNNs can use their internal state (memory) to process sequences of inputs. The prototypical architecture in Fig. 6.26(a) shows that each cell feeds back on itself. This self-interaction, which is not part of the FF architecture, allows for a variety of innovations. Specifically, it allows for time delays and/or feedback loops. Such controlled states are referred to as gated state or gated memory, and are part of two key innovations: *long short-term memory* (LSTM) networks [331] and *gated recurrent units* (GRU) [177]. LSTM is of particular importance, as it revolutionized speech recognition, setting a variety of performance records and outperforming traditional models in a variety of speech applications. GRUs are a variation of LSTMs that have been demonstrated to

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

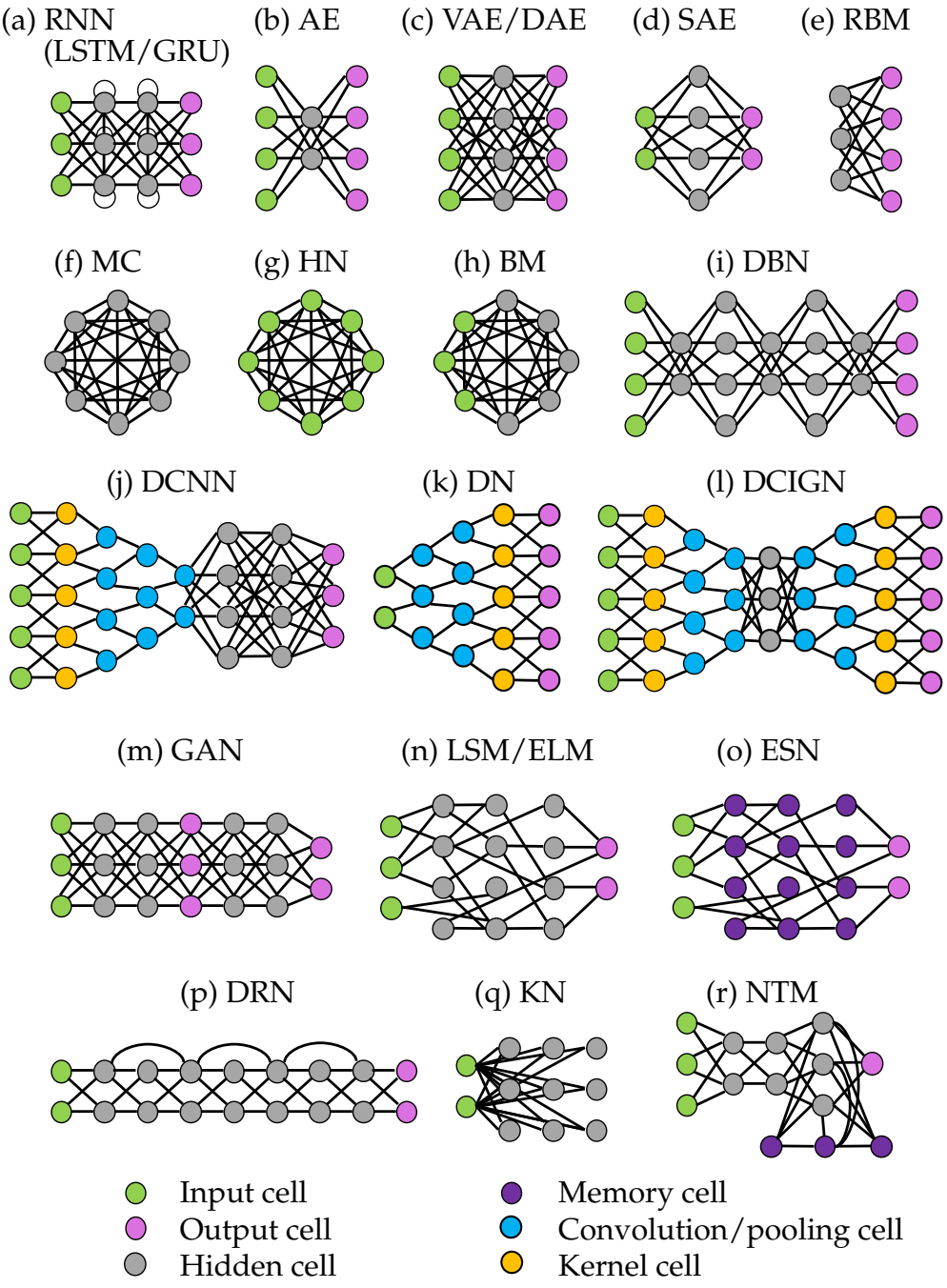


Figure 6.26: Neural network architectures commonly considered in the literature. The NNs comprise input nodes, output nodes, and hidden nodes. Additionally, the nodes can have memory, perform convolution and/or pooling, and perform a kernel transformation. Each network and their acronym are explained in the text.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

exhibit better performance on smaller data sets.

Autoencoder (AE)

The aim of an autoencoder, represented in Fig. 6.26(b), is to learn a representation (encoding) for a set of data, typically for the purpose of dimensionality reduction. For AEs, the input and output cells are matched so that the AE is essentially constructed to be a nonlinear transform into and out of a new representation, acting as an approximate identity map on the data. Thus AEs can be thought of as a generalization of linear dimensionality reduction techniques such as PCA. AEs can potentially produce nonlinear PCA representations of the data, or nonlinear manifolds on which the data should be embedded [99]. Since most data lives in nonlinear subspaces, AEs are an important class of NN for data science, with many innovations and modifications. Three important modifications of the standard AE are commonly used. The *variational autoencoder* (VAE) [387] (shown in Fig. 6.26(c)) is a popular approach to unsupervised learning of complicated distributions. By making strong assumptions concerning the distribution of latent variables, it can be trained using standard gradient descent in order to provide a good assessment of data in an unsupervised fashion. The *de-noising autoencoder* (DAE) [734] (shown in Fig. 6.26(c)) takes a partially corrupted input during training to recover the original undistorted input. Thus noise is intentionally added to the input in order to learn the nonlinear embedding. Finally, the *sparse autoencoder* (SAE) [585] (shown in Fig. 6.26(d)) imposes sparsity on the hidden units during training, while having a larger number of hidden units than inputs, so that an autoencoder can learn useful structures in the input data. Sparsity is typically imposed by thresholding all but the few strongest hidden unit activations.

Markov Chain (MC)

A Markov chain is a stochastic model describing a sequence of possible events in which the probability of each event depends only on the state attained in the previous event. So, although not formally a NN, it shares many common features with RNNs. Markov chains are standard even in undergraduate probability and statistics courses. Figure 6.26(f) shows the basic architecture where each cell is connected to the other cells by a probability model for a transition.

Hopfield Network (HN)

A Hopfield network is a form of RNN which was popularized by John Hopfield in 1982 for understanding human memory [337]. Figure 6.26(g) shows the basic architecture of an all-to-all connected network where each node can act as

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

an input cell. The network serves as a trainable content-addressable *associative* memory system with binary threshold nodes. Given an input, it is iterated on the network with a guarantee to converge to a local minimum. Sometimes it converges to a false pattern, or memory (wrong local minimum), rather than the stored pattern (expected local minimum).

Boltzmann Machine (BM)

The Boltzmann machine, sometimes called a stochastic Hopfield network with hidden units, is a stochastic, generative counterpart of the Hopfield network. They were one of the first neural networks capable of learning internal representations, and are able to represent and (given sufficient time) solve difficult combinatoric problems [327]. Figure 6.26(h) shows the structure of the BM. Note that, unlike Markov chains (which have no input units) or Hopfield networks (where all cells are inputs), the BM is a hybrid which has a mixture of input cells and hidden units. Boltzmann machines are intuitively appealing due to their resemblance to the dynamics of simple physical processes. They are named after the Boltzmann distribution in statistical mechanics, which is used in their sampling function.

Restricted Boltzmann Machine (RBM)

Introduced under the name *Harmonium* by Paul Smolensky in 1986 [666], RBMs have been proposed for dimensionality reduction, classification, collaborative filtering, feature learning, and topic modeling. They can be trained for either supervised or unsupervised tasks. G. Hinton helped bring them to prominence by developing fast algorithms for evaluating them [519]. RBMs are a subset of BMs where restrictions are imposed on the NN such that nodes in the NN must form a bipartite graph (see Fig. 6.26(e)). Thus a pair of nodes from each of the two groups of units (commonly referred to as the "visible" and "hidden" units, respectively) may have a symmetric connection between them; there are no connections between nodes within a group. RBMs can be used in deep learning networks and deep belief networks by stacking RBMs and optionally fine-tuning the resulting deep network with gradient descent and backpropagation.

Deep Belief Network (DBN)

DBNs are a generative graphical model that are composed of multiple layers of latent hidden variables, with connections between the layers but not between units within each layer [73]. Figure 6.26(i) shows the architecture of the DBN. The training of the DBNs can be done stack by stack from AE or RBM layers. Thus each of these layers only has to learn to encode the previous network,

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

which is effectively a greedy training algorithm for finding locally optimal solutions. Thus DBNs can be viewed as a composition of simple, unsupervised networks such as RBMs and AEs where each sub-network's hidden layer serves as the visible layer for the next.

Deep Convolutional Neural Network (DCNN)

DCNNs are the workhorse of computer vision and have already been considered in this chapter. They are abstractly represented in Fig. 6.26(j), and in a more specific fashion in Fig. 6.12. Their impact and influence on computer vision cannot be overestimated. They were originally developed for document recognition [433].

Deconvolutional Network (DN)

Deconvolutional networks, shown in Fig. 6.26(k), are essentially a reverse of DCNNs [770]. The mathematical structure of DNs permits the unsupervised construction of hierarchical image representations. These representations can be used for both low-level tasks such as de-noising, as well as providing features for object recognition. Each level of the hierarchy groups information from the level beneath to form more complex features that exist over a larger scale in the image. As with DCNNs, it is well suited for computer vision tasks.

Deep Convolutional Inverse Graphics Network (DCIGN)

The DCIGN is a form of VAE that uses DCNNs for the encoding and decoding [417]. As with the AE/VAE/SAE structures, the output layer shown in Fig. 6.26(l) is constrained to match the input layer. DCIGNs combine the power of DCNNs with VAEs, which provides a formative mathematical architecture for computer vision and image processing.

Generative Adversarial Network (GAN)

In an innovative modification of NNs, the GAN architecture of Fig. 6.26(m) trains two networks simultaneously [291]. The networks, which are often a combination of DCNNs and/or FFs, train by one of the networks generating content which the other attempts to judge. Specifically, one network generates candidates and the other evaluates them. Typically, the generative network learns to map from a latent space to a particular data distribution of interest, while the discriminative network discriminates between instances from the true data distribution and candidates produced by the generator. The generative network's training objective is to increase the error rate of the discrimina-

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

tive network (i.e., "fool" the discriminator network by producing novel synthesized instances that appear to have come from the true data distribution). The GAN architecture has produced interesting results in computer vision for producing synthetic data, such as images and movies.

Liquid State Machine (LSM)

The LSM shown in Fig. 6.26(n) is a particular kind of spiking neural network [469]. An LSM consists of a large collection of nodes, each of which receives time-varying input from external sources (the inputs) as well as from other nodes. Nodes are randomly connected to each other. The recurrent nature of the connections turns the time-varying input into a spatio-temporal pattern of activations in the network nodes. The spatio-temporal patterns of activation are read out by linear discriminant units. This architecture is motivated by spiking neurons in the brain, thus helping understand how information processing and discrimination might happen using spiking neurons.

Extreme Learning Machine (ELM)

With the same underlying architecture of an LSM shown in Fig. 6.26(n), the ELM is a FF network for classification, regression, clustering, sparse approximation, compression, and feature learning with a single layer or multiple layers of hidden nodes, where the parameters of hidden nodes (not just the weights connecting inputs to hidden nodes) need not be tuned. These hidden nodes can be randomly assigned and never updated, or can be inherited from their ancestors without being changed. In most cases, the output weights of hidden nodes are usually learned in a single step, which essentially amounts to learning a linear model [150].

Echo State Network (ESN)

ESNs are RNNs with a sparsely connected hidden layer (with typically 1% connectivity). The connectivity and weights of hidden neurons have memory and are fixed and randomly assigned (see Fig. 6.26(o)). Thus, like LSMs and ELMs, they are not fixed into a well-ordered layered structure. The weights of output neurons can be learned so that the network can generate specific temporal patterns [347].

Deep Residual Network (DRN)

DRNs took the deep learning world by storm when Microsoft Research released Deep Residual Learning for Image Recognition [317]. These networks

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

led to first-place-winning entries in all five main tracks of the ImageNet and COCO 2015 competitions, which covered image classification, object detection, and semantic segmentation. The robustness of residual networks (ResNets) has since been proven by various visual recognition tasks and by non-visual tasks involving speech and language. DRNs are very deep FF networks where there are extra connections that pass from one layer to a layer two to five layers downstream. This then carries input from an earlier stage to a future stage. These networks can be 150 layers deep, which is only abstractly represented in Fig. 6.26(p).

Kohonen Network (KN)

Kohonen networks are also known as self-organizing feature maps [400]. KNs use competitive learning to classify data without supervision. Input is presented to the KN as in Fig. 6.26(q), after which the network assesses which of the neurons closely match that input. These self-organizing maps differ from other NNs, as they apply competitive learning as opposed to error-correction learning (such as backpropagation with gradient descent), and in the sense that they use a neighborhood function to preserve the topological properties of the input space. This makes KNs useful for low-dimensional visualization of high-dimensional data.

Neural Turing Machine (NTM)

The NTM architecture implements a neural network controller coupled to an external memory resource (see Fig. 6.26(r)), which it interacts with through attentional mechanisms [294]. The memory interactions are differentiable end-to-end, making it possible to optimize them using gradient descent. Pairing the NTM with an LSTM controller can infer simple algorithms such as copying, sorting, and associative recall from input and output examples.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Suggested Reading

Texts

(1) Deep learning, by I. Goodfellow, Y. Bengio, and A. Courville, 2016 [290].

(2) Neural networks for pattern recognition, by C. M. Bishop, 1995 [90].

Papers and reviews

(1) Deep learning, by Y. LeCun, Y. Bengio, and G. Hinton, *Nature*, 2015 [432].

(2) Understanding deep convolutional networks, by S. Mallat, *Philosophical Transactions of the Royal Society of London A*, 2016 [476].

(3) Deep learning: Mathematics and neuroscience, by T. Poggio, *Views & Reviews, McGovern Center for Brains, Minds and Machines*, 2016 [566].

(4) ImageNet classification with deep convolutional neural, by A. Krizhevsky, I. Sutskever, and G. Hinton, *Advances in Neural Information Processing Systems*, 2012 [414].

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Homework

Exercise 6-1. Download the code base for solving (i) a reaction–diffusion system of equations, and (ii) the Kuramoto–Sivashinsky (KS) equation.

(a) Train a NN that can advance the solution from t to $t + \Delta t$ for the KS equation.

(b) Compare your evolution trajectories for your NN against using the ODE time-stepper provided with different initial conditions.

(c) For the reaction–diffusion system, first project to a low-dimensional subspace via the SVD and see how forecasting works in the low-rank variables.

For the Lorenz equations, consider the following.

(d) Train an NN to advance the solution from t to $t+\Delta t$ for $\rho = 10$, 28, and 40. Now see how well your NN works for future state prediction for $\rho = 17$ and $\rho = 35$.

(e) See if you can train your NN to identify (for $\rho = 28$) when a transition from one lobe to another is imminent. Determine how far in advance you can make this prediction. (Note: You will have to label the transitions in a test set in order to do this task.)

Exercise 6-2. Consider time-series data acquired from power grid loads, specifically: T. V. Jensen and P. Pinson. Re-Europe, a large-scale data set for modeling a highly renewable European electricity system. *Scientific Data*, 4:170175, 2017. Compare the forecasting capabilities of the following neural networks on the power grid data: (i) a feedforward neural network; (ii) an LSTM; (iii) an RNN; and (iv) an echo state network. Consider the performance of each under cross-validation for forecasting ranges of Δt into the future and $N\Delta t$ into the future (where $N \gg 1$).

Exercise 6-3. Download the flow around the cylinder data. Using the first $P\%$ of the temporal snapshots, forecast the remaining $(100 - P)\%$ future state data. Do this by training a neural network on the high-dimensional data and using: (i) a feedforward neural network; (ii) an LSTM; (iii) an RNN; and (iv) an echo state network. Determine the performance of the algorithms as a function of decreasing data P.

Redo the forecasting calculations by training a model in the reduced subspace $\mathbf{U}$ from the singular value decomposition. Evaluate the forecasting performance

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

as a function of the percentage of the training data P and the rank of the reduced space r.

Exercise 6-4. Generate simulation data for the Kuramoto–Sivashinsky (KS) equation in three distinct parameter regimes where non-trivial spatio-temporal dynamics occurs. Using a convolutional neural network, map the high-dimensional snapshots of the system to a classification of the system into one of the three parameter regimes. Evaluate the performance of the classification scheme on test data as a function of different convolutional window sizes and stride lengths. For the best performance, what is the convolutional window size and what spatial length scale is extracted to make the classification decision?

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Part III

Dynamics and Control

Chapter 7

Data-Driven Dynamical Systems

Dynamical systems provide a mathematical framework to describe the world around us, modeling the rich interactions between quantities that co-evolve in time. Formally, dynamical systems concern the analysis, prediction, and understanding of the behavior of systems of differential equations or iterative mappings that describe the evolution of the state of a system. This formulation is general enough to encompass a staggering range of phenomena, including those observed in classical mechanical systems, electrical circuits, turbulent fluids, climate science, finance, ecology, social systems, neuroscience, epidemiology, and nearly every other system that evolves in time.

Modern dynamical systems began with the seminal work of Poincaré on the chaotic motion of planets. It is rooted in classical mechanics, and may be viewed as the culmination of hundreds of years of mathematical modeling, beginning with Newton and Leibniz. The full history of dynamical systems is too rich for these few pages, having captured the interest and attention of the greatest minds for centuries, and having been applied to countless fields and challenging problems. Dynamical systems provide one of the most complete and well-connected fields of mathematics, bridging diverse topics from linear algebra and differential equations, to topology, numerical analysis, and geometry. Dynamical systems have become central in the modeling and analysis of systems in nearly every field of the engineering, physical, and life sciences.

Modern dynamical systems are currently undergoing a renaissance, with analytical derivations and first-principles models giving way to data-driven approaches. The confluence of big data and machine learning is driving a paradigm shift in the analysis and understanding of dynamical systems in science and engineering. Data are abundant, while physical laws or governing equations remain elusive, as is true for problems in climate science, finance, epidemiology, and neuroscience. Even in classical fields, such as optics and turbulence, where governing equations do exist, researchers are increasingly turning toward data-driven analysis. Many critical data-driven problems, such as predicting climate change, understanding cognition from neural recordings, pre-

dicting and suppressing the spread of disease, or controlling turbulence for energy-efficient power production and transportation, are primed to take advantage of progress in the data-driven discovery of dynamics.

In addition, the classical geometric and statistical perspectives on dynamical systems are being complemented by a third *operator-theoretic* perspective, based on the evolution of measurements of the system. This so-called *Koopman* operator theory is poised to capitalize on the increasing availability of measurement data from complex systems. Moreover, Koopman theory provides a path to identify intrinsic coordinate systems to represent nonlinear dynamics in a linear framework. Obtaining linear representations of strongly nonlinear systems has the potential to revolutionize our ability to predict and control these systems.

This chapter presents a modern perspective on dynamical systems in the context of current goals and open challenges. Data-driven dynamical systems is a rapidly evolving field, and therefore we focus on a mix of established and emerging methods that are driving current developments. In particular, we will focus on the key challenges of discovering dynamics from data and finding data-driven representations that make nonlinear systems amenable to linear analysis.

7.1 Overview, Motivations, and Challenges

Before summarizing recent developments in data-driven dynamical systems, it is important to first provide a mathematical introduction to the notation and summarize key motivations and open challenges in dynamical systems.

Dynamical Systems

Throughout this chapter, we will consider dynamical systems of the form

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{x}(t) = \mathbf{f}(\mathbf{x}(t), t; \boldsymbol{\beta}), \tag{7.1}$$

where $\mathbf{x}$ is the state of the system and $\mathbf{f}$ is a vector field that possibly depends on the state $\mathbf{x}$, time t, and a set of parameters $\boldsymbol{\beta}$.

For example, consider the Lorenz equations [460]

$$\dot{x} = \sigma(y - x), \tag{7.2a}$$
$$\dot{y} = x(\rho - z) - y, \tag{7.2b}$$
$$\dot{z} = xy - \beta z, \tag{7.2c}$$

with parameters $\sigma = 10$, $\rho = 28$, and $\beta = 8/3$. A trajectory of the Lorenz system is shown in Fig. 7.1. In this case, the state vector is $\mathbf{x} = \begin{bmatrix} x & y & z \end{bmatrix}^T$ and the parameter vector is $\boldsymbol{\beta} = \begin{bmatrix} \sigma & \rho & \beta \end{bmatrix}^T$.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

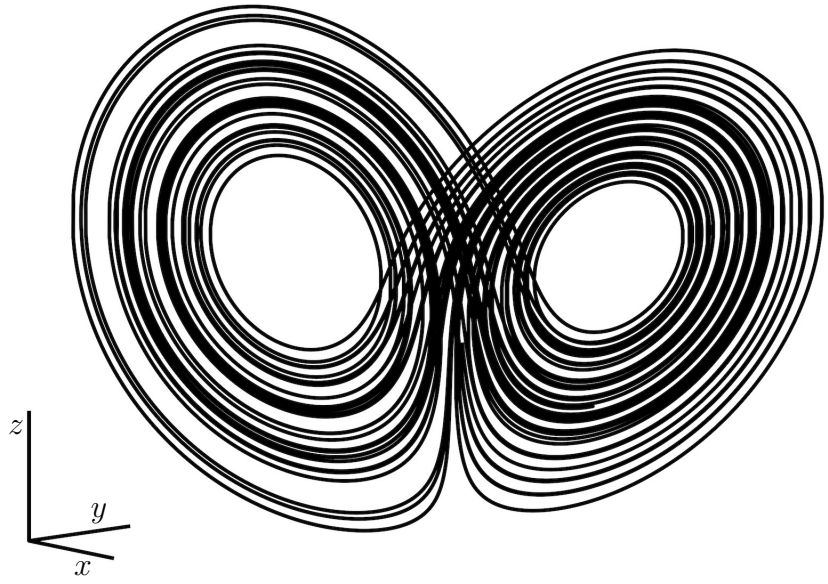


Figure 7.1: Chaotic trajectory of the Lorenz system from (7.2).

The Lorenz system is among the simplest and most well-studied dynamical systems that exhibit chaos, which is characterized as a sensitive dependence on initial conditions. Two trajectories with nearby initial conditions will rapidly diverge in behavior, and, after long times, only statistical statements can be made.

It is simple to simulate dynamical systems, such as the Lorenz system. First, the vector field $\mathbf{f}(\mathbf{x}, t; \boldsymbol{\beta})$ is defined in the function **lorenz** in Code 7.1.

Code 7.1: [MATLAB] Define Lorenz vector field.

```
function dx = lorenz(t,x,Beta)
dx = [
Beta(1)*(x(2)-x(1));
x(1)*(Beta(2)-x(3))-x(2);
x(1)*x(2)-Beta(3)*x(3);
];
```

Code 7.1: [Python] Define Lorenz vector field.

```
def lorenz(x_y_z, t0, sigma=sigma, beta=beta, rho=rho):
    x, y, z = x_y_z
    return [sigma*(y-x), x*(rho-z)-y, x*y-beta*z]
```

In Code 7.2, we define the system parameters $\boldsymbol{\beta}$, initial condition $\mathbf{x}_0$, and timespan, and simulate the equations with a fourth-order Runge–Kutta integration scheme with adaptive time-step; in MATLAB we use the **ode45** command and in Python we use the **integrate.odeint** command.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Code 7.2: [MATLAB] Define Lorenz system parameters and simulate with Runge–Kutta integrator.

```
Beta = [10; 28; 8/3]; % Lorenz's parameters (chaotic)
x0=[0; 1; 20];  % Initial condition
dt = 0.001;
tspan=dt:dt:50;
options = odeset('RelTol',1e-12,'AbsTol',1e-12*ones(1,3));

[t,x]=ode45(@(t,x) lorenz(t,x,Beta),tspan,x0,options);
plot3(x(:,1),x(:,2),x(:,3));
```

Code 7.2: [Python] Define Lorenz system parameters and simulate with Runge–Kutta integrator.

```
beta = 8/3
sigma = 10
rho = 28
x0 = (0,1,20)
dt = 0.001
t = np.arange(0,50+dt,dt)

x_t = integrate.odeint(lorenz, x0, t,rtol=10**(-12),atol
   =10**(-12)*np.ones_like(x0))
x, y, z = x_t.T
plt.plot(x, y, z,linewidth=1)
```

We will often consider the simpler case of an autonomous system without time dependence or parameters:

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{x}(t) = \mathbf{f}(\mathbf{x}(t)). \tag{7.3}$$

In general, $\mathbf{x}(t) \in \mathbf{M}$ is an n-dimensional state that lives on a smooth manifold $\mathbf{M}$, and $\mathbf{f}$ is an element of the tangent bundle $\mathbf{TM}$ of $\mathbf{M}$ so that $\mathbf{f}(\mathbf{x}(t)) \in \mathbf{T}_{\mathbf{x}(t)}\mathbf{M}$. However, we will typically consider the simpler case where $\mathbf{x}$ is a vector, $\mathbf{M} = \mathbb{R}^n$, and $\mathbf{f}$ is a Lipschitz continuous function, guaranteeing existence and uniqueness of solutions to (7.3). For the more general formulation, see [4].

Discrete-Time Systems

We will also consider the discrete-time dynamical system

$$\mathbf{x}_{k+1} = \mathbf{F}(\mathbf{x}_k). \tag{7.4}$$

Also known as a *map*, the discrete-time dynamics are more general than the continuous-time formulation in (7.3), encompassing discontinuous and hybrid systems as well.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

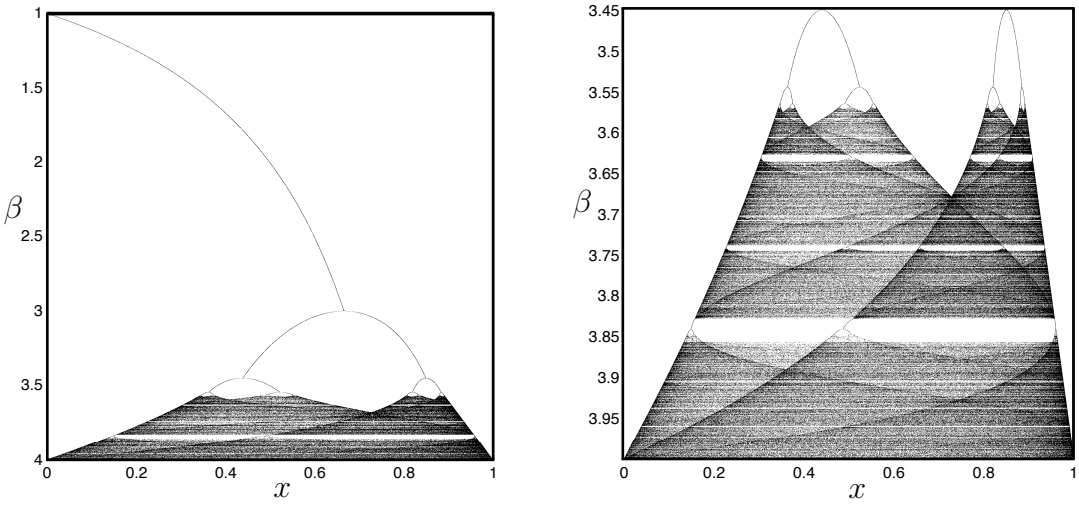


Figure 7.2: Attracting sets of the logistic map for varying parameter β.

For example, consider the logistic map:

$$x_{k+1} = \beta x_k(1 - x_k). \tag{7.5}$$

As the parameter β is increased, the attracting set becomes increasingly complex, as shown in Fig. 7.2. A series of period-doubling bifurcations occur until the attracting set becomes fractal.

Discrete-time dynamics may be induced from continuous-time dynamics, where $\mathbf{x}_k$ is obtained by sampling the trajectory in (7.3) discretely in time, so that $\mathbf{x}_k = \mathbf{x}(k\Delta t)$. The discrete-time propagator $\mathbf{F}_{\Delta t}$ is now parameterized by the time-step Δt. For an arbitrary time t, the *flow map* $\mathbf{F}_t$ is defined as

$$\mathbf{F}_t(\mathbf{x}(t_0)) = \mathbf{x}(t_0) + \int_{t_0}^{t_0+t} \mathbf{f}(\mathbf{x}(\tau))\,\mathrm{d}\tau. \tag{7.6}$$

The discrete-time perspective is often more natural when considering experimental data and digital control.

Linear Dynamics and Spectral Decomposition

Whenever possible, it is desirable to work with linear dynamics of the form

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{x} = \mathbf{A}\mathbf{x}. \tag{7.7}$$

Linear dynamical systems admit closed-form solutions, and there are a wealth of techniques for the analysis, prediction, numerical simulation, estimation, and control of such systems. The solution of (7.7) is given by

$$\mathbf{x}(t_0 + t) = e^{\mathbf{A}t}\mathbf{x}(t_0). \tag{7.8}$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

The dynamics are entirely characterized by the eigenvalues and eigenvectors of the matrix $\mathbf{A}$, given by the *spectral decomposition* (eigendecomposition) of $\mathbf{A}$:

$$\mathbf{AT} = \mathbf{T\Lambda}. \tag{7.9}$$

When $\mathbf{A}$ has n distinct eigenvalues, then $\mathbf{\Lambda}$ is a diagonal matrix containing the eigenvalues λ_j and $\mathbf{T}$ is a matrix whose columns are the linearly independent eigenvectors $\boldsymbol{\xi}_j$ associated with eigenvalues λ_j. In this case, it is possible to write $\mathbf{A} = \mathbf{T\Lambda T}^{-1}$, and the solution in (7.8) becomes

$$\mathbf{x}(t_0 + t) = \mathbf{T}e^{\mathbf{\Lambda} t}\mathbf{T}^{-1}\mathbf{x}(t_0). \tag{7.10}$$

More generally, in the case of repeated eigenvalues, the matrix $\mathbf{\Lambda}$ will consist of Jordan blocks [562]. See Section 8.2 for a detailed derivation of the above arguments for control systems. Note that the continuous-time system gives rise to a discrete-time dynamical system, with $\mathbf{F}_t$ given by the solution map $\exp(\mathbf{A}t)$ in (7.8). In this case, the discrete-time eigenvalues are given by $e^{\lambda t}$.

The matrix $\mathbf{T}^{-1}$ defines a transformation, $\mathbf{z} = \mathbf{T}^{-1}\mathbf{x}$, into intrinsic eigenvector coordinates, $\mathbf{z}$, where the dynamics become decoupled:

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{z} = \mathbf{\Lambda z}. \tag{7.11}$$

In other words, each coordinate, z_j, only depends on itself, with simple dynamics given by

$$\frac{\mathrm{d}}{\mathrm{d}t}z_j = \lambda_j z_j. \tag{7.12}$$

Thus, it is highly desirable to work with linear systems, since it is possible to easily transform the system into eigenvector coordinates where the dynamics become decoupled. No such closed-form solution or simple linear change of coordinates exist in general for nonlinear systems, motivating many of the directions described in this chapter.

Goals and Challenges in Modern Dynamical Systems

As we generally use dynamical systems to model real-world phenomena, there are a number of high-priority goals associated with the analysis of dynamical systems:

(a) **Future state prediction.** In many cases, such as meteorology and climatology, we seek predictions of the future state of a system. Long-time predictions may still be challenging.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

(b) **Design and optimization.** We may seek to tune the parameters of a system for improved performance or stability, for example through the placement of fins on a rocket.

(c) **Estimation and control.** It is often possible to actively control a dynamical system through feedback, using measurements of the system to inform actuation to modify the behavior. In this case, it is often necessary to estimate the full state of the system from limited measurements.

(d) **Interpretability and physical understanding.** Perhaps a more fundamental goal of dynamical systems is to provide physical insight and interpretability into a system's behavior through analyzing trajectories and solutions to the governing equations of motion.

Real-world systems are generally nonlinear and exhibit multi-scale behavior in both space and time. It must also be assumed that there is uncertainty in the equations of motion, in the specification of parameters, and in the measurements of the system. Some systems are more sensitive to this uncertainty than others, and probabilistic approaches must be used. Increasingly, it is also the case that the basic equations of motion are not specified and they might be intractable to derive from first principles.

This chapter will cover recent data-driven techniques to identify and analyze dynamical systems. The majority of this chapter addresses two primary challenges of modern dynamical systems:

1. **Nonlinearity.** Nonlinearity remains a primary challenge in analyzing and controlling dynamical systems, giving rise to complex global dynamics. We saw above that linear systems may be completely characterized in terms of the spectral decomposition (i.e., eigenvalues and eigenvectors) of the matrix $\mathbf{A}$, leading to general procedures for prediction, estimation, and control. No such overarching framework exists for nonlinear systems, and developing this general framework is a mathematical grand challenge of the twenty-first century.

   The leading perspective on nonlinear dynamical systems considers the geometry of subspaces of local linearizations around fixed points and periodic orbits, global heteroclinic and homoclinic orbits connecting these structures, and more general attractors [334]. This geometric theory, originating with Poincaré, has transformed how we model complex systems, and its success can be largely attributed to theoretical results, such as the Hartman–Grobman theorem, which establish when and where it is possible to approximate a nonlinear system with linear dynamics. Thus, it is often possible to apply the wealth of linear analysis techniques in a small neighborhood of a fixed point or periodic orbit. Although the geometric

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

perspective provides quantitative locally linear models, global analysis has remained largely qualitative and computational, limiting the theory of nonlinear prediction, estimation, and control away from fixed points and periodic orbits.

2. **Unknown dynamics.** Perhaps an even more central challenge arises from the lack of known governing equations for many modern systems of interest. Increasingly, researchers are tackling more complex and realistic systems, such as are found in neuroscience, epidemiology, and ecology. In these fields, there is a basic lack of known *physical laws* that provide first principles from which it is possible to derive equations of motion. Even in systems where we do know the governing equations, such as turbulence, protein folding, and combustion, we struggle to find patterns in these high-dimensional systems to uncover intrinsic coordinates and coarse-grained variables along which the dominant behavior evolves.

   Traditionally, physical systems were analyzed by making ideal approximations and then deriving simple differential equation models via Newton's second law. Dramatic simplifications could often be made by exploiting symmetries and clever coordinate systems, as highlighted by the success of Lagrangian and Hamiltonian dynamics [3, 486]. With increasingly complex systems, the paradigm is shifting from this classical approach to data-driven methods to discover governing equations.

   All models are approximations, and, with increasing complexity, these approximations often become suspect. Determining what is the correct model is becoming more subjective, and there is a growing need for automated model discovery techniques that illuminate underlying physical mechanisms. There are also often latent variables that are relevant to the dynamics but may go unmeasured. Uncovering these hidden effects is a major challenge for data-driven methods.

Identifying unknown dynamics from data and learning intrinsic coordinates that enable the linear representation of nonlinear systems are two of the most pressing goals of modern dynamical systems. Overcoming the challenges of unknown dynamics and nonlinearity has the promise of transforming our understanding of complex systems, with tremendous potential benefit to nearly all fields of science and engineering.

Throughout this chapter we will explore these issues in further detail and describe a number of the emerging techniques to address these challenges. In particular, there are two key approaches that are defining modern data-driven dynamical systems:

(a) **Operator-theoretic representations.** To address the issue of nonlinearity, operator-theoretic approaches to dynamical systems are becoming in-

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

creasingly used. As we will show, it is possible to represent nonlinear dynamical systems in terms of infinite-dimensional but linear operators, such as the Koopman operator from Section 7.4 that advances measurement functions, and the Perron–Frobenius operator that advances probability densities and ensembles through the dynamics.

(b) **Data-driven regression and machine learning.** As data becomes increasingly abundant, and we continue to investigate systems that are not amenable to first-principles analysis, regression and machine learning are becoming vital tools to discover dynamical systems from data. This is the basis of many of the techniques described in this chapter, including the dynamic mode decomposition (DMD) in Section 7.2, the sparse identification of nonlinear dynamics (SINDy) in Section 7.3, the data-driven Koopman methods in Section 7.5, as well as the use of genetic programming to identify dynamics from data [95, 640].

It is important to note that many of the methods and perspectives described in this chapter are interrelated, and continuing to strengthen and uncover these relationships is the subject of ongoing research. It is also worth mentioning that a third major challenge is the high dimensionality associated with many modern dynamical systems, such as are found in population dynamics, brain simulations, and high-fidelity numerical discretizations of partial differential equations. High dimensionality is addressed extensively in the subsequent chapters on reduced-order models (ROMs).

Finally, several open-source software libraries are being developed for data-driven dynamical systems, including

- PyDMD (`https://github.com/mathLab/PyDMD`);
- PySINDy (`https://github.com/dynamicslab/pysindy`);
- PyKoopman (`https://github.com/dynamicslab/pykoopman`);
- Data-driven dynamical systems toolbox (`https://github.com/sklus/d3s`);
- Deeptime (`https://github.com/deeptime-ml/deeptime`).

7.2 Dynamic Mode Decomposition (DMD)

Dynamic mode decomposition was developed by Schmid [635, 636] in the fluid dynamics community to identify spatio-temporal coherent structures from high-dimensional data. DMD is based on proper orthogonal decomposition (POD), which utilizes the computationally efficient singular value decomposition (SVD),

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

so that it scales well to provide effective dimensionality reduction in high-dimensional systems. In contrast to SVD/POD, which results in a hierarchy of modes based entirely on spatial correlation and energy content, while largely ignoring temporal information, DMD provides a modal decomposition where each mode consists of spatially correlated structures that have the same linear behavior in time (e.g., oscillations at a given frequency with growth or decay). Thus, DMD provides not only dimensionality reduction in terms of a reduced set of modes, but also a model for how these modes evolve in time.

Soon after the development of the original DMD algorithm [635, 636], Rowley, Mezić, and collaborators established an important connection between DMD and Koopman theory [611] (see Section 7.4). DMD may be formulated as an algorithm to identify the best-fit linear dynamical system that advances high-dimensional measurements forward in time [727]. In this way, DMD approximates the Koopman operator restricted to the set of direct measurements of the state of a high-dimensional system. This connection between the computationally straightforward and linear DMD framework and nonlinear dynamical systems has generated considerable interest in these methods [422].

Within a short amount of time, DMD has become a workhorse algorithm for the data-driven characterization of high-dimensional systems. DMD is equally valid for experimental and numerical data, as it is not based on knowledge of the governing equations, but is instead based purely on measurement data. The DMD algorithm may also be seen as connecting the favorable aspects of the SVD (see Chapter 1) for spatial dimensionality reduction and the FFT (see Chapter 2) for temporal frequency identification [173, 422]. Thus, each DMD mode is associated with a particular *eigenvalue* $\lambda = a + ib$, with a particular frequency of oscillation b and growth or decay rate a.

There are many variants of DMD and it is connected to existing techniques from system identification and modal extraction. DMD has become especially popular in recent years, in large part due to its simple numerical implementation and strong connections to nonlinear dynamical systems via Koopman spectral theory. Finally, DMD is an extremely flexible platform, both mathematically and numerically, facilitating innovations related to compressed sensing, control theory, and multi-resolution techniques. These connections and extensions will be discussed at the end of this section.

The DMD Algorithm

Several algorithms have been proposed for DMD, although here we present the *exact* DMD framework developed by Tu et al. [727]. Whereas earlier formulations required uniform sampling of the dynamics in time, the approach presented here works with irregularly sampled data and with concatenated data from several different experiments or numerical simulations. Moreover, the ex-

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

act formulation of Tu et al. provides a precise mathematical definition of DMD that allows for rigorous theoretical results. Finally, exact DMD is based on the efficient and numerically well-conditioned singular value decomposition, as is the original formulation by Schmid [635].

DMD is inherently data-driven, and the first step is to collect a number of pairs of snapshots of the state of a system as it evolves in time. These snapshot pairs may be denoted by $\{(\mathbf{x}(t_k), \mathbf{x}(t'_k))\}_{k=1}^m$, where $t'_k = t_k + \Delta t$, and the time-step Δt is sufficiently small to resolve the highest frequencies in the dynamics. As before, a snapshot may be the state of a system, such as a three-dimensional fluid velocity field sampled at a number of discretized locations, which is reshaped into a high-dimensional column vector. These snapshots are then arranged into two data matrices, $\mathbf{X}$ and $\mathbf{X}'$:

$$\mathbf{X} = \begin{bmatrix} | & | & & | \\ \mathbf{x}(t_1) & \mathbf{x}(t_2) & \cdots & \mathbf{x}(t_m) \\ | & | & & | \end{bmatrix}, \tag{7.13a}$$

$$\mathbf{X}' = \begin{bmatrix} | & | & & | \\ \mathbf{x}(t'_1) & \mathbf{x}(t'_2) & \cdots & \mathbf{x}(t'_m) \\ | & | & & | \end{bmatrix}. \tag{7.13b}$$

The original formulations of Schmid [635] and Rowley et al. [611] assumed uniform sampling in time, so that $t_k = k\Delta t$ and $t'_k = t_k + \Delta t = t_{k+1}$. If we assume uniform sampling in time, we will adopt the notation $\mathbf{x}_k = \mathbf{x}(k\Delta t)$.

The DMD algorithm seeks the leading spectral decomposition (i.e., eigenvalues and eigenvectors) of the best-fit linear operator $\mathbf{A}$ that relates the two snapshot matrices in time:

$$\mathbf{X}' \approx \mathbf{A}\mathbf{X}. \tag{7.14}$$

The best-fit operator $\mathbf{A}$ then establishes a linear dynamical system that best advances snapshot measurements forward in time. If we assume uniform sampling in time, this becomes

$$\mathbf{x}_{k+1} \approx \mathbf{A}\mathbf{x}_k. \tag{7.15}$$

Mathematically, the best-fit operator $\mathbf{A}$ is defined as

$$\mathbf{A} = \underset{\mathbf{A}}{\operatorname{argmin}} \|\mathbf{X}' - \mathbf{A}\mathbf{X}\|_F = \mathbf{X}'\mathbf{X}^\dagger, \tag{7.16}$$

where $\|\cdot\|_F$ is the Frobenius norm and † denotes the pseudo-inverse. The optimized DMD algorithm generalizes the optimization framework of exact DMD to perform a regression to exponential-time dynamics, thus providing an improved computation of the DMD modes and their eigenvalues [27].

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

It is worth noting at this point that the matrix $\mathbf{A}$ in (7.15) closely resembles the Koopman operator in Section 7.4 (see equation (7.63)), if we choose direct linear measurements of the state, so that $\mathbf{g}(\mathbf{x}) = \mathbf{x}$. This connection was originally established by Rowley, Mezić, and collaborators [611], and has sparked considerable interest in both DMD and Koopman theory. These connections will be explored in more depth below. Because $\mathbf{A}$ is an approximate representation of the Koopman operator restricted to a finite-dimensional subspace of linear measurements, we are often interested in the eigenvectors $\mathbf{\Phi}$ and eigenvalues $\mathbf{\Lambda}$ of $\mathbf{A}$:

$$\mathbf{A}\mathbf{\Phi} = \mathbf{\Phi}\mathbf{\Lambda}. \tag{7.17}$$

However, for a high-dimensional state vector $\mathbf{x} \in \mathbb{R}^n$, the matrix $\mathbf{A}$ has n^2 elements, and representing this operator, let alone computing its spectral decomposition, may be intractable. Instead, the DMD algorithm leverages dimensionality reduction to compute the dominant eigenvalues and eigenvectors of $\mathbf{A}$ without requiring any explicit computations using $\mathbf{A}$ directly. In particular, the pseudo-inverse $\mathbf{X}^\dagger$ in (7.16) is computed via the singular value decomposition of the matrix $\mathbf{X}$. Since this matrix typically has far fewer columns than rows, i.e., $m \ll n$, there are at most m non-zero singular values and corresponding singular vectors, and hence the matrix $\mathbf{A}$ will have at most rank m. Instead of computing $\mathbf{A}$ directly, we compute the projection of $\mathbf{A}$ onto these leading singular vectors, resulting in a small matrix $\tilde{\mathbf{A}}$ of size at most $m \times m$. A major contribution of Schmid [635] was a procedure to approximate the high-dimensional DMD modes (eigenvectors of $\mathbf{A}$) from the reduced matrix $\tilde{\mathbf{A}}$ and the data matrix $\mathbf{X}$ without ever resorting to computations on the full $\mathbf{A}$. Tu et al. [727] later proved that these approximate modes are in fact exact eigenvectors of the full $\mathbf{A}$ matrix under certain conditions. Thus, the exact DMD algorithm of Tu et al. [727] is given by the following steps:

Step 1. Compute the singular value decomposition of $\mathbf{X}$ (see Chapter 1):

$$\mathbf{X} \approx \tilde{\mathbf{U}}\tilde{\mathbf{\Sigma}}\tilde{\mathbf{V}}^*, \tag{7.18}$$

where $\tilde{\mathbf{U}} \in \mathbb{C}^{n\times r}$, $\tilde{\mathbf{\Sigma}} \in \mathbb{C}^{r\times r}$, and $\tilde{\mathbf{V}} \in \mathbb{C}^{m\times r}$, and $r \leq m$ denotes either the exact or the approximate rank of the data matrix $\mathbf{X}$. In practice, choosing the approximate rank r is one of the most important and subjective steps in DMD, and in dimensionality reduction in general. We advocate the principled hard-thresholding algorithm of Gavish and Donoho [267] to determine r from noisy data (see Section 1.7). The columns of the matrix $\tilde{\mathbf{U}}$ are also known as POD modes, and they satisfy $\tilde{\mathbf{U}}^*\tilde{\mathbf{U}} = \mathbf{I}$. Similarly, the columns of $\tilde{\mathbf{V}}$ are orthonormal and satisfy $\tilde{\mathbf{V}}^*\tilde{\mathbf{V}} = \mathbf{I}$.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Step 2. According to (7.16), the full matrix $\mathbf{A}$ may be obtained by computing the pseudo-inverse of $\mathbf{X}$:

$$\mathbf{A} = \mathbf{X}'\tilde{\mathbf{V}}\tilde{\mathbf{\Sigma}}^{-1}\tilde{\mathbf{U}}^{*}. \tag{7.19}$$

However, we are only interested in the leading r eigenvalues and eigenvectors of $\mathbf{A}$, and we may thus project $\mathbf{A}$ onto the POD modes in $\mathbf{U}$:

$$\tilde{\mathbf{A}} = \tilde{\mathbf{U}}^{*}\mathbf{A}\tilde{\mathbf{U}} = \tilde{\mathbf{U}}^{*}\mathbf{X}'\tilde{\mathbf{V}}\tilde{\mathbf{\Sigma}}^{-1}. \tag{7.20}$$

The key observation here is that the reduced matrix $\tilde{\mathbf{A}}$ has the same nonzero eigenvalues as the full matrix $\mathbf{A}$. Thus, we need only compute the reduced $\tilde{\mathbf{A}}$ directly, without ever working with the high-dimensional $\mathbf{A}$ matrix. The reduced-order matrix $\tilde{\mathbf{A}}$ defines a linear model for the dynamics of the vector of POD coefficients $\tilde{\mathbf{x}}$:

$$\tilde{\mathbf{x}}_{k+1} = \tilde{\mathbf{A}}\tilde{\mathbf{x}}_k. \tag{7.21}$$

Note that the matrix $\tilde{\mathbf{U}}$ provides a map to reconstruct the full state $\mathbf{x}$ from the reduced state $\tilde{\mathbf{x}}$, i.e., $\mathbf{x} = \tilde{\mathbf{U}}\tilde{\mathbf{x}}$.

Step 3. The spectral decomposition of $\tilde{\mathbf{A}}$ is computed:

$$\tilde{\mathbf{A}}\mathbf{W} = \mathbf{W}\mathbf{\Lambda}. \tag{7.22}$$

The entries of the diagonal matrix $\mathbf{\Lambda}$ are the DMD eigenvalues, which also correspond to eigenvalues of the full $\mathbf{A}$ matrix. The columns of $\mathbf{W}$ are eigenvectors of $\tilde{\mathbf{A}}$, and provide a coordinate transformation that diagonalizes the matrix. These columns may be thought of as linear combinations of POD mode amplitudes that behave linearly with a single temporal pattern given by λ.

Step 4. The high-dimensional DMD modes $\mathbf{\Phi}$ are reconstructed using the eigenvectors $\mathbf{W}$ of the reduced system and the time-shifted snapshot matrix $\mathbf{X}'$ according to

$$\mathbf{\Phi} = \mathbf{X}'\tilde{\mathbf{V}}\tilde{\mathbf{\Sigma}}^{-1}\mathbf{W}. \tag{7.23}$$

Remarkably, these DMD modes are eigenvectors of the high-dimensional $\mathbf{A}$ matrix corresponding to the eigenvalues in $\mathbf{\Lambda}$, as shown in Tu et al. [727]:

$$\begin{aligned}\mathbf{A}\mathbf{\Phi} &= (\mathbf{X}'\tilde{\mathbf{V}}\tilde{\mathbf{\Sigma}}^{-1}\underbrace{\tilde{\mathbf{U}}^{*})(\mathbf{X}'\tilde{\mathbf{V}}\tilde{\mathbf{\Sigma}}^{-1}}_{\tilde{\mathbf{A}}}\mathbf{W}) \\ &= \mathbf{X}'\tilde{\mathbf{V}}\tilde{\mathbf{\Sigma}}^{-1}\tilde{\mathbf{A}}\mathbf{W} \\ &= \mathbf{X}'\tilde{\mathbf{V}}\tilde{\mathbf{\Sigma}}^{-1}\mathbf{W}\mathbf{\Lambda} \\ &= \mathbf{\Phi}\mathbf{\Lambda}.\end{aligned}$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

In the original paper by Schmid [635], DMD modes are computed using $\mathbf{\Phi} = \tilde{\mathbf{U}}\mathbf{W}$, which are known as *projected modes*; however, these modes are not guaranteed to be exact eigenvectors of $\mathbf{A}$. Because $\mathbf{A}$ is defined as $\mathbf{A} = \mathbf{X}'\mathbf{X}^{\dagger}$, eigenvectors of $\mathbf{A}$ should be in the column space of $\mathbf{X}'$, as in the exact DMD definition, instead of the column space of $\mathbf{X}$ in the original DMD algorithm. In practice, the column spaces of $\mathbf{X}$ and $\mathbf{X}'$ will tend to be nearly identical for dynamical systems with low-rank structure, so that the projected and exact DMD modes often converge.

To find a DMD mode corresponding to a zero eigenvalue, $\lambda = 0$, it is possible to use the exact formulation if $\boldsymbol{\phi} = \mathbf{X}'\tilde{\mathbf{V}}\tilde{\mathbf{\Sigma}}^{-1}\mathbf{w} \neq 0$. However, if this expression is null, then the projected mode $\boldsymbol{\phi} = \tilde{\mathbf{U}}\mathbf{w}$ should be used.

Historical Perspective

In the original formulation, the snapshot matrices $\mathbf{X}$ and $\mathbf{X}'$ were formed with a collection of sequential snapshots, evenly spaced in time:

$$\mathbf{X} = \begin{bmatrix} | & | & & | \\ \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_m \\ | & | & & | \end{bmatrix}, \tag{7.24a}$$

$$\mathbf{X}' = \begin{bmatrix} | & | & & | \\ \mathbf{x}_2 & \mathbf{x}_3 & \cdots & \mathbf{x}_{m+1} \\ | & | & & | \end{bmatrix}. \tag{7.24b}$$

Thus, the matrix $\mathbf{X}$ can be written in terms of iterations of the matrix $\mathbf{A}$ as

$$\mathbf{X} \approx \begin{bmatrix} | & | & & | \\ \mathbf{x}_1 & \mathbf{A}\mathbf{x}_1 & \cdots & \mathbf{A}^{m-1}\mathbf{x}_1 \\ | & | & & | \end{bmatrix}. \tag{7.25}$$

Thus, the columns of the matrix $\mathbf{X}$ belong to a Krylov subspace generated by the propagator $\mathbf{A}$ and the initial condition $\mathbf{x}_1$. In addition, the matrix $\mathbf{X}'$ may be related to $\mathbf{X}$ through the *shift* operator as

$$\mathbf{X}' = \mathbf{X}\mathbf{S}, \tag{7.26}$$

where $\mathbf{S}$ is defined as

$$\mathbf{S} = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 & a_1 \\ 1 & 0 & 0 & \cdots & 0 & a_2 \\ 0 & 1 & 0 & \cdots & 0 & a_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & a_m \end{bmatrix}. \tag{7.27}$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Thus, the first $m-1$ columns of $\mathbf{X}'$ are obtained directly by shifting the corresponding columns of $\mathbf{X}$, and the last column is obtained as a best-fit combination of the m columns of $\mathbf{X}$ that minimizes the residual. In this way, the DMD algorithm resembles an Arnoldi algorithm used to find the dominant eigenvalues and eigenvectors of a matrix $\mathbf{A}$ through iteration. The matrix $\mathbf{S}$ will share eigenvalues with the high-dimensional $\mathbf{A}$ matrix, so that decomposition of $\mathbf{S}$ may be used to obtain dynamic modes and eigenvalues. However, computations based on $\mathbf{S}$ is not as numerically stable as the exact algorithm above.

Spectral Decomposition and DMD Expansion

One of the most important aspects of the DMD is the ability to expand the system state in terms of a data-driven spectral decomposition:

$$\mathbf{x}_k = \sum_{j=1}^{r} \boldsymbol{\phi}_j \lambda_j^{k-1} b_j = \boldsymbol{\Phi}\boldsymbol{\Lambda}^{k-1}\mathbf{b} = \begin{bmatrix} | & & | \\ \boldsymbol{\phi}_1 & \cdots & \boldsymbol{\phi}_r \\ | & & | \end{bmatrix} \begin{bmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_r \end{bmatrix} \begin{bmatrix} b_1 \\ \vdots \\ b_r \end{bmatrix}, \tag{7.28}$$

where $\boldsymbol{\phi}_j$ are DMD modes (eigenvectors of the $\mathbf{A}$ matrix), λ_j are DMD eigenvalues (eigenvalues of the $\mathbf{A}$ matrix), and b_j is the mode amplitude. The DMD expansion above has a direct connection to the Koopman mode decomposition in Section 7.4 (see equation (7.79)). The DMD expansion may be written equivalently as

$$\mathbf{x}_k = \begin{bmatrix} | & & | \\ \boldsymbol{\phi}_1 & \cdots & \boldsymbol{\phi}_r \\ | & & | \end{bmatrix} \begin{bmatrix} b_1 & & \\ & \ddots & \\ & & b_r \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \vdots \\ \lambda_r \end{bmatrix}, \tag{7.29}$$

which makes it possible to express the data matrix $\mathbf{X}$ as

$$\mathbf{X} = \begin{bmatrix} | & & | \\ \boldsymbol{\phi}_1 & \cdots & \boldsymbol{\phi}_r \\ | & & | \end{bmatrix} \begin{bmatrix} b_1 & & \\ & \ddots & \\ & & b_r \end{bmatrix} \begin{bmatrix} \lambda_1 & \cdots & \lambda_1^{m-1} \\ \vdots & \ddots & \vdots \\ \lambda_r & \cdots & \lambda_r^{m-1} \end{bmatrix}. \tag{7.30}$$

The vector $\mathbf{b}$ of mode amplitudes is generally computed as

$$\mathbf{b} = \boldsymbol{\Phi}^{\dagger}\mathbf{x}_1, \tag{7.31}$$

using the first snapshot to determine the mixture of DMD mode amplitudes. However, computing the mode amplitudes is generally quite expensive, even

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

using the straightforward definition in (7.31). Instead, it is possible to compute these amplitudes using POD projected data:

$$\mathbf{x}_1 = \mathbf{\Phi}\mathbf{b} \tag{7.32a}$$
$$\implies \tilde{\mathbf{U}}\tilde{\mathbf{x}}_1 = \mathbf{X}'\tilde{\mathbf{V}}\tilde{\mathbf{\Sigma}}^{-1}\mathbf{W}\mathbf{b} \tag{7.32b}$$
$$\implies \tilde{\mathbf{x}}_1 = \tilde{\mathbf{U}}^*\mathbf{X}'\tilde{\mathbf{V}}\tilde{\mathbf{\Sigma}}^{-1}\mathbf{W}\mathbf{b} \tag{7.32c}$$
$$\implies \tilde{\mathbf{x}}_1 = \tilde{\mathbf{A}}\mathbf{W}\mathbf{b} \tag{7.32d}$$
$$\implies \tilde{\mathbf{x}}_1 = \mathbf{W}\mathbf{\Lambda}\mathbf{b} \tag{7.32e}$$
$$\implies \mathbf{b} = (\mathbf{W}\mathbf{\Lambda})^{-1}\tilde{\mathbf{x}}_1. \tag{7.32f}$$

The matrices $\mathbf{W}$ and $\mathbf{\Lambda}$ are both size $r \times r$, as opposed to the large $\mathbf{\Phi}$ matrix, which is $n \times r$. Alternative approaches to compute $\mathbf{b}$ [27, 173, 356] will be discussed in the next subsection.

The spectral expansion above may also be written in continuous time by introducing the continuous eigenvalues $\omega = \log(\lambda)/\Delta t$:

$$\mathbf{x}(t) = \sum_{j=1}^{r} \boldsymbol{\phi}_j e^{\omega_j t} b_j = \mathbf{\Phi}\exp(\mathbf{\Omega}t)\mathbf{b}, \tag{7.33}$$

where $\mathbf{\Omega}$ is a diagonal matrix containing the continuous-time eigenvalues ω_j. Thus, the data matrix $\mathbf{X}$ may be represented as

$$\mathbf{X} \approx \begin{bmatrix} | & & | \\ \boldsymbol{\phi}_1 & \cdots & \boldsymbol{\phi}_r \\ | & & | \end{bmatrix} \begin{bmatrix} b_1 & & \\ & \ddots & \\ & & b_r \end{bmatrix} \begin{bmatrix} e^{\omega_1 t_1} & \cdots & e^{\omega_1 t_m} \\ \vdots & \ddots & \vdots \\ e^{\omega_r t_1} & \cdots & e^{\omega_r t_m} \end{bmatrix} = \mathbf{\Phi}\,\mathrm{diag}(\mathbf{b})\mathbf{T}(\boldsymbol{\omega}). \tag{7.34}$$

Alternative Optimizations to De-Noise and Robustify DMD

The DMD algorithm is purely data-driven, and is thus equally applicable to experimental and numerical data. When characterizing experimental data with DMD, the effects of sensor noise and stochastic disturbances must be accounted for. Bagheri [37] showed that DMD is particularly sensitive to the effects of noisy data, and it has been shown that significant and systematic biases are introduced to the eigenvalue distribution [36, 195, 221, 321]. Although increased sampling decreases the variance of the eigenvalue distribution, it does not remove the bias [321]. This noise sensitivity has motivated several alternative optimization algorithms for DMD to improve the quality and performance of DMD over the standard optimization in (7.16), which is a least-squares fitting procedure involving the Frobenius norm. These algorithms include the total

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

least-squares DMD [321], forward–backward DMD [195], variable projection [27], and robust principal component analysis [631].

One of the simplest ways to remove the systematic bias of the DMD algorithm is by computing it both forward and backward in time and averaging the equivalent matrices, as proposed by Dawson et al. [195]. Thus the two following approximations are considered:

$$\mathbf{X}' \approx \mathbf{A}_1\mathbf{X} \quad \text{and} \quad \mathbf{X} \approx \mathbf{A}_2\mathbf{X}', \tag{7.35}$$

where $\mathbf{A}_2^{-1} \approx \mathbf{A}_1$ for noise-free data. Thus the matrix $\mathbf{A}_2$ is the inverse, or backward time-step, mapping the snapshots from t_{k+1} to t_k. The forward- and backward-time matrices are then averaged, removing the systematic bias from the measurement noise:

$$\mathbf{A} = \tfrac{1}{2}(\mathbf{A}_1 + \mathbf{A}_2^{-1}), \tag{7.36}$$

where the optimization (7.16) can be used to compute both the forward and backward mappings $\mathbf{A}_1$ and $\mathbf{A}_2$. This optimization can be formulated as

$$\mathbf{A} = \underset{\mathbf{A}}{\operatorname{argmin}} \tfrac{1}{2}(\|\mathbf{X}' - \mathbf{A}\mathbf{X}\|_F + \|\mathbf{X} - \mathbf{A}^{-1}\mathbf{X}'\|_F), \tag{7.37}$$

which is highly nonlinear and non-convex due to the inverse $\mathbf{A}^{-1}$. An improved optimization framework was developed by Azencot et al. [32], which proposes

$$\mathbf{A} = \underset{\mathbf{A}_1,\mathbf{A}_2}{\operatorname{argmin}} \tfrac{1}{2}(\|\mathbf{X}'-\mathbf{A}_1\mathbf{X}\|_F+\|\mathbf{X}-\mathbf{A}_2\mathbf{X}'\|_F) \quad \text{s.t.} \quad \mathbf{A}_1\mathbf{A}_2 = \mathbf{I},\ \mathbf{A}_2\mathbf{A}_1 = \mathbf{I} \tag{7.38}$$

to circumvent some of the difficulties of the optimization in (7.37).

Hemati et al. [321] formulate another DMD algorithm, replacing the original least-squares regression with a total least-squares regression to account for the possibility of noisy measurements and disturbances to the state. This work also provides an excellent discussion on the sources of noise and a comparison of various de-noising algorithms. The subspace DMD algorithm of Takeishi et al. [693] compensates for measurement noise by computing an orthogonal projection of future snapshots onto the space of previous snapshots and then constructing a linear model. Extensions that combine DMD with Bayesian approaches have also been developed [691].

Good approximations for the mode amplitudes $\mathbf{b}$ in (7.34) have also proven to be difficult to achieve, with and without noise. Jovanović et al. [356] developed the first algorithm to improve the estimate of the modal amplitudes by promoting sparsity. In this case, the underlying optimization algorithm is framed around improving the approximation (7.34) using the formulation

$$\underset{\mathbf{b}}{\operatorname{argmin}}(\|\mathbf{X} - \boldsymbol{\Phi}\,\mathrm{diag}(\mathbf{b})\mathbf{T}(\boldsymbol{\omega})\|_F + \gamma\|\mathbf{b}\|_1), \tag{7.39}$$

where $\|\cdot\|_1$ denotes the ℓ_1-norm penalization which promotes sparsity of $\mathbf{b}$.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

The subspace DMD algorithm of Takeishi et al. [693] also compensates for measurement noise by computing an orthogonal projection of future snapshots onto the space of previous snapshots and then constructing a linear model. A Bayesian DMD approach has also been developed [691]. More recently, Askham and Kutz [27] introduced the *optimized DMD* algorithm, which uses a variable projection method for nonlinear least-squares to compute the DMD for unevenly timed samples, significantly mitigating the bias due to noise. The optimized DMD algorithm solves the exponential fitting problem directly:

$$\underset{\boldsymbol{\omega},\boldsymbol{\Phi}_\mathbf{b}}{\operatorname{argmin}} \|\mathbf{X} - \boldsymbol{\Phi}_\mathbf{b}\mathbf{T}(\boldsymbol{\omega})\|_F. \tag{7.40}$$

This has been shown to suppress bias, although one must solve a nonlinear optimization problem. However, using statistical bagging methods, optimized DMD can be stabilized and the *boosted optimized DMD* (BOP-DMD) method can not only improve performance of the decomposition, but also provide uncertainty estimates for the DMD eigenvalues and DMD eigenmodes [622].

DMD is able to accurately identify an approximate linear model for dynamics that are linear, periodic, or quasi-periodic. However, DMD is unable to capture a linear dynamical system model with essential nonlinear features, such as multiple fixed points, unstable periodic orbits, or chaos [127]. As an example, DMD will fail to yield a reasonable linear model for the chaotic Lorenz system, and it also will not capture important features of the linear portion of the Lorenz model. The sparse identification of nonlinear dynamics (SINDy) [132], discussed in Section 7.3, is a related algorithm that identifies fully nonlinear dynamical systems models from data. However, SINDy often faces scaling issues for high-dimensional systems that do not admit a low-dimensional subspace or submanifold. In this case, the recent linear and nonlinear disambiguation optimization (LANDO) algorithm [34] leverages kernel methods to identify an implicit model for the full nonlinear dynamics, where it is then possible to extract a low-rank DMD approximation for the linear portion linearized about some specified operating condition. In this way, the LANDO algorithm robustly extracts the linear DMD dynamics even from strongly nonlinear systems. This work is part of a much larger effort to use kernels for learning dynamical systems and Koopman representations [34, 143, 191, 257, 391, 393, 758].

Example and Code

Code 7.3 provides a basic DMD implementation. This DMD code is demonstrated in Fig. 7.3 for the fluid flow past a circular cylinder at Reynolds number 100, based on the cylinder diameter. The DMD eigenvalues are shown in Fig. 7.4 for clean data and for data corrupted with Gaussian white noise. The two-dimensional Navier–Stokes equations are simulated using the immersed

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

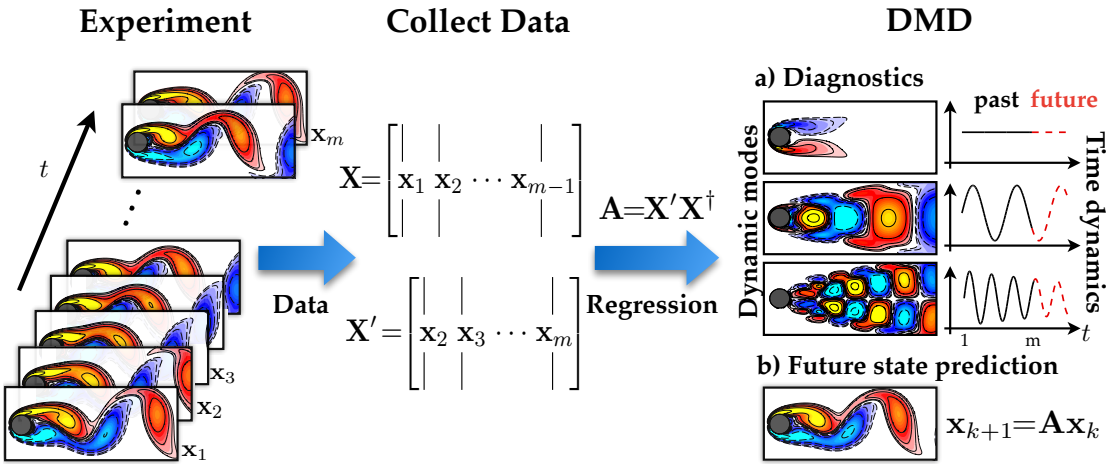


Figure 7.3: Overview of DMD illustrated on the fluid flow past a circular cylinder at Reynolds number 100. Reproduced from [422].

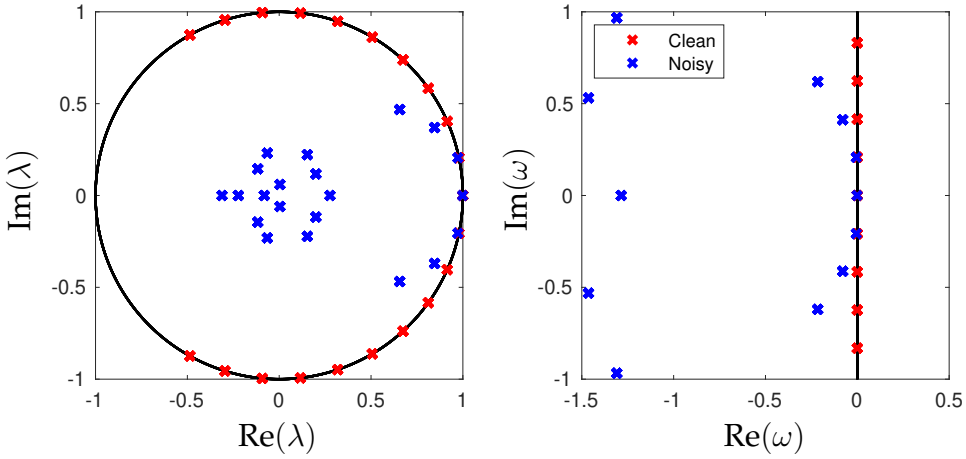


Figure 7.4: DMD eigenvalues for the fluid flow past a circular cylinder at Reynolds number 100. When trained on clean data without noise, the discrete-time DMD eigenvalues are on the unit circle, and the continuous-time DMD eigenvalues are on the imaginary axis, indicating purely oscillatory dynamics. However, when noise is added to the training data, the eigenvalues exhibit spurious damping, as predicted by Bagheri [37].

boundary projection method (IBPM) solver[1] based on the fast multi-domain method of Taira and Colonius [181, 688]. The data required for this example may be downloaded without running the IBPM code at `http://DMDbook.com`.

[1]The IBPM code is publicly available at: `https://github.com/cwrowley/ibpm`.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Code 7.3: [MATLAB] DMD implementation.

```
function [Phi, Lambda, b] = DMD(X,Xprime,r)

[U,Sigma,V] = svd(X,'econ');       % Step 1
Ur = U(:,1:r);
Sigmar = Sigma(1:r,1:r);
Vr = V(:,1:r);

Atilde = Ur'*Xprime*Vr/Sigmar;     % Step 2
[W,Lambda] = eig(Atilde);          % Step 3

Phi = Xprime*(Vr/Sigmar)*W;        % Step 4
alpha1 = Sigmar*Vr(1,:)';
b = (W*Lambda)\alpha1;
```

Code 7.3: [Python] DMD implementation.

```
def DMD(X,Xprime,r):
    U,Sigma,VT = np.linalg.svd(X,full_matrices=0) # Step 1
    Ur = U[:,:r]
    Sigmar = np.diag(Sigma[:r])
    VTr = VT[:r,:]
    Atilde = np.linalg.solve(Sigmar.T,(Ur.T @ Xprime @ VTr.T
        ).T).T # Step 2
    Lambda, W = np.linalg.eig(Atilde) # Step 3
    Lambda = np.diag(Lambda)
    # Step 4
    Phi = Xprime @ np.linalg.solve(Sigmar.T,VTr).T @ W
    alpha1 = Sigmar @ VTr[:,0]
    b = np.linalg.solve(W @ Lambda,alpha1)
    return Phi, Lambda, b
```

With this data, it is simple to compute the dynamic mode decomposition. In MATLAB, the following code is used:

```
% VORTALL contains flow fields reshaped into column vectors
X = VORTALL;
[Phi, Lambda, b] = DMD(X(:,1:end-1),X(:,2:end),21);
```

In Python, the following code is used:

```
vortall_mat = io.loadmat(os.path.join('..','DATA','VORTALL.
    mat'))
X = vortall_mat['VORTALL']
Phi, Lambda, b = DMD(X[:,:-1],X[:,1:],21)
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Extensions, Applications, and Limitations

One of the major advantages of dynamic mode decomposition is its simple framing in terms of linear regression. DMD does not require knowledge of governing equations. For this reason, DMD has been rapidly extended to include several methodological innovations and has been widely applied beyond fluid dynamics [422], where it originated. Here, we present a number of the leading algorithmic extensions and promising domain applications, and we also present current limitations of the DMD theory that must be addressed in future research.

Methodological Extensions

Compression and Randomized Linear Algebra. DMD was originally designed for high-dimensional data sets in fluid dynamics, such as a fluid velocity or vorticity field, which may contain millions of degrees of freedom. However, the fact that DMD often uncovers low-dimensional structure in these high-dimensional data implies that there may be more efficient measurement and computational strategies based on principles of *sparsity* (see Chapter 3). There have been several independent and highly successful extensions and modifications of DMD to exploit low-rank structure and sparsity.

In 2014, Jovanović et al. [356] used sparsity-promoting optimization to identify the fewest DMD modes required to describe a data set, essentially identifying a few dominant DMD mode amplitudes in $\mathbf{b}$. The alternative approach, of testing and comparing all subsets of DMD modes, represents a computationally intractable brute-force search.

Another line of work is based on the fact that DMD modes generally admit a sparse representation in Fourier or wavelet bases. Moreover, the time dynamics of each mode are simple pure tone harmonics, which are the definition of sparse in a Fourier basis. This sparsity has facilitated several efficient measurement strategies that reduce the number of measurements required in time [726] and space [134, 232, 303], based on compressed sensing. This has the broad potential to enable high-resolution characterization of systems from under-resolved measurements.

Related to the use of compressed sensing, randomized linear algebra has recently been used to accelerate DMD computations when full-state data is available. Instead of collecting subsampled measurements and using compressed sensing to infer high-dimensional structures, randomized methods start with full data and then randomly project into a lower-dimensional subspace, where computations may be performed more efficiently. Bistrian and Navon [93] have successfully accelerated DMD using a randomized singular value decomposition, and Erichson et al. [234] demonstrate how all of the expensive DMD computations may be performed in a projected subspace.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Finally, libraries of DMD modes have also been used to identify dynamical regimes [411], based on the sparse representation for classification [762] (see Section 3.6), which was used earlier to identify dynamical regimes using libraries of POD modes [112, 136].

Inputs and Control. A major strength of DMD is the ability to describe complex and high-dimensional dynamical systems in terms of a small number of dominant modes, which represent spatio-temporal coherent structures. Reducing the dimensionality of the system from n (often millions or billions) to r (tens or hundreds) enables faster and lower-latency prediction and estimation. Lower-latency predictions generally translate directly into controllers with higher performance and robustness. Thus, compact and efficient representations of complex systems such as fluid flows have been long sought, resulting in the field of reduced-order modeling. However, the original DMD algorithm was designed to characterize naturally evolving systems, without accounting for the effect of actuation and control.

Shortly after the original DMD algorithm, Proctor et al. [570] extended the algorithm to disambiguate between the natural unforced dynamics and the effect of actuation. This essentially amounts to a generalized evolution equation

$$\mathbf{x}_{k+1} \approx \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k, \tag{7.41}$$

which results in another linear regression problem (see Section 10.2).

The original motivation for DMD with control (DMDc) was the use of DMD to characterize epidemiological systems (e.g., malaria spreading across a continent), where it is not possible to stop intervention efforts, such as vaccinations and bed nets, in order to characterize the unforced dynamics [569].

Since the original DMDc algorithm, the compressed sensing DMD and DMDc algorithms have been combined, resulting in a new framework for compressive system identification [41]. In this framework, it is possible to collect undersampled measurements of an actuated system and identify an accurate and efficient low-order model, related to DMD and the eigensystem realization algorithm (ERA; see Section 9.3) [358].

DMDc models, based on linear and nonlinear measurements of the system, have recently been used with model predictive control (MPC) for enhanced control of nonlinear systems by Korda and Mezić [404]. Model predictive control using DMDc models was subsequently used as a benchmark comparison for MPC based on fully nonlinear models in the work of Kaiser et al. [366], and the DMDc models performed surprisingly well, even for strongly nonlinear systems.

Nonlinear Measurements. Much of the excitement around DMD is due to the strong connection to nonlinear dynamics via the Koopman operator [611].

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Indeed, DMD is able to accurately characterize periodic and quasi-periodic behavior, even in nonlinear systems, as long as a sufficient amount of data is collected. However, the basic DMD algorithm uses linear measurements of the system, which are generally not rich enough to characterize truly nonlinear phenomena, such as transients, intermittent phenomena, or broadband frequency cross-talk. In Williams et al. [757], DMD measurements were augmented to include nonlinear measurements of the system, enriching the basis used to represent the Koopman operator. The so-called *extended DMD* (eDMD) algorithm then seeks to obtain a linear model $\mathbf{A_Y}$ advancing nonlinear measurements $\mathbf{y} = \mathbf{g}(\mathbf{x})$:

$$\mathbf{y}_{k+1} \approx \mathbf{A_Y}\mathbf{y}_k. \tag{7.42}$$

For high-dimensional systems, this augmented state $\mathbf{y}$ may be intractably large, motivating the use of kernel methods to approximate the evolution operator $\mathbf{A_Y}$ [758]. This kernel DMD has since been extended to include dictionary learning techniques [440].

It has recently been shown that eDMD is equivalent to the variational approach of conformation dynamics (VAC) [528, 534, 535], first derived by Noé and Nüske in 2013 to simulate molecular dynamics with a broad separation of timescales. Further connections between eDMD and VAC and between DMD and the time-lagged independent component analysis (TICA) are explored in a recent review [392]. A key contribution of VAC is a variational score enabling the objective assessment of Koopman models via cross-validation.

Following the extended DMD, it was shown that there are relatively restrictive conditions for obtaining a linear regression model that includes the original state of the system [127]. For nonlinear systems with multiple fixed points, periodic orbits, and other attracting structures, there is no finite-dimensional linear system including the state $\mathbf{x}$ that is topologically conjugate to the nonlinear system. Instead, it is important to identify Koopman-invariant subspaces, spanned by eigenfunctions of the Koopman operator; in general, it will not be possible to directly write the state $\mathbf{x}$ in the span of these eigenvectors, although it may be possible to identify $\mathbf{x}$ through a unique inverse. A practical algorithm for identifying eigenfunctions is provided by Kaiser et al. [365].

Multi-Resolution. DMD is often applied to complex, high-dimensional dynamical systems, such as fluid turbulence or epidemiological systems, that exhibit multi-scale dynamics in both space and time. Many multi-scale systems exhibit transient or intermittent phenomena, such as the El Niño observed in global climate data. These transient dynamics are not captured accurately by DMD, which seeks spatio-temporal modes that are globally coherent across the entire time series of data. To address this challenge, the multi-resolution DMD (mrDMD) algorithm was introduced [423], which effectively decomposes

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

the dynamics into different timescales, isolating transient and intermittent patterns. Multi-resolution DMD modes were recently shown to be advantageous for sparse sensor placement by Manohar et al. [483].

Delay Measurements. Although DMD was developed for high-dimensional data where it is assumed that one has access to the full state of a system, it is often desirable to characterize spatio-temporal coherent structures for systems with incomplete measurements. As an extreme example, consider a single measurement that oscillates as a sinusoid, $x(t) = \sin(\omega t)$. Although this would appear to be a perfect candidate for DMD, the algorithm incorrectly identifies a real eigenvalue because the data does not have sufficient rank to extract a complex conjugate pair of eigenvalues $\pm i\omega$. This paradox was first explored by Tu et al. [727], where it was discovered that a solution is to stack delayed measurements into a larger matrix to augment the rank of the data matrix and extract phase information. Delay coordinates have been used effectively to extract coherent patterns in neural recordings [123]. The connections between delay DMD and Koopman [25, 126, 190] will be discussed more in Section 7.5.

Streaming and Parallelized Codes. Because of the computational burden of computing the DMD on high-resolution data, several advances have been made to accelerate DMD in streaming applications and with parallelized algorithms. DMD is often used in a streaming setting, where a moving window of snapshots are processed continuously, resulting in redundant computations when new data becomes available. Several algorithms exist for streaming DMD, based on the incremental SVD [322], a streaming method of snapshots SVD [559], and rank-one updates to the DMD matrix [772]. The DMD algorithm is also readily parallelized, as it is based on the SVD. Several parallelized codes are available, based on the QR [623] and SVD [234, 233, 236].

Tensor Formulations. Most data used to compute DMD has additional spatial structure that is discarded when the data is reshaped into column vectors. The tensor DMD extension of Klus et al. [390] performs DMD on a tensorial, rather than vectorized, representation of the data, retaining this additional structure. In addition, this approach reduces the memory requirements and computational complexity for large-scale systems. Extensions to this approach have been introduced based on reproducing kernel Hilbert spaces [257] and the extended DMD [533], and additional connections have recently been made between the Koopman mode decomposition and tensor factorizations [594]. Tensor approaches to related methods, such as the sparse identification of nonlinear dynamics [132], have also been developed recently [273].

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Resolvent Analysis. DMD and Koopman operator theory have also been connected to the resolvent analysis from fluid mechanics [324, 655]. Resolvent analysis seeks to find the most receptive states of a dynamical system that will be most amplified by forcing, along with the corresponding most responsive forcings [354, 355, 495, 712]. Sharma, Mezić, and McKeon [655] established several important connections between DMD, Koopman theory, and the resolvent operator, including a generalization of DMD to enforce symmetries and traveling wave structures. They also showed that the resolvent modes provide an optimal basis for the Koopman mode decomposition. Typically, resolvent analysis is performed by linearizing the governing equations about a base state, often a turbulent mean flow. However, this approach is *invasive*, requiring a working Navier–Stokes solver. Herrmann et al. [324] have recently developed a purely data-driven resolvent algorithm, based on DMD, that bypasses knowledge of the governing equations. DMD and resolvent analysis are also both closely related to the spectral POD [642, 690, 708], which is related to the classical POD of Lumley and provides time-harmonic modes at a set of discrete frequencies.

Applications

Fluid Dynamics. DMD originated in the fluid dynamics community [635], and has since been applied to a wide range of flow geometries (jets, cavity flow, wakes, channel flow, boundary layers, etc.) to study mixing, acoustics, and combustion, among other phenomena. In the original papers of Schmid [635, 636], both a cavity flow and a jet were considered. In the original paper of Rowley et al. [611], a jet in cross-flow was investigated. It is no surprise that DMD has subsequently been used widely in both cavity flows [56, 57, 466, 635, 647] and jets [68, 637, 638, 649].

DMD has also been applied to wake flows, including to investigate frequency lock-on [725], the wake past a gurney flap [543], the cylinder wake [36], and dynamic stall [223]. Boundary layers have also been extensively studied with DMD [505, 539, 624]. In acoustics, DMD has been used to capture the near-field and far-field acoustics that result from instabilities observed in shear flows [668]. In combustion, DMD has been used to understand the coherent heat release in turbulent swirl flames [507] and to analyze a rocket combustor [341]. DMD has also been used to analyze non-normal growth mechanisms in thermoacoustic interactions in a Rijke tube. DMD has been compared with POD for reacting flows [612]. DMD has also been used to analyze more exotic flows, including a simulated model of a high-speed train [514]. Shock–turbulent boundary layer interaction (STBLI) has also been investigated, and DMD was used to identify a pulsating separation bubble that is accompanied by shockwave motion [297]. DMD has also been used to study self-excited fluctuations in detonation waves [490]. Other problems include identifying hairpin

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

vortices [695], decomposing the flow past a surface-mounted cube [515], modeling shallow-water equations [92], studying nanofluids past a square cylinder [620], fluid–structure interaction [292], and measuring the growth rate of instabilities in annular liquid sheets [220]. A modified *recursive* DMD algorithm was also formulated by Noack et al. [527] to provide an orthogonal basis for empirical Galerkin models in fluids. The use of DMD in fluids fits into a broader effort to leverage machine learning for improved models and controllers [52, 111, 131, 399, 441, 617], especially for turbulence closure modeling [64, 65, 224, 421, 448, 492].

Epidemiology. DMD has recently been applied to investigate epidemiological systems by Proctor and Eckhoff [568]. This is a particularly interpretable application, as modal frequencies often correspond to yearly or seasonal fluctuations. Moreover, the phase of DMD modes gives insight into how disease fronts propagate spatially, potentially informing future intervention efforts. The application of DMD to disease systems also motivated the DMD with control [570], since it is infeasible to stop vaccinations in order to identify the unforced dynamics.

Neuroscience. Complex signals from neural recordings are increasingly high-fidelity and high-dimensional, with advances in hardware pushing the frontiers of data collection. DMD has the potential to transform the analysis of such neural recordings, as evidenced in a recent study that identified dynamically relevant features in electrocorticography (ECOG) data of sleeping patients [123]. Since then, several works have applied DMD to neural recordings or suggested possible implementation in hardware [5, 117, 704].

Video Processing. Separating foreground and background objects in video is a common task in surveillance applications. Real-time separation is a challenge that is only exacerbated by ever-increasing video resolutions. DMD provides a flexible platform for video separation, as the background may be approximated by a DMD mode with zero eigenvalue [232, 298, 559].

Other Applications. DMD has been applied to an increasingly diverse array of problems, including robotics [78], finance [480], and plasma physics [697]. It is expected that this trend will increase.

Challenges

Traveling Waves. DMD is based on the SVD of a data matrix $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^*$ whose columns are spatial measurements evolving in time. In this case, the SVD is a space–time separation of variables into spatial modes, given by the

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

columns of $\mathbf{U}$, and time dynamics, given by the columns of $\mathbf{V}$. As in POD, DMD thus has limitations for problems that exhibit traveling waves, where separation of variables is known to fail.

Transients. Many systems of interest are characterized by transients and intermittent phenomena. Several methods have been proposed to identify these events, such as the multi-resolution DMD and the use of delay coordinates. However, it is still necessary to formalize the choice of relevant timescales and the window size to compute DMD.

Continuous Spectrum. Related to the above, many systems are characterized by broadband frequency content, as opposed to a few distinct and discrete frequencies. This broadband frequency content is also known as a *continuous spectrum*, where every frequency in a continuous range is observed. For example, the simple pendulum exhibits a continuous spectrum, as the system has a natural frequency for small deflections, and this frequency continuously deforms and slows as energy is added to the pendulum. Other systems include nonlinear optics and broadband turbulence. These systems pose a serious challenge for DMD, as they result in a large number of modes, even though the dynamics are likely generated by the nonlinear interactions of a few dominant modes.

Several data-driven approaches have been recently proposed to handle systems with continuous spectra. Applying DMD to a vector of delayed measurements of a system, the so-called HAVOK analysis in Section 7.5, has been shown to approximate the dynamics of chaotic systems, such as the Lorenz system, which exhibits a continuous spectrum. In addition, Lusch et al. [465] showed that it is possible to design a deep learning architecture with an auxiliary network to parameterize the continuous frequency.

Strong Nonlinearity and Choice of Measurements. Although significant progress has been made connecting DMD to nonlinear systems [758], choosing nonlinear measurements to augment the DMD regression is still not an exact science. Identifying measurement subspaces that remain closed under the Koopman operator is an ongoing challenge [127]. Recent progress in deep learning has the potential to enable the representation of extremely complex eigenfunctions from data [465, 485, 540, 692, 747, 766].

7.3 Sparse Identification of Nonlinear Dynamics (SINDy)

Discovering dynamical systems models from data is a central challenge in mathematical physics, with a rich history going back at least as far as the time of

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Kepler and Newton and the discovery of the laws of planetary motion. Historically, this process relied on a combination of high-quality measurements and expert intuition. With vast quantities of data and increasing computational power, the *automated* discovery of governing equations and dynamical systems is a new and exciting scientific paradigm.

Typically, either the form of a candidate model is constrained via prior knowledge of the governing equations, as in Galerkin projection [44, 160, 161, 524, 526, 610, 633, 744] (see Chapter 13), or a handful of heuristic models are tested and parameters are optimized to fit data. Alternatively, best-fit linear models may be obtained using DMD or ERA. Simultaneously identifying the nonlinear structure and parameters of a model from data is considerably more challenging, as there are combinatorially many possible model structures.

The sparse identification of nonlinear dynamics (SINDy) algorithm [132] bypasses the intractable combinatorial search through all possible model structures, leveraging the fact that many dynamical systems

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{x} = \mathbf{f}(\mathbf{x}) \tag{7.43}$$

have dynamics $\mathbf{f}$ with only a few active terms in the space of possible right-hand side functions; for example, the Lorenz equations in (7.2) only have a few linear and quadratic interaction terms per equation.

We then seek to approximate $\mathbf{f}$ by a generalized linear model

$$\mathbf{f}(\mathbf{x}) \approx \sum_{k=1}^{p} \theta_k(\mathbf{x})\xi_k = \boldsymbol{\Theta}(\mathbf{x})\boldsymbol{\xi}, \tag{7.44}$$

with the fewest non-zero terms in $\boldsymbol{\xi}$ as possible. It is then possible to solve for the relevant terms that are active in the dynamics using sparse regression [315, 348, 702, 777] that penalizes the number of terms in the dynamics and scales well to large problems.

First, time-series data is collected from (7.43) and formed into a data matrix:

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}(t_1) & \mathbf{x}(t_2) & \cdots & \mathbf{x}(t_m) \end{bmatrix}^T. \tag{7.45}$$

A similar matrix of derivatives is formed:

$$\dot{\mathbf{X}} = \begin{bmatrix} \dot{\mathbf{x}}(t_1) & \dot{\mathbf{x}}(t_2) & \cdots & \dot{\mathbf{x}}(t_m) \end{bmatrix}^T. \tag{7.46}$$

In practice, this may be computed directly from the data in $\mathbf{X}$; for noisy data, the total-variation regularized derivative tends to provide numerically robust derivatives [169]. Alternatively, it is possible to formulate the SINDy algorithm for discrete-time systems $\mathbf{x}_{k+1} = \mathbf{F}(\mathbf{x}_k)$, as in the DMD algorithm, and avoid derivatives entirely.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

A library of candidate nonlinear functions $\boldsymbol{\Theta}(\mathbf{X})$ may be constructed from the data in $\mathbf{X}$:

$$\boldsymbol{\Theta}(\mathbf{X}) = \begin{bmatrix} \mathbf{1} & \mathbf{X} & \mathbf{X}^2 & \cdots & \mathbf{X}^d & \cdots & \sin(\mathbf{X}) & \cdots \end{bmatrix}. \tag{7.47}$$

Here, the matrix $\mathbf{X}^d$ denotes a matrix with column vectors given by all possible time series of dth-degree polynomials in the state $\mathbf{x}$. In general, this library of candidate functions is only limited by one's imagination.

The dynamical system in (7.43) may now be represented in terms of the data matrices in (7.46) and (7.47) as

$$\dot{\mathbf{X}} = \boldsymbol{\Theta}(\mathbf{X})\boldsymbol{\Xi}. \tag{7.48}$$

Each column $\boldsymbol{\xi}_k$ in $\boldsymbol{\Xi}$ is a vector of coefficients determining the active terms in the kth row in (7.43). A parsimonious model will provide an accurate model fit in (7.48) with as few terms as possible in $\boldsymbol{\Xi}$. Such a model may be identified using a convex ℓ_1-regularized sparse regression:

$$\boldsymbol{\xi}_k = \underset{\boldsymbol{\xi}'_k}{\operatorname{argmin}} \|\dot{\mathbf{X}}_k - \boldsymbol{\Theta}(\mathbf{X})\boldsymbol{\xi}'_k\|_2 + \lambda\|\boldsymbol{\xi}'_k\|_1. \tag{7.49}$$

Here, $\dot{\mathbf{X}}_k$ is the kth column of $\dot{\mathbf{X}}$, and λ is a sparsity-promoting knob. Sparse regression, such as the LASSO [702] or the sequential thresholded least-squares (STLS) algorithm used in SINDy [132], improves the numerical robustness of this identification for noisy over-determined problems, in contrast to earlier methods [743] that used compressed sensing [53, 151, 153, 156, 157, 204, 717]. We advocate the STLS (Code 7.4) to select active terms.

Code 7.4: [MATLAB] Sequentially thresholded least-squares.

```
function Xi = sparsifyDynamics(Theta,dXdt,lambda,n)
% Compute Sparse regression: sequential least squares
Xi = Theta\dXdt;  % Initial guess: Least-squares

% Lambda is our sparsification knob.
for k=1:10
    smallinds = (abs(Xi)<lambda);   % Find small coefficients
    Xi(smallinds)=0;                % and threshold
    for ind = 1:n                   % n is state dimension
        biginds = ~smallinds(:,ind);
% Regress dynamics onto remaining terms to find sparse Xi
        Xi(biginds,ind) = Theta(:,biginds)\dXdt(:,ind);
    end
end
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

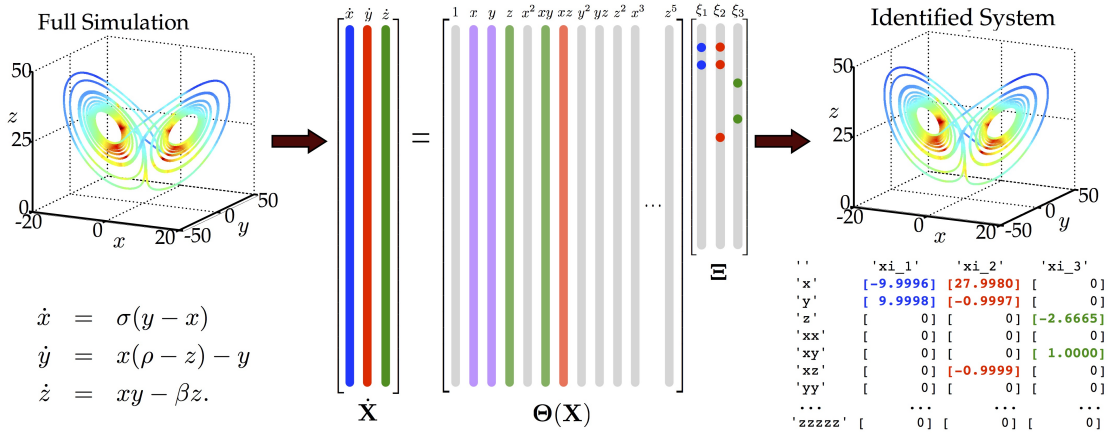


Figure 7.5: Schematic of the sparse identification of nonlinear dynamics (SINDy) algorithm [132]. Parsimonious models are selected from a library of candidate nonlinear terms using sparse regression. This library $\boldsymbol{\Theta}(\mathbf{X})$ may be constructed purely from measurement data. Modified from Brunton et al. [132].

Code 7.4: [Python] Sequentially thresholded least-squares.

```
def sparsifyDynamics(Theta,dXdt,lamb,n):
    # Initial guess: Least-squares
    Xi = np.linalg.lstsq(Theta,dXdt,rcond=None)[0]

    for k in range(10):
        smallinds = np.abs(Xi) < lamb # Find small coeffs.
        Xi[smallinds] = 0       # and threshold
        for ind in range(n):    # n is state dimension
            biginds = smallinds[:,ind] == 0
            # Regress onto remaining terms to find sparse Xi
            Xi[biginds,ind] = np.linalg.lstsq(Theta[:,
                biginds],dXdt[:,ind],rcond=None)[0]

    return Xi
```

The sparse vectors $\boldsymbol{\xi}_k$ may be synthesized into a dynamical system:

$$\dot{x}_k = \boldsymbol{\Theta}(\mathbf{x})\boldsymbol{\xi}_k. \tag{7.50}$$

Note that $\dot{x}_k$ is the kth element of $\dot{\mathbf{x}}$ and $\boldsymbol{\Theta}(\mathbf{x})$ is a row vector of symbolic functions of $\mathbf{x}$, as opposed to the data matrix $\boldsymbol{\Theta}(\mathbf{X})$. Figure 7.5 shows how SINDy may be used to discover the Lorenz equations from data. Code 7.5 performs the SINDy regression for the Lorenz system based on the data generated in Code 7.2.

Code 7.5: [MATLAB] SINDy regression to identify the Lorenz system from

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

data.

```
%% Compute derivatives by evaluating lorenz on trajectory x
for i=1:length(x)
    dx(i,:) = lorenz(0,x(i,:),Beta);
end

%% Build library and compute sparse regression
Theta = poolData(x,n,3);  % up to third order polynomials
lambda = 0.025;       % lambda is our sparsification knob.
Xi = sparsifyDynamics(Theta,dx,lambda,n)
```

Code 7.5: [Python] SINDy regression to identify the Lorenz system from data.

```
## Compute Derivative
dx = np.zeros_like(x)
for j in range(len(t)):
    dx[j,:] = lorenz_deriv(x[j,:],0,sigma,beta,rho)

Theta = poolData(x,n,3) # Up to third order polynomials
lamb = 0.025 # sparsification knob lambda
Xi = sparsifyDynamics(Theta,dx,lamb,n)
```

This code also relies on a function **poolData** that generates the library Θ. In this case, polynomials up to third order are used. This code is available online. For more in-depth applications, we strongly recommend using the open-source Python software package PySINDy [199] at `https://github.com/dynamicslab/pysindy`.

The output of the SINDy algorithm is a sparse matrix of coefficients Ξ:

```
''        'xdot'        'ydot'        'zdot'
 '1'       [       0]    [       0]    [       0]
 'x'       [-10.0000]    [28.0000]    [       0]
 'y'       [ 10.0000]    [-1.0000]    [       0]
 'z'       [       0]    [       0]    [-2.6667]
 'xx'      [       0]    [       0]    [       0]
 'xy'      [       0]    [       0]    [ 1.0000]
 'xz'      [       0]    [-1.0000]    [       0]
 'yy'      [       0]    [       0]    [       0]
 'yz'      [       0]    [       0]    [       0]
 'zz'      [       0]    [       0]    [       0]
 'xxx'     [       0]    [       0]    [       0]
 'xxy'     [       0]    [       0]    [       0]
 'xxz'     [       0]    [       0]    [       0]
 'xyy'     [       0]    [       0]    [       0]
 'xyz'     [       0]    [       0]    [       0]
 'xzz'     [       0]    [       0]    [       0]
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

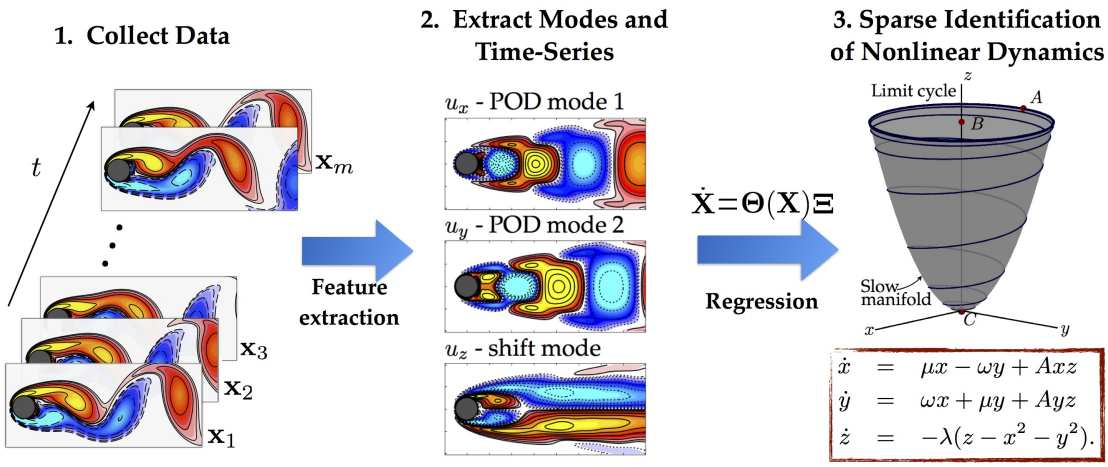


Figure 7.6: Schematic overview of nonlinear model identification from high-dimensional data using the sparse identification of nonlinear dynamics (SINDy) [132]. This procedure is modular, so that different techniques can be used for the feature extraction and regression steps. In this example of flow past a cylinder, SINDy discovers the model of Noack et al. [524]. Modified from Brunton et al. [132].

```
    'yyy'    [    0]    [    0]    [    0]
    'yyz'    [    0]    [    0]    [    0]
    'yzz'    [    0]    [    0]    [    0]
    'zzz'    [    0]    [    0]    [    0]
```

The result of the SINDy regression is a parsimonious model that includes only the most important terms required to explain the observed behavior. The sparse regression procedure used to identify the most parsimonious nonlinear model is a convex procedure. The alternative approach, which involves regression onto every possible sparse nonlinear structure, constitutes an intractable brute-force search through the combinatorially many candidate model forms. SINDy bypasses this combinatorial search with modern convex optimization and machine learning. It is interesting to note that, for discrete-time dynamics, if $\Theta(\mathbf{X})$ consists only of linear terms, and if we remove the sparsity-promoting term by setting $\lambda = 0$, then this algorithm reduces to the dynamic mode decomposition [422, 611, 635, 727]. If a least-squares regression is used, as in DMD, then even a small amount of measurement error or numerical round-off will lead to every term in the library being active in the dynamics, which is non-physical. A major benefit of the SINDy architecture is the ability to identify parsimonious models that contain only the required nonlinear terms, resulting in interpretable models that avoid overfitting.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Applications, Extensions, and Historical Context

The SINDy algorithm has recently been applied to identify high-dimensional dynamical systems, such as fluid flows, based on POD coefficients [132, 455, 456]. Figure 7.6 illustrates the application of SINDy to the flow past a cylinder, where the generalized mean-field model of Noack et al. [524] was discovered from data. Since its introduction, SINDy has been applied to a wide range of systems, including for reduced-order models of fluid dynamics [145, 148, 201, 301, 454, 455, 456] and plasma dynamics [187, 373], turbulence closures [64, 65, 634], nonlinear optics [670], numerical integration schemes [701], discrepancy modeling [198, 363], boundary value problems [656], identifying dynamics on Poincaré maps [104, 105], tensor formulations [273], and systems with stochastic dynamics [96, 146]. The integral formulation of SINDy [627] has also proven to be powerful, enabling the identification of governing equations in a weak form that averages over control volumes; this approach has recently been used to discover a hierarchy of fluid and plasma models [12, 305, 597, 598]. The open-source software package, PySINDy,[2] has been developed in Python to integrate the various extensions of SINDy [199], such as promoting global boundedness [372] by incorporating the Schlegel and Noack constraint [632] in the optimization.

Because SINDy is formulated in terms of linear regression in a nonlinear library, it is highly extensible. The SINDy framework has been recently generalized by Loiseau and Brunton [455] to incorporate known physical constraints and symmetries in the equations by implementing a constrained sequentially thresholded least-squares optimization. In particular, energy-preserving constraints on the quadratic nonlinearities in the Navier–Stokes equations were imposed to identify fluid systems [455], where it is known that these constraints promote stability [44, 160, 472]. This work also showed that polynomial libraries are particularly useful for building models of fluid flows in terms of POD coefficients, yielding interpretable models that are related to classical Galerkin projection [132, 455]. Loiseau et al. [456] also demonstrated the ability of SINDy to identify dynamical systems models of high-dimensional systems, such as fluid flows, from a few physical sensor measurements, such as lift and drag measurements on the cylinder in Fig. 7.6. For actuated systems, SINDy has been generalized to include inputs and control [133], and these models are highly effective for model predictive control [366]. It is also possible to extend the SINDy algorithm to identify dynamics with rational function nonlinearities [364, 478], integral terms [627], and based on highly corrupt and incomplete data [709]. SINDy was also recently extended to incorporate information criteria for objective model selection [479], and to identify models with hidden variables using delay coordinates [126]. Champion et al. [168] combined SINDy with a deep

[2] `https://github.com/dynamicslab/pysindy`.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

autoencoder neural network to simultaneously learn coordinates and dynamics, which will be discussed in Chapter 14. Finally, the SINDy framework was generalized to include partial derivatives, enabling the identification of partial differential equation models [613, 626]. Several of these recent innovations will be explored in more detail below.

More generally, the use of sparsity-promoting methods in dynamics is quite recent [40, 42, 122, 136, 470, 481, 482, 542, 569, 628, 743]. Other techniques for dynamical system discovery include methods to discover equations from time series [186], equation-free modeling [383], empirical dynamic modeling [677, 765], modeling emergent behavior [603], the nonlinear autoregressive model with exogenous inputs (NARMAX) [86, 281, 650, 774], and automated inference of dynamics [188, 189, 641]. Broadly speaking, these techniques may be classified as system identification, where methods from statistics and machine learning are used to identify dynamical systems from data. Nearly all methods of system identification involve some form of regression of data onto dynamics, and the main distinction between the various techniques is the degree to which this regression is constrained. For example, the dynamic mode decomposition generates best-fit linear models. Recent nonlinear regression techniques have produced nonlinear dynamic models that preserve physical constraints, such as conservation of energy. A major breakthrough in automated nonlinear system identification was made by Bongard and Lipson [95] and Schmidt and Lipson [640], where they used genetic programming to identify the structure of nonlinear dynamics. These methods are highly flexible and impose very few constraints on the form of the dynamics identified. In addition, SINDy is closely related to NARMAX [86], which identifies the structure of models from time-series data through an orthogonal least-squares procedure.

Discovering Partial Differential Equations

A major extension of the SINDy modeling framework generalized the library to include partial derivatives, enabling the identification of partial differential equations [613, 626]. The resulting algorithm, called the partial differential equation functional identification of nonlinear dynamics (PDE-FIND), has been demonstrated to successfully identify several canonical PDEs from classical physics, purely from noisy data. These PDEs include Navier–Stokes, Kuramoto–Sivashinsky, Schrödinger, reaction–diffusion, Burgers, Korteweg–de Vries, and the diffusion equation for Brownian motion [613].

PDE-FIND is similar to SINDy, in that it is based on sparse regression in a library constructed from measurement data. The sparse regression and discovery method is shown in Fig. 7.7. PDE-FIND is outlined below for PDEs in a single variable, although the theory is readily generalized to higher-dimensional PDEs. The spatial time-series data is arranged into a single column vector $\Upsilon \in$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

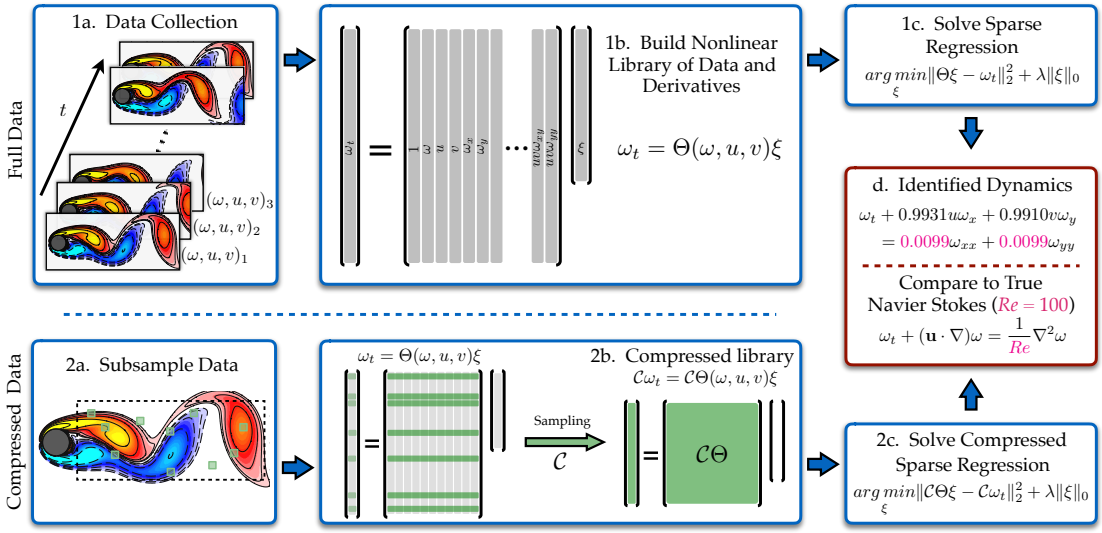


Figure 7.7: Steps in the PDE-FIND algorithm, applied to infer the Navier–Stokes equations from data. 1a. Data is collected as snapshots of a solution to a PDE. 1b. Numerical derivatives are taken and data is compiled into a large matrix $\boldsymbol{\Theta}$, incorporating candidate terms for the PDE. 1c. Sparse regression is used to identify active terms in the PDE. 2a. For large data sets, sparse sampling may be used to reduce the size of the problem. 2b. Subsampling the data set is equivalent to taking a subset of rows from the linear system in (7.52). 2c. An identical sparse regression problem is formed but with fewer rows. d. Active terms in $\boldsymbol{\xi}$ are synthesized into a PDE. Reproduced from Rudy et al. [613].

$\mathbb{C}^{mn}$, representing data collected over m time points and n spatial locations. Additional inputs, such as a known potential for the Schrödinger equation, or the magnitude of complex data, are arranged into a column vector $\mathbf{Q} \in \mathbb{C}^{mn}$. Next, a library $\boldsymbol{\Theta}(\boldsymbol{\Upsilon}, \mathbf{Q}) \in \mathbb{C}^{mn \times D}$ of D candidate linear and nonlinear terms and partial derivatives for the PDE is constructed. Derivatives are taken either using finite differences for clean data, or, when noise is added, with polynomial interpolation. The candidate linear and nonlinear terms and partial derivatives are then combined into a matrix $\boldsymbol{\Theta}(\boldsymbol{\Upsilon}, \mathbf{Q})$ which takes the form

$$\boldsymbol{\Theta}(\boldsymbol{\Upsilon}, \mathbf{Q}) = \begin{bmatrix} 1 & \boldsymbol{\Upsilon} & \boldsymbol{\Upsilon}^2 & \cdots & \mathbf{Q} & \cdots & \boldsymbol{\Upsilon}_x & \boldsymbol{\Upsilon}\boldsymbol{\Upsilon}_x & \cdots \end{bmatrix}. \tag{7.51}$$

Each column of $\boldsymbol{\Theta}$ contains all of the values of a particular candidate function across all of the mn space–time grid points on which data is collected. The time derivative $\boldsymbol{\Upsilon}_t$ is also computed and reshaped into a column vector. Figure 7.7 demonstrates the data collection and processing. As an example, a column of $\boldsymbol{\Theta}(\boldsymbol{\Upsilon}, \mathbf{Q})$ may be qu_x^2.

The PDE evolution can be expressed in this library as follows:

$$\boldsymbol{\Upsilon}_t = \boldsymbol{\Theta}(\boldsymbol{\Upsilon}, \mathbf{Q})\boldsymbol{\xi}. \tag{7.52}$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Each entry in $\boldsymbol{\xi}$ is a coefficient corresponding to a term in the PDE, and, for canonical PDEs, the vector $\boldsymbol{\xi}$ is *sparse*, meaning that only a few terms are active.

If the library $\boldsymbol{\Theta}$ has a sufficiently rich column space that the dynamics are in its span, then the PDE should be well represented by (7.52) with a sparse vector of coefficients $\boldsymbol{\xi}$. To identify the few active terms in the dynamics, a sparsity-promoting regression is employed, as in SINDy. Importantly, the regression problem in (7.52) may be poorly conditioned. Error in computing the derivatives will be magnified by numerical errors when inverting $\boldsymbol{\Theta}$. Thus a least-squares regression radically changes the qualitative nature of the inferred dynamics.

In general, we seek the sparsest vector $\boldsymbol{\xi}$ that satisfies (7.52) with a small residual. Instead of an intractable combinatorial search through all possible sparse vector structures, a common technique is to relax the problem to a convex ℓ_1-regularized least-squares [702]; however, this tends to perform poorly with highly correlated data. Instead, we use ridge regression with hard thresholding, which we call sequential threshold ridge regression (STRidge in Algorithm 1, reproduced from Rudy et al. [613]). For a given tolerance and threshold λ, this gives a sparse approximation to $\boldsymbol{\xi}$. We iteratively refine the tolerance of Algorithm 1 to find the best predictor based on the selection criteria,

$$\hat{\boldsymbol{\xi}} = \text{argmin}_{\boldsymbol{\xi}} \|\boldsymbol{\Theta}(\boldsymbol{\Upsilon}, \mathbf{Q})\boldsymbol{\xi} - \boldsymbol{\Upsilon}_t\|_2^2 + \epsilon\kappa(\boldsymbol{\Theta}(\boldsymbol{\Upsilon}, \mathbf{Q}))\|\boldsymbol{\xi}\|_0, \tag{7.53}$$

where $\kappa(\boldsymbol{\Theta})$ is the condition number of the matrix $\boldsymbol{\Theta}$, providing stronger regularization for ill-posed problems. Penalizing $\|\boldsymbol{\xi}\|_0$ discourages overfitting by selecting from the optimal position in a Pareto front.

Algorithm 1: STRidge($\boldsymbol{\Theta}, \boldsymbol{\Upsilon}_t, \lambda, tol,$ iters) [613].

$\hat{\boldsymbol{\xi}} = \arg\min_{\boldsymbol{\xi}} \|\boldsymbol{\Theta}\boldsymbol{\xi} - \boldsymbol{\Upsilon}_t\|_2^2 + \lambda\|\boldsymbol{\xi}\|_2^2$ % ridge regression
bigcoeffs = $\{j : |\hat{\xi}_j| \geq tol\}$ % select large coefficients
$\hat{\boldsymbol{\xi}}[\sim$ bigcoeffs] = 0 % apply hard threshold
$\hat{\boldsymbol{\xi}}$[bigcoeffs] = STRidge($\boldsymbol{\Theta}[:,$ bigcoeffs$], \boldsymbol{\Upsilon}_t, tol,$ iters $- 1$)
% recursive call with fewer coefficients
return $\hat{\boldsymbol{\xi}}$

As in the SINDy algorithm, it is important to provide sufficiently rich training data to disambiguate between several different models. For example, Fig. 7.8 illustrates the use of the PDE-FIND algorithm identifying the Korteweg–de Vries (KdV) equation. If only a single traveling wave is analyzed, the method incorrectly identifies the standard linear advection equation, as this is the simplest equation that describes a single traveling wave. However, if two traveling waves of different amplitudes are analyzed, the KdV equation is correctly identified, as it describes the different amplitude-dependent wave speeds.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

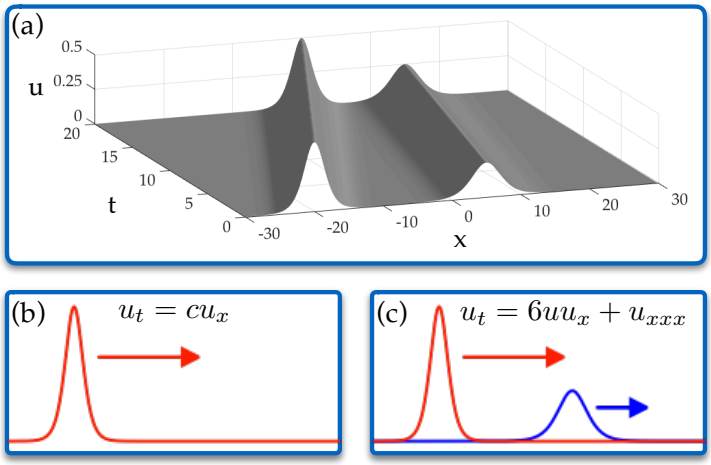


Figure 7.8: Inferring nonlinearity via observing solutions at multiple amplitudes. (a) An example two-soliton solution to the KdV equation. (b) Applying our method to a single-soliton solution determines that it solves the standard advection equation. (c) Looking at two completely separate solutions reveals nonlinearity. Reproduced from Rudy et al. [613].

The PDE-FIND algorithm can also be used to identify PDEs based on Lagrangian measurements that follow the path of individual particles. For example, Fig. 7.9 illustrates the identification of the diffusion equation describing Brownian motion of a particle based on a single long time-series measurement of the particle position. In this example, the time series is broken up into several short sequences, and the evolution of the distribution of these positions is used to identify the diffusion equation.

Extension of SINDy for Rational Function Nonlinearities

Many dynamical systems, such as metabolic and regulatory networks in biology, contain rational function nonlinearities in the dynamics. Often, these rational function nonlinearities arise because of a separation of timescales. Although the original SINDy algorithm is highly flexible in terms of the choice of the library of nonlinearities, it is not straightforward to identify rational functions, since general rational functions are not sparse linear combinations of a few basis functions. Instead, it is necessary to reformulate the dynamics in an *implicit* ordinary differential equation and modify the optimization procedure accordingly, as in Mangan et al. [478] and Kaheman et al. [364].

We consider dynamical systems with rational nonlinearities:

$$\dot{x}_k = \frac{f_N(\mathbf{x})}{f_D(\mathbf{x})}, \tag{7.54}$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

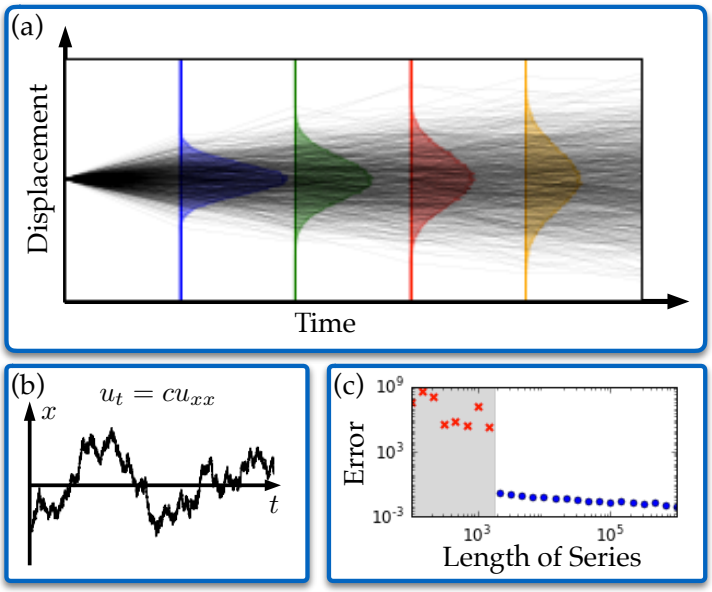


Figure 7.9: Inferring the diffusion equation from a single Brownian motion. (a) The time series is broken into many short random walks that are used to construct histograms of the displacement. (b) The Brownian motion trajectory, following the diffusion equation. (c) Parameter error ($\|\boldsymbol{\xi}^* - \hat{\boldsymbol{\xi}}\|_1$) versus length of known time series. Blue symbols correspond to correct identification of the structure of the diffusion model, $u_t = cu_{xx}$. Reproduced from Rudy et al. [613].

where x_k is the kth variable, and $f_N(\mathbf{x})$ and $f_D(\mathbf{x})$ represent numerator and denominator polynomials in the state variable $\mathbf{x}$. For each index k, it is possible to multiply both sides by the denominator f_D, resulting in the equation:

$$f_N(\mathbf{x}) - f_D(\mathbf{x})\dot{x}_k = 0. \tag{7.55}$$

The implicit form of (7.55) motivates a generalization of the function library $\boldsymbol{\Theta}$ in (7.47) in terms of the state $\mathbf{x}$ and the derivative $\dot{x}_k$:

$$\boldsymbol{\Theta}(\mathbf{X}, \dot{x}_k(\mathbf{t})) = \begin{bmatrix} \boldsymbol{\Theta}_N(\mathbf{X}) & \text{diag}(\dot{x}_k(\mathbf{t}))\boldsymbol{\Theta}_D(\mathbf{X}) \end{bmatrix}. \tag{7.56}$$

The first term, $\boldsymbol{\Theta}_N(\mathbf{X})$, is the library of numerator monomials in $\mathbf{x}$, as in (7.47). The second term, $\text{diag}(\dot{x}_k(\mathbf{t}))\boldsymbol{\Theta}_D(\mathbf{X})$, is obtained by multiplying each column of the library of denominator polynomials $\boldsymbol{\Theta}_D(\mathbf{X})$ with the vector $\dot{x}_k(\mathbf{t})$ in an element-wise fashion. For a single variable x_k, this would give the following:

$$\text{diag}(\dot{x}_k(\mathbf{t}))\boldsymbol{\Theta}(\mathbf{X}) = \begin{bmatrix} \dot{x}_k(\mathbf{t}) & (\dot{x}_k x_k)(\mathbf{t}) & (\dot{x}_k x_k^2)(\mathbf{t}) & \cdots \end{bmatrix}. \tag{7.57}$$

In most cases, we will use the same polynomial degree for both the numerator and denominator library, so that $\boldsymbol{\Theta}_N(\mathbf{X}) = \boldsymbol{\Theta}_D(\mathbf{X})$. Thus, the augmented library in (7.56) is only twice the size of the original library in (7.47).

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

We may now write the dynamics in (7.55) in terms of the augmented library in (7.56):

$$\boldsymbol{\Theta}(\mathbf{X}, \dot{x}_k(\mathbf{t}))\boldsymbol{\xi}_k = \mathbf{0}. \tag{7.58}$$

The sparse vector of coefficients $\boldsymbol{\xi}_k$ will have non-zero entries for the active terms in the dynamics. However, it is not possible to use the same sparse regression procedure as in SINDy, since the sparsest vector $\boldsymbol{\xi}_k$ that satisfies (7.58) is the trivial zero vector.

Instead, the sparsest non-zero vector $\boldsymbol{\xi}_k$ that satisfies (7.58) is identified as the sparsest vector in the null space of $\boldsymbol{\Theta}$. This is generally a non-convex problem, although there are recent algorithms developed by Qu et al. [576], based on the alternating directions method (ADM), to identify the sparsest vector in a subspace. Unlike the original SINDy algorithm, this procedure is quite sensitive to noise, as the null space is numerically approximated as the span of the singular vectors corresponding to small singular value. When noise is added to the data matrix $\mathbf{X}$, and hence to $\boldsymbol{\Theta}$, the noise floor of the singular value decomposition goes up, increasing the rank of the numerical null space.

A recent technique by Kaheman et al. [364] circumvents this ill-conditioned search through the null space of $\boldsymbol{\Theta}$. Instead, this approach picks a candidate term from the library and moves it to the right-hand side, so that the regression problem no longer involves a null space. Candidate terms are tested until one is found that is actually in the model, after which it is possible to find a sparse model that is also accurate.

General Formulation for Implicit ODEs

The optimization procedure above may be generalized to include a larger class of implicit ordinary differential equations, in addition to those containing rational function nonlinearities. The library $\boldsymbol{\Theta}(\mathbf{X}, \dot{x}_k(\mathbf{t}))$ contains a subset of the columns of the library $\boldsymbol{\Theta}(\begin{bmatrix}\mathbf{X} & \dot{\mathbf{X}}\end{bmatrix})$, which is obtained by building nonlinear functions of the state $\mathbf{x}$ and derivative $\dot{\mathbf{x}}$. Identifying the sparsest vector in the null space of $\boldsymbol{\Theta}(\begin{bmatrix}\mathbf{X} & \dot{\mathbf{X}}\end{bmatrix})$ provides more flexibility in identifying nonlinear equations with mixed terms containing various powers of any combination of derivatives and states. For example, the system given by

$$\dot{x}^2 x^2 - \dot{x}x - x^2 = 0 \tag{7.59}$$

may be represented as a sparse vector in the null space of $\boldsymbol{\Theta}(\begin{bmatrix}\mathbf{X} & \dot{\mathbf{X}}\end{bmatrix})$. This formulation may be extended to include higher-order derivatives in the library $\boldsymbol{\Theta}$, for example to identify second-order implicit differential equations:

$$\boldsymbol{\Theta}(\begin{bmatrix}\mathbf{X} & \dot{\mathbf{X}} & \ddot{\mathbf{X}}\end{bmatrix}). \tag{7.60}$$

The generality of this approach enables the identification of many systems of interest, including those systems with rational function nonlinearities.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

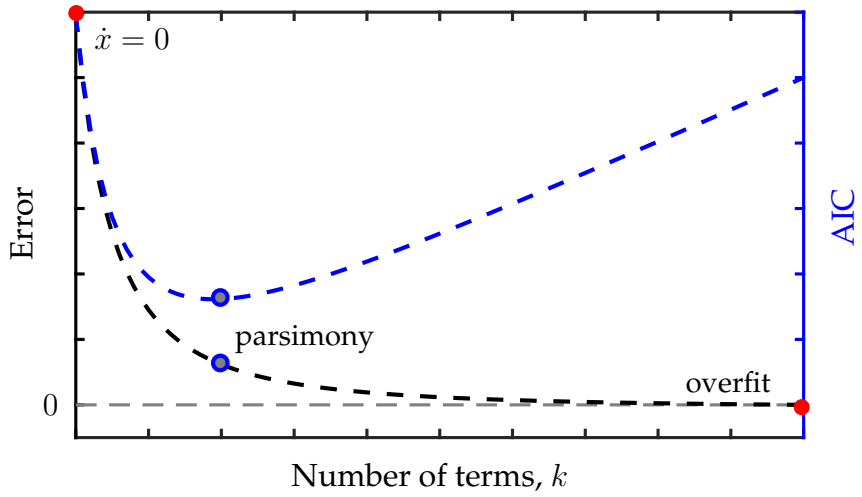


Figure 7.10: Illustration of model selection using SINDy and information criteria, as in Mangan et al. [479]. The most parsimonious model on the Pareto front is chosen to minimize the AIC score (blue circle), preventing overfitting.

Information Criteria for Model Selection

When performing the sparse regression in the SINDy algorithm, the sparsity-promoting parameter λ is a free variable. In practice, different values of λ will result in different models with various levels of sparsity, ranging from the trivial model $\dot{x} = 0$ for very large λ to the simple least-squares solution for $\lambda = 0$. Thus, by varying λ, it is possible to sweep out a Pareto front, balancing error versus complexity, as in Fig. 7.10. To identify the most parsimonious model, with low error and a reasonable complexity, it is possible to leverage information criteria for model selection, as described in Mangan et al. [479]. In particular, if we compute the Akaike information criterion (AIC) [8, 9], which penalizes the number of terms in the model, then the most parsimonious model minimizes the AIC. This procedure has been applied to several sparse identification problems, and in every case the true model was correctly identified [479].

7.4 Koopman Operator Theory

Koopman operator theory has recently emerged as an alternative perspective for dynamical systems in terms of the evolution of measurements $g(\mathbf{x})$. In 1931, Bernard O. Koopman demonstrated that it is possible to represent a nonlinear dynamical system in terms of an infinite-dimensional linear operator acting on a Hilbert space of measurement functions of the state of the system. This

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

so-called *Koopman operator* is linear, and its spectral decomposition completely characterizes the behavior of a nonlinear system, analogous to (7.7). However, it is also infinite-dimensional, as there are infinitely many degrees of freedom required to describe the space of all possible measurement functions g of the state. This poses new challenges. Obtaining finite-dimensional, matrix approximations of the Koopman operator is the focus of intense research efforts and holds the promise of enabling globally linear representations of nonlinear dynamical systems. Expressing nonlinear dynamics in a linear framework is appealing because of the wealth of optimal estimation and control techniques available for linear systems (see Chapter 8) and the ability to analytically predict the future state of the system. Obtaining a finite-dimensional approximation of the Koopman operator has been challenging in practice, as it involves identifying a subspace spanned by a subset of eigenfunctions of the Koopman operator. For a more complete discussion of modern Koopman theory and data-driven approximations, see [128].

Mathematical Formulation of Koopman Theory

The Koopman operator advances measurement functions of the state with the flow of the dynamics. We consider real-valued measurement functions $g : \mathbf{M} \to \mathbb{R}$, which are elements of an infinite-dimensional Hilbert space. The functions g are also commonly known as *observables*, although this may be confused with the unrelated *observability* from control theory. Typically, the Hilbert space is given by the Lebesgue square-integrable functions on $\mathbf{M}$; other choices of a measure space are also valid.

The Koopman operator $\mathcal{K}_t$ is an infinite-dimensional linear operator that acts on measurement functions g as

$$\mathcal{K}_t g = g \circ \mathbf{F}_t, \tag{7.61}$$

where $\circ$ is the composition operator. For a discrete-time system with time-step Δt, this becomes

$$\mathcal{K}_{\Delta t} g(\mathbf{x}_k) = g(\mathbf{F}_{\Delta t}(\mathbf{x}_k)) = g(\mathbf{x}_{k+1}). \tag{7.62}$$

In other words, the Koopman operator defines an infinite-dimensional linear dynamical system that advances the observation of the state $g_k = g(\mathbf{x}_k)$ to the next time-step:

$$g(\mathbf{x}_{k+1}) = \mathcal{K}_{\Delta t} g(\mathbf{x}_k). \tag{7.63}$$

Note that this is true for *any* observable function g and for any state $\mathbf{x}_k$.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

The Koopman operator is linear, a property that is inherited from the linearity of the addition operation in function spaces:

$$\mathcal{K}_t(\alpha_1 g_1(\mathbf{x}) + \alpha_2 g_2(\mathbf{x})) = \alpha_1 g_1(\mathbf{F}_t(\mathbf{x})) + \alpha_2 g_2(\mathbf{F}_t(\mathbf{x})) \tag{7.64a}$$

$$= \alpha_1 \mathcal{K}_t g_1(\mathbf{x}) + \alpha_2 \mathcal{K}_t g_2(\mathbf{x}). \tag{7.64b}$$

For sufficiently smooth dynamical systems, it is also possible to define the continuous-time analogue of the Koopman dynamical system in (7.63):

$$\frac{\mathrm{d}}{\mathrm{d}t} g = \mathcal{K} g. \tag{7.65}$$

The operator $\mathcal{K}$ is the infinitesimal generator of the one-parameter family of transformations $\mathcal{K}_t$ [4]. It is defined by its action on an observable function g:

$$\mathcal{K} g = \lim_{t \to 0} \frac{\mathcal{K}_t g - g}{t} = \lim_{t \to 0} \frac{g \circ \mathbf{F}_t - g}{t}. \tag{7.66}$$

The linear dynamical systems in (7.65) and (7.63) are analogous to the dynamical systems in (7.3) and (7.4), respectively. It is important to note that the original state $\mathbf{x}$ may be the observable, and the infinite-dimensional operator $\mathcal{K}_t$ will advance this function. However, the simple representation of the observable $g = \mathbf{x}$ in a chosen basis for Hilbert space may become arbitrarily complex once iterated through the dynamics. In other words, finding a representation for $\mathcal{K}\mathbf{x}$ may not be simple or straightforward.

Koopman Eigenfunctions and Intrinsic Coordinates

The Koopman operator is linear, which is appealing, but is infinite-dimensional, posing issues for representation and computation. Instead of capturing the evolution of all measurement functions in a Hilbert space, applied Koopman analysis attempts to identify key measurement functions that evolve linearly with the flow of the dynamics. Eigenfunctions of the Koopman operator provide just such a set of special measurements that behave linearly in time. In fact, a primary motivation to adopt the Koopman framework is the ability to simplify the dynamics through the eigendecomposition of the operator.

A discrete-time Koopman eigenfunction $\varphi(\mathbf{x})$ corresponding to eigenvalue λ satisfies

$$\varphi(\mathbf{x}_{k+1}) = \mathcal{K}_{\Delta t} \varphi(\mathbf{x}_k) = \lambda \varphi(\mathbf{x}_k). \tag{7.67}$$

In continuous time, a Koopman eigenfunction $\varphi(\mathbf{x})$ satisfies

$$\frac{\mathrm{d}}{\mathrm{d}t} \varphi(\mathbf{x}) = \mathcal{K} \varphi(\mathbf{x}) = \lambda \varphi(\mathbf{x}). \tag{7.68}$$

Obtaining Koopman eigenfunctions from data or from analytic expressions is a central applied challenge in modern dynamical systems. Discovering these

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

eigenfunctions enables globally linear representations of strongly nonlinear systems.

Applying the chain rule to the time derivative of the Koopman eigenfunction $\varphi(\mathbf{x})$ yields

$$\frac{\mathrm{d}}{\mathrm{d}t}\varphi(\mathbf{x}) = \nabla\varphi(\mathbf{x}) \cdot \dot{\mathbf{x}} = \nabla\varphi(\mathbf{x}) \cdot \mathbf{f}(\mathbf{x}). \tag{7.69}$$

Combined with (7.68), this results in a partial differential equation for the eigenfunction $\varphi(\mathbf{x})$:

$$\nabla\varphi(\mathbf{x}) \cdot \mathbf{f}(\mathbf{x}) = \lambda\varphi(\mathbf{x}). \tag{7.70}$$

With this nonlinear PDE, it is possible to approximate the eigenfunctions, either by solving for the Laurent series or with data via regression, both of which are explored below. This formulation assumes that the dynamics are both continuous and differentiable. The discrete-time dynamics in (7.4) are more general, although in many examples the continuous-time dynamics have a simpler representation than the discrete-time map for long times. For example, the simple Lorenz system has a simple continuous-time representation, yet is generally unrepresentable for even moderately long discrete-time updates.

The key takeaway from (7.67) and (7.68) is that the nonlinear dynamics become completely linear in eigenfunction coordinates, given by $\varphi(\mathbf{x})$. As a simple example, any conserved quantity of a dynamical system is a Koopman eigenfunction corresponding to eigenvalue $\lambda = 0$. This establishes a Koopman extension of the famous Noether's theorem [529], implying that any symmetry in the governing equations gives rise to a new Koopman eigenfunction with eigenvalue $\lambda = 0$. For example, the Hamiltonian energy function is a Koopman eigenfunction for a conservative system. In addition, the constant function $\varphi = 1$ is always a trivial eigenfunction corresponding to $\lambda = 0$ for every dynamical system.

Eigenvalue Lattices. Interestingly, a set of Koopman eigenfunctions may be used to generate more eigenfunctions. In discrete time, we find that the product of two eigenfunctions $\varphi_1(\mathbf{x})$ and $\varphi_2(\mathbf{x})$ is also an eigenfunction,

$$\mathcal{K}_t(\varphi_1(\mathbf{x})\varphi_2(\mathbf{x})) = \varphi_1(\mathbf{F}_t(\mathbf{x}))\varphi_2(\mathbf{F}_t(\mathbf{x})) \tag{7.71a}$$

$$= \lambda_1\lambda_2\varphi_1(\mathbf{x})\varphi_2(\mathbf{x}), \tag{7.71b}$$

corresponding to a new eigenvalue $\lambda_1\lambda_2$ given by the product of the two eigenvalues of $\varphi_1(\mathbf{x})$ and $\varphi_2(\mathbf{x})$. In continuous time, the relationship becomes

$$\mathcal{K}(\varphi_1\varphi_2) = \frac{\mathrm{d}}{\mathrm{d}t}(\varphi_1\varphi_2) \tag{7.72a}$$

$$= \dot{\varphi}_1\varphi_2 + \varphi_1\dot{\varphi}_2 \tag{7.72b}$$

$$= \lambda_1\varphi_1\varphi_2 + \lambda_2\varphi_1\varphi_2 \tag{7.72c}$$

$$= (\lambda_1 + \lambda_2)\varphi_1\varphi_2. \tag{7.72d}$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Interestingly, this means that the set of Koopman eigenfunctions establishes a commutative monoid under pointwise multiplication; a monoid has the structure of a group, except that the elements need not have inverses. Thus, depending on the dynamical system, there may be a finite set of *generator* eigenfunction elements that may be used to construct all other eigenfunctions. The corresponding eigenvalues similarly form a lattice, based on the product $\lambda_1\lambda_2$ or sum $\lambda_1 + \lambda_2$, depending on whether the dynamics are in discrete time or continuous time. For example, given a linear system $\dot{x} = \lambda x$, then $\varphi(x) = x$ is an eigenfunction with eigenvalue λ. Moreover, $\varphi^\alpha = x^\alpha$ is also an eigenfunction with eigenvalue $\alpha\lambda$ for any α.

The continuous-time and discrete-time lattices are related in a simple way. If the continuous-time eigenvalues are given by λ, then the corresponding discrete-time eigenvalues are given by $e^{\lambda t}$. Thus, the eigenvalue expressions in (7.71b) and (7.72d) are related as

$$e^{\lambda_1 t}e^{\lambda_2 t}\varphi_1(\mathbf{x})\varphi_2(\mathbf{x}) = e^{(\lambda_1+\lambda_2)t}\varphi_1(\mathbf{x})\varphi_2(\mathbf{x}). \tag{7.73}$$

As another simple demonstration of the relationship between continuous-time and discrete-time eigenvalues, consider the continuous-time definition in (7.66) applied to an eigenfunction:

$$\lim_{t\to 0}\frac{\mathcal{K}_t\varphi(\mathbf{x}) - \varphi(\mathbf{x})}{t} = \lim_{t\to 0}\frac{e^{\lambda t}\varphi(\mathbf{x}) - \varphi(\mathbf{x})}{t} = \lambda\varphi(\mathbf{x}). \tag{7.74}$$

Koopman Mode Decomposition and Finite Representations

Until now, we have considered scalar measurements of a system, and we uncovered special *eigen*-measurements that evolve linearly in time. However, we often take multiple measurements of a system. In extreme cases, we may measure the entire state of a high-dimensional spatial system, such as an evolving fluid flow. These measurements may then be arranged in a vector $\mathbf{g}$:

$$\mathbf{g}(\mathbf{x}) = \begin{bmatrix} g_1(\mathbf{x}) \\ g_2(\mathbf{x}) \\ \vdots \\ g_p(\mathbf{x}) \end{bmatrix}. \tag{7.75}$$

Each of the individual measurements may be expanded in terms of the eigenfunctions $\varphi_j(\mathbf{x})$, which provide a basis for Hilbert space:

$$g_i(\mathbf{x}) = \sum_{j=1}^{\infty} v_{ij}\varphi_j(\mathbf{x}). \tag{7.76}$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Thus, the vector of observables, $\mathbf{g}$, may be similarly expanded:

$$\mathbf{g}(\mathbf{x}) = \begin{bmatrix} g_1(\mathbf{x}) \\ g_2(\mathbf{x}) \\ \vdots \\ g_p(\mathbf{x}) \end{bmatrix} = \sum_{j=1}^{\infty} \varphi_j(\mathbf{x})\mathbf{v}_j, \tag{7.77}$$

where $\mathbf{v}_j$ is the jth *Koopman mode* associated with the eigenfunction φ_j.

For conservative dynamical systems, such as those governed by Hamiltonian dynamics, the Koopman operator is unitary on the Hilbert space of square-integrable functions. Thus, the Koopman eigenfunctions are orthonormal for conservative systems, and it is possible to compute the Koopman modes $\mathbf{v}_j$ directly by projection:

$$\mathbf{v}_j = \begin{bmatrix} \langle \varphi_j, g_1 \rangle \\ \langle \varphi_j, g_2 \rangle \\ \vdots \\ \langle \varphi_j, g_p \rangle \end{bmatrix}, \tag{7.78}$$

where $\langle \cdot, \cdot \rangle$ is the standard inner product of functions in Hilbert space. Thus, the expansion of the observable function in (7.77) may be thought of as a change of basis into eigenfunction coordinates. These Koopman modes have a physical interpretation in the case of direct spatial measurements of a system, $\mathbf{g}(\mathbf{x}) = \mathbf{x}$, in which case the modes are coherent *spatial* modes that behave linearly with the same temporal dynamics (i.e., oscillations, possibly with linear growth or decay). These Koopman modes $\mathbf{v}$ are also known as dynamic modes in DMD.

Given the decomposition in (7.77), it is possible to represent the dynamics of the measurements $\mathbf{g}$ as follows:

$$\mathbf{g}(\mathbf{x}_k) = \mathcal{K}_{\Delta t}^k \mathbf{g}(\mathbf{x}_0) = \mathcal{K}_{\Delta t}^k \sum_{j=0}^{\infty} \varphi_j(\mathbf{x}_0)\mathbf{v}_j \tag{7.79a}$$

$$= \sum_{j=0}^{\infty} \mathcal{K}_{\Delta t}^k \varphi_j(\mathbf{x}_0)\mathbf{v}_j \tag{7.79b}$$

$$= \sum_{j=0}^{\infty} \lambda_j^k \varphi_j(\mathbf{x}_0)\mathbf{v}_j, \tag{7.79c}$$

where $\mathcal{K}_{\Delta t}^k$ is the Koopman operator $\mathcal{K}_{\Delta t}$ applied k times. This sequence of triples, $\{(\lambda_j, \varphi_j, \mathbf{v}_j)\}_{j=0}^{\infty}$, is known as the *Koopman mode decomposition*, and was introduced by Mezić in 2005 [497]. Often, it is possible to approximate this expansion as a truncated sum of only a few dominant terms. The Koopman mode decomposition was later connected to data-driven regression via the dynamic mode decomposition [611], which was discussed in Section 7.2. The

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

DMD eigenvalues approximate the Koopman eigenvalues λ_j, the DMD modes approximate the Koopman modes $\mathbf{v}_j$, and the DMD mode amplitudes approximate the corresponding Koopman eigenfunctions evaluated at the initial condition $\varphi_j(\mathbf{x}_0)$. In fact, the Koopman mode decomposition in (7.79) is nearly identical to the DMD spectral expansion in (7.28), with the DMD mode amplitudes b_j replaced with the Koopman eigenfunctions $\varphi_j(\mathbf{x}_0)$ evaluated at the initial condition, and the DMD modes $\boldsymbol{\phi}_j$ replaced with the Koopman modes $\mathbf{v}_j$. It is important to note that the Koopman modes and eigenfunctions are distinct mathematical objects, requiring different approaches for approximation. Koopman eigenfunctions are often more challenging to compute than Koopman modes, motivating advanced techniques, such as the extended DMD algorithm [757] in Section 7.5.

Invariant Eigenspaces and Finite-Dimensional Models

Instead of capturing the evolution of all measurement functions in a Hilbert space, applied Koopman analysis approximates the evolution on an invariant subspace spanned by a finite set of measurement functions.

A *Koopman-invariant subspace* is defined as the span of a set of functions $\{g_1, g_2, \ldots, g_p\}$ if all functions g in this subspace,

$$g = \alpha_1 g_1 + \alpha_2 g_2 + \cdots + \alpha_p g_p, \tag{7.80}$$

remain in this subspace after being acted on by the Koopman operator $\mathcal{K}$:

$$\mathcal{K}g = \beta_1 g_1 + \beta_2 g_2 + \cdots + \beta_p g_p. \tag{7.81}$$

It is possible to obtain a finite-dimensional matrix representation of the Koopman operator by restricting it to an invariant subspace spanned by a finite number of functions $\{g_j\}_{j=0}^p$; this is illustrated in Fig. 7.11. The matrix representation $\mathbf{K}$ acts on a vector space $\mathbb{R}^p$, with the coordinates given by the values of $g_j(\mathbf{x})$. This induces a finite-dimensional linear system, as in (7.63) and (7.65).

Any finite set of eigenfunctions of the Koopman operator will span an invariant subspace. Discovering these eigenfunction coordinates is, therefore, a central challenge, as they provide intrinsic coordinates along which the dynamics behave linearly. In practice, it is more likely that we will identify an *approximately* invariant subspace, given by a set of functions $\{g_j\}_{j=0}^p$, where each of the functions g_j is well approximated by a finite sum of eigenfunctions: $g_j \approx \sum_{k=0}^p \alpha_k \varphi_k$.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

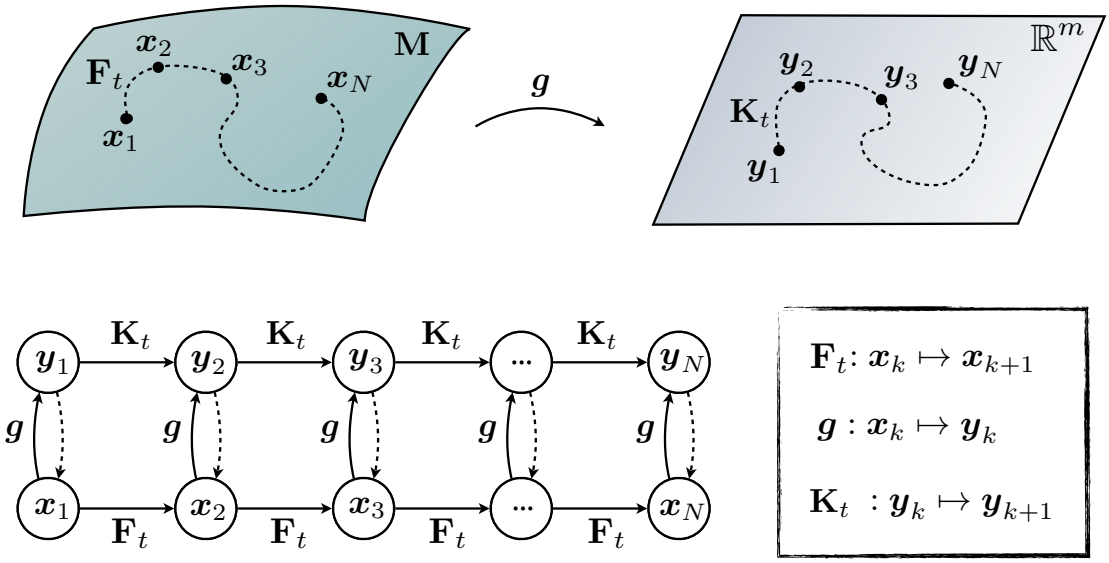


Figure 7.11: Schematic illustrating the Koopman operator for nonlinear dynamical systems. The dashed lines from $\mathbf{y}_k \to \mathbf{x}_k$ indicate that we would like to be able to recover the original state.

Examples of Koopman Embeddings

Nonlinear System with Single Fixed Point and a Slow Manifold

Here, we consider an example system with a single fixed point, given by

$$\dot{x}_1 = \mu x_1, \tag{7.82a}$$
$$\dot{x}_2 = \lambda(x_2 - x_1^2). \tag{7.82b}$$

For $\lambda < \mu < 0$, the system exhibits a slow attracting manifold given by $x_2 = x_1^2$. It is possible to augment the state $\mathbf{x}$ with the nonlinear measurement $g = x_1^2$, to define a three-dimensional Koopman-invariant subspace. In these coordinates, the dynamics become linear:

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} \mu & 0 & 0 \\ 0 & \lambda & -\lambda \\ 0 & 0 & 2\mu \end{bmatrix}\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} \quad \text{for} \quad \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_1^2 \end{bmatrix}. \tag{7.83}$$

The full three-dimensional Koopman observable vector space is visualized in Fig. 7.12. Trajectories that start on the invariant manifold $y_3 = y_1^2$, visualized by the blue parabolic surface, are constrained to stay on this manifold. There is a *slow* subspace, spanned by the eigenvectors corresponding to the slow eigenvalues μ and 2μ; this subspace is visualized by the green planar surface. Finally,

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

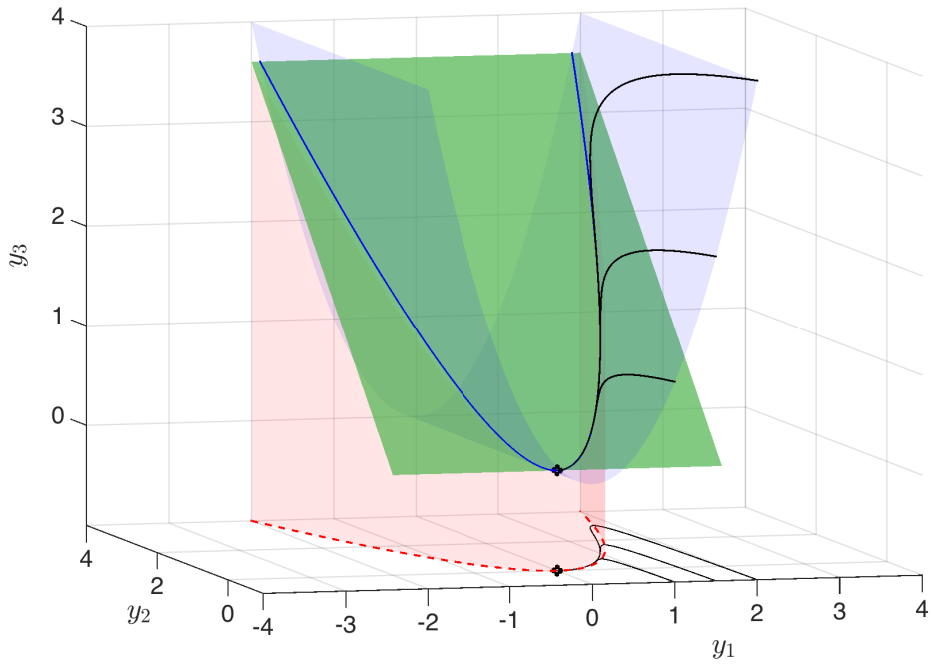


Figure 7.12: Visualization of three-dimensional linear Koopman system from (7.83) along with projection of dynamics onto the x_1–x_2 plane. The attracting slow manifold is shown in red, the constraint $y_3 = y_1^2$ is shown in blue, and the slow unstable subspace of (7.83) is shown in green. Black trajectories of the linear Koopman system in y project onto trajectories of the full nonlinear system in x in the y_1–y_2 plane. Here, $\mu = -0.05$ and $\lambda = 1$. Reproduced from Brunton et al. [127].

there is the original asymptotically attracting manifold of the original system, $y_2 = y_1^2$, which is visualized as the red parabolic surface. The blue and red parabolic surfaces always intersect in a parabola that is inclined at a 45° angle in the y_2–y_3 direction. The green surface approaches this 45° inclination as the ratio of fast to slow dynamics become increasingly large. In the full three-dimensional Koopman observable space, the dynamics produce a single stable node, with trajectories rapidly attracting onto the green subspace and then slowly approaching the fixed point.

Intrinsic Coordinates Defined by Eigenfunctions of the Koopman Operator. The left eigenvectors of the Koopman operator yield Koopman eigenfunctions (i.e., eigen-observables). The Koopman eigenfunctions of (7.83) corresponding

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

to eigenvalues μ and λ are

$$\varphi_\mu = x_1 \quad \text{and} \quad \varphi_\lambda = x_2 - bx_1^2 \quad \text{with} \quad b = \frac{\lambda}{\lambda - 2\mu}. \tag{7.84}$$

The constant b in φ_λ captures the fact that for a finite ratio λ/μ, the dynamics only shadow the asymptotically attracting slow manifold $x_2 = x_1^2$, but in fact follow neighboring parabolic trajectories. This is illustrated more clearly by the various surfaces in Fig. 7.12 for different ratios λ/μ.

In this way, a set of intrinsic coordinates may be determined from the observable functions defined by the left eigenvectors of the Koopman operator on an invariant subspace. Explicitly,

$$\varphi_\alpha(\mathbf{x}) = \boldsymbol{\xi}_\alpha \mathbf{y}(\mathbf{x}), \quad \text{where} \quad \boldsymbol{\xi}_\alpha \mathbf{K} = \alpha \boldsymbol{\xi}_\alpha. \tag{7.85}$$

These eigen-observables define observable subspaces that remain invariant under the Koopman operator, even after coordinate transformations. As such, they may be regarded as intrinsic coordinates [757] on the Koopman-invariant subspace.

Example of Intractable Representation

Consider the logistic map, given by

$$x_{k+1} = \beta x_k (1 - x_k). \tag{7.86}$$

Let our observable subspace include x and x^2:

$$\mathbf{y}_k = \begin{bmatrix} x \\ x^2 \end{bmatrix}_k \triangleq \begin{bmatrix} x_k \\ x_k^2 \end{bmatrix}. \tag{7.87}$$

Writing out the Koopman operator, the first row equation is simple:

$$\mathbf{y}_{k+1} = \begin{bmatrix} x \\ x^2 \end{bmatrix}_{k+1} = \begin{bmatrix} \beta & -\beta \\ ? & ? \end{bmatrix} \begin{bmatrix} x \\ x^2 \end{bmatrix}_k, \tag{7.88}$$

but the second row is not obvious. To find this expression, expand x_{k+1}^2:

$$x_{k+1}^2 = (\beta x_k (1 - x_k))^2 = \beta^2 (x_k^2 - 2x_k^3 + x_k^4). \tag{7.89}$$

Thus, cubic and quartic polynomial terms are required to advance x^2. Similarly, these terms need polynomials up to sixth and eighth order, respectively, and so on, *ad infinitum*:

$$\begin{bmatrix} x \\ x^2 \\ x^3 \\ x^4 \\ x^5 \\ \vdots \end{bmatrix}_{k+1} = \begin{array}{c} \begin{matrix} x & x^2 & x^3 & x^4 & x^5 & x^6 & x^7 & x^8 & x^9 & x^{10} & \end{matrix} \\ \begin{bmatrix} \beta & -\beta & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\ 0 & \beta^2 & -2\beta^2 & r^2 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\ 0 & 0 & \beta^3 & -3\beta^3 & 3\beta^3 & \beta^3 & 0 & 0 & 0 & 0 & \cdots \\ 0 & 0 & 0 & \beta^4 & -4\beta^4 & 6\beta^4 & -4\beta^4 & \beta^4 & 0 & 0 & \cdots \\ 0 & 0 & 0 & 0 & \beta^5 & -5\beta^5 & 10\beta^5 & -10\beta^5 & 5\beta^5 & -\beta^5 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \end{array} \begin{bmatrix} x \\ x^2 \\ x^3 \\ x^4 \\ x^5 \\ \vdots \end{bmatrix}_k.$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

It is interesting to note that the rows of this equation are related to the rows of Pascal's triangle, with the nth row scaled by r^n, and with the omission of the first row:

$$\left[x^0\right]_{k+1} = \left[0\right]\left[x^0\right]_k. \tag{7.90}$$

The above representation of the Koopman operator in a polynomial basis is somewhat troubling. Not only is there no closure, but the determinant of any finite-rank truncation is very large for $\beta > 1$. This illustrates a pitfall associated with naive representation of the infinite-dimensional Koopman operator for a simple chaotic system. Truncating the system, or performing a least-squares fit on an augmented observable vector (i.e., DMD on a nonlinear measurement; see Section 7.5), yields poor results, with the truncated system only agreeing with the true dynamics for a small handful of iterations, as the complexity of the representation grows quickly:

$$\begin{matrix} 1 \\ x \\ x^2 \\ x^3 \\ x^4 \\ x^5 \\ x^6 \\ x^7 \\ x^8 \\ \vdots \end{matrix}\begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \end{bmatrix} \overset{\mathcal{K}}{\Longrightarrow} \begin{bmatrix} 0 \\ \beta \\ -\beta \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \end{bmatrix} \overset{\mathcal{K}}{\Longrightarrow} \begin{bmatrix} 0 \\ \beta^2 \\ -\beta^2-\beta^3 \\ 2\beta^3 \\ -\beta^3 \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \end{bmatrix} \overset{\mathcal{K}}{\Longrightarrow} \begin{bmatrix} 0 \\ \beta^3 \\ -\beta^3-\beta^4-\beta^5 \\ 2\beta^4+2\beta^5+2\beta^6 \\ -\beta^4-\beta^5-6\beta^6-\beta^7 \\ 6\beta^6+4\beta^7 \\ -2\beta^6-6\beta^7 \\ 4\beta^7 \\ -\beta^7 \\ \vdots \end{bmatrix}. \tag{7.91}$$

Analytic Series Expansions for Eigenfunctions

Given the dynamics in (7.1), it is possible to solve the PDE in (7.70) using standard techniques, such as recursively solving for the terms in a Taylor or Laurent series. A number of simple examples are explored below.

Linear Dynamics

Consider the simple linear dynamics

$$\frac{\mathrm{d}}{\mathrm{d}t}x = x. \tag{7.92}$$

Assuming a Taylor series expansion for $\varphi(x)$:

$$\varphi(x) = c_0 + c_1 x + c_2 x^2 + c_3 x^3 + \cdots,$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

then the gradient and directional derivatives are given by

$$\begin{aligned}\nabla\varphi &= c_1 + 2c_2x + 3c_3x^2 + 4c_4x^3 + \cdots,\\ \nabla\varphi \cdot f &= c_1x + 2c_2x^2 + 3c_3x^3 + 4c_4x^4 + \cdots.\end{aligned}$$

Solving for terms in the Koopman eigenfunction PDE (7.70), we see that $c_0 = 0$ must hold. For any positive integer λ in (7.70), only one of the coefficients may be non-zero. Specifically, for $\lambda = k \in \mathbb{Z}^+$, then $\varphi(x) = cx^k$ is an eigenfunction for any constant c. For instance, if $\lambda = 1$, then $\varphi(x) = x$.

Quadratic Nonlinear Dynamics

Consider a nonlinear dynamical system

$$\frac{\mathrm{d}}{\mathrm{d}t}x = x^2. \tag{7.93}$$

There is no Taylor series that satisfies (7.70), except the trivial solution $\varphi = 0$ for $\lambda = 0$. Instead, we assume a Laurent series:

$$\begin{aligned}\varphi(x) = \cdots + c_{-3}x^{-3} + c_{-2}x^{-2} + c_{-1}x^{-1} + c_0\\ + c_1x + c_2x^2 + c_3x^3 + \cdots.\end{aligned}$$

The gradient and directional derivatives are given by

$$\begin{aligned}\nabla\varphi = \cdots - 3c_{-3}x^{-4} - 2c_{-2}x^{-3} - c_{-1}x^{-2}\\ + c_1 + 2c_2x + 3c_3x^2 + 4c_4x^3 + \cdots,\\ \nabla\varphi \cdot f = \cdots - 3c_{-3}x^{-2} - 2c_{-2}x^{-1} - c_{-1}\\ + c_1x^2 + 2c_2x^3 + 3c_3x^4 + 4c_4x^5 + \cdots.\end{aligned}$$

Solving for the coefficients of the Laurent series that satisfy (7.70), we find that all coefficients with positive index are zero, i.e., $c_k = 0$ for all $k \geq 1$. However, the non-positive index coefficients are given by the recursion $\lambda c_{k+1} = kc_k$, for negative $k \leq -1$. Thus, the Laurent series is

$$\varphi(x) = c_0\left(1 - \lambda x^{-1} + \frac{\lambda^2}{2}x^{-2} - \frac{\lambda^3}{3!}x^{-3} + \cdots\right) = c_0e^{-\lambda/x}.$$

This holds for all values of $\lambda \in \mathbb{C}$. There are also other Koopman eigenfunctions that can be identified from the Laurent series.

Polynomial Nonlinear Dynamics

For a more general nonlinear dynamical system

$$\frac{\mathrm{d}}{\mathrm{d}t}x = ax^n, \tag{7.94}$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

we have that

$$\varphi(x) = \exp\left\{\frac{\lambda}{(1-n)a}\, x^{1-n}\right\}$$

is an eigenfunction for all $\lambda \in \mathbb{C}$.

As mentioned above, it is also possible to generate new eigenfunctions by taking powers of these primitive eigenfunctions; the resulting eigenvalues generate a *lattice* in the complex plane.

History and Recent Developments

The original analysis of Koopman in 1931 was introduced to describe the evolution of measurements of Hamiltonian systems [402], and this theory was generalized by Koopman and von Neumann to systems with continuous eigenvalue spectrum in 1932 [403]. In the case of Hamiltonian flows, the Koopman operator $\mathcal{K}_t$ is unitary, and forms a one-parameter family of unitary transformations in Hilbert space. Unitary operators should be familiar by now, as the discrete Fourier transform (DFT) and the singular value decomposition (SVD) both provide unitary coordinate transformations. Unitarity implies that the inner product of any two observable functions remains unchanged through action of the Koopman operator, which is intuitively related to the phase-space volume-preserving property of Hamiltonian systems. In the original paper [402], Koopman drew connections between the Koopman eigenvalue spectrum and conserved quantities, integrability, and ergodicity. Interestingly, Koopman's 1931 paper was central in the celebrated proofs of the ergodic theorem by Birkhoff and von Neumann [88, 89, 510, 521].

Koopman analysis has recently gained renewed interest with the pioneering work of Mezić and collaborators [138, 139, 140, 427, 497, 498, 500]. The Koopman operator is also known as the composition operator, which is formally the pull-back operator on the space of scalar observable functions [4], and it is the dual, or left-adjoint, of the Perron–Frobenius operator, or transfer operator, which is the push-forward operator on the space of probability density functions. When a polynomial basis is chosen to represent the Koopman operator, then it is closely related to Carleman linearization [163, 164, 165], which has been used extensively in nonlinear control [51, 408, 674, 686]. Koopman analysis is also connected to the resolvent operator theory from fluid dynamics [655].

Recently, it has been shown that the operator-theoretic framework complements the traditional geometric and probabilistic perspectives. For example, level sets of Koopman eigenfunctions form invariant partitions of the state space of a dynamical system [139]; in particular, eigenfunctions of the Koopman operator may be used to analyze the ergodic partition [138, 501]. Koopman

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

analysis has also been recently shown to generalize the Hartman–Grobman theorem to the entire basin of attraction of a stable or unstable equilibrium point or periodic orbit [427].

At the time of this writing, representing Koopman eigenfunctions for general dynamical systems remains a central unsolved challenge. Significant research efforts are focused on developing data-driven techniques to identify Koopman eigenfunctions and use these for control, which will be discussed in the following sections and chapters. Recently, new work has emerged that attempts to leverage the power of deep learning to discover and represent eigenfunctions from data [465, 485, 540, 692, 747, 766].

7.5 Data-Driven Koopman Analysis

Obtaining linear representations for strongly nonlinear systems has the potential to revolutionize our ability to predict and control these systems. The linearization of dynamics near fixed points or periodic orbits has long been employed for *local* linear representation of the dynamics [334]. The Koopman operator is appealing because it provides a *global* linear representation, valid far away from fixed points and periodic orbits. However, previous attempts to obtain finite-dimensional approximations of the Koopman operator have had limited success. Dynamic mode decomposition [422, 611, 635] seeks to approximate the Koopman operator with a best-fit linear model advancing spatial measurements from one time to the next, although these linear measurements are not rich enough for many nonlinear systems. Augmenting DMD with nonlinear measurements may enrich the model, but there is no guarantee that the resulting models will be closed under the Koopman operator [127]. Here, we describe several approaches for identifying Koopman embeddings and eigenfunctions from data. These methods include the extended dynamic mode decomposition [757], extensions based on SINDy [365], and the use of delay coordinates [126].

Extended DMD

The extended DMD algorithm [757] is essentially the same as standard DMD [727], except that, instead of performing regression on direct measurements of the state, regression is performed on an augmented vector containing nonlinear measurements of the state. As discussed earlier, eDMD is equivalent to the variational approach of conformation dynamics [528, 534, 535], which was developed in 2013 by Noé and Nüske.

Here, we will modify the notation slightly to conform to related methods.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

In eDMD, an augmented state is constructed:

$$\mathbf{y} = \mathbf{\Theta}^T(\mathbf{x}) = \begin{bmatrix} \theta_1(\mathbf{x}) \\ \theta_2(\mathbf{x}) \\ \vdots \\ \theta_p(\mathbf{x}) \end{bmatrix}. \tag{7.95}$$

Here $\mathbf{\Theta}$ may contain the original state $\mathbf{x}$ as well as nonlinear measurements, so often $p \gg n$. Next, two data matrices are constructed, as in DMD:

$$\mathbf{Y} = \begin{bmatrix} | & | & & | \\ \mathbf{y}_1 & \mathbf{y}_2 & \cdots & \mathbf{y}_m \\ | & | & & | \end{bmatrix}, \quad \mathbf{Y}' = \begin{bmatrix} | & | & & | \\ \mathbf{y}_2 & \mathbf{y}_3 & \cdots & \mathbf{y}_{m+1} \\ | & | & & | \end{bmatrix}. \tag{7.96a}$$

Finally, a best-fit linear operator $\mathbf{A_Y}$ is constructed that maps $\mathbf{Y}$ into $\mathbf{Y}'$:

$$\mathbf{A_Y} = \underset{\mathbf{A_Y}}{\operatorname{argmin}} \|\mathbf{Y}' - \mathbf{A_Y}\mathbf{Y}\| = \mathbf{Y}'\mathbf{Y}^\dagger. \tag{7.97}$$

This regression may be written in terms of the data matrices $\mathbf{\Theta}(\mathbf{X})$ and $\mathbf{\Theta}(\mathbf{X}')$:

$$\mathbf{A_Y} = \underset{\mathbf{A_Y}}{\operatorname{argmin}} \|\mathbf{\Theta}^T(\mathbf{X}') - \mathbf{A_Y}\mathbf{\Theta}^T(\mathbf{X})\| = \mathbf{\Theta}^T(\mathbf{X}')(\mathbf{\Theta}^T(\mathbf{X}))^\dagger. \tag{7.98}$$

Because the augmented vector $\mathbf{y}$ may be significantly larger than the state $\mathbf{x}$, kernel methods are often employed to compute this regression [758]. In principle, the enriched library $\mathbf{\Theta}$ provides a larger basis in which to approximate the Koopman operator. It has been shown recently that, in the limit of infinite snapshots, the extended DMD operator converges to the Koopman operator projected onto the subspace spanned by $\mathbf{\Theta}$ [405]. However, if $\mathbf{\Theta}$ does not span a Koopman-invariant subspace, then the projected operator may not have any resemblance to the original Koopman operator, as all of the eigenvalues and eigenvectors may be different. In fact, it was shown that the extended DMD operator will have spurious eigenvalues and eigenvectors unless it is represented in terms of a Koopman-invariant subspace [127]. Therefore, it is essential to use validation and cross-validation techniques to ensure that eDMD models are not overfit, as discussed below. For example, it was shown that eDMD cannot contain the original state $\mathbf{x}$ as a measurement and represent a system that has multiple fixed points, periodic orbits, or other attractors, because these systems cannot be topologically conjugate to a finite-dimensional linear system [127].

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Approximating Koopman Eigenfunctions from Data

In discrete time, a Koopman eigenfunction $\varphi(\mathbf{x})$ evaluated at a number of data points in $\mathbf{X}$ will satisfy:

$$\begin{bmatrix} \lambda\varphi(\mathbf{x}_1) \\ \lambda\varphi(\mathbf{x}_2) \\ \vdots \\ \lambda\varphi(\mathbf{x}_m) \end{bmatrix} = \begin{bmatrix} \varphi(\mathbf{x}_2) \\ \varphi(\mathbf{x}_3) \\ \vdots \\ \varphi(\mathbf{x}_{m+1}) \end{bmatrix}. \tag{7.99}$$

It is possible to approximate this eigenfunction as an expansion in terms of a set of candidate functions,

$$\boldsymbol{\Theta}(\mathbf{x}) = \begin{bmatrix} \theta_1(\mathbf{x}) & \theta_2(\mathbf{x}) & \cdots & \theta_p(\mathbf{x}) \end{bmatrix}. \tag{7.100}$$

The Koopman eigenfunction may be approximated in this basis as

$$\varphi(\mathbf{x}) \approx \sum_{k=1}^{p} \theta_k(\mathbf{x})\xi_k = \boldsymbol{\Theta}(\mathbf{x})\boldsymbol{\xi}. \tag{7.101}$$

Writing (7.99) in terms of this expansion yields the matrix system:

$$(\lambda\boldsymbol{\Theta}(\mathbf{X}) - \boldsymbol{\Theta}(\mathbf{X}'))\boldsymbol{\xi} = \mathbf{0}. \tag{7.102}$$

If we seek the best *least-squares* fit to (7.102), this reduces to the extended DMD [757, 758] formulation:

$$\lambda\boldsymbol{\xi} = \boldsymbol{\Theta}(\mathbf{X})^{\dagger}\boldsymbol{\Theta}(\mathbf{X}')\boldsymbol{\xi}. \tag{7.103}$$

Note that (7.103) is the transpose of (7.98), so that left eigenvectors become right eigenvectors. Thus, the eigenvectors $\boldsymbol{\xi}$ of $\boldsymbol{\Theta}^{\dagger}\boldsymbol{\Theta}'$ yield the coefficients of the eigenfunction $\varphi(\mathbf{x})$ represented in the basis $\boldsymbol{\Theta}(\mathbf{x})$. It is absolutely essential then to confirm that predicted eigenfunctions actually behave linearly on trajectories, by comparing them with the predicted dynamics $\varphi_{k+1} = \lambda\varphi_k$, because the regression above will result in spurious eigenvalues and eigenvectors unless the basis elements θ_j span a Koopman-invariant subspace [127].

Sparse Identification of Eigenfunctions

It is possible to leverage the SINDy regression [132] to identify Koopman eigenfunctions corresponding to a particular eigenvalue λ, selecting only the few active terms in the library $\boldsymbol{\Theta}(\mathbf{x})$ to avoid overfitting. Given the data matrices, $\mathbf{X}$ and $\dot{\mathbf{X}}$ from above, it is possible to construct the library of basis functions $\boldsymbol{\Theta}(\mathbf{X})$ as well as a library of directional derivatives, representing the possible terms in $\nabla\varphi(\mathbf{x}) \cdot \mathbf{f}(\mathbf{x})$ from (7.70):

$$\boldsymbol{\Gamma}(\mathbf{x}, \dot{\mathbf{x}}) = \begin{bmatrix} \nabla\theta_1(\mathbf{x}) \cdot \dot{\mathbf{x}} & \nabla\theta_2(\mathbf{x}) \cdot \dot{\mathbf{x}} & \cdots & \nabla\theta_p(\mathbf{x}) \cdot \dot{\mathbf{x}} \end{bmatrix}. \tag{7.104}$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

It is then possible to construct $\mathbf{\Gamma}$ from data:

$$\mathbf{\Gamma}(\mathbf{X},\dot{\mathbf{X}}) = \begin{bmatrix} \nabla\theta_1(\mathbf{x}_1)\cdot\dot{\mathbf{x}}_1 & \nabla\theta_2(\mathbf{x}_1)\cdot\dot{\mathbf{x}}_1 & \cdots & \nabla\theta_p(\mathbf{x}_1)\cdot\dot{\mathbf{x}}_1 \\ \nabla\theta_1(\mathbf{x}_2)\cdot\dot{\mathbf{x}}_2 & \nabla\theta_2(\mathbf{x}_2)\cdot\dot{\mathbf{x}}_2 & \cdots & \nabla\theta_p(\mathbf{x}_2)\cdot\dot{\mathbf{x}}_2 \\ \vdots & \vdots & \ddots & \vdots \\ \nabla\theta_1(\mathbf{x}_m)\cdot\dot{\mathbf{x}}_m & \nabla\theta_2(\mathbf{x}_m)\cdot\dot{\mathbf{x}}_m & \cdots & \nabla\theta_p(\mathbf{x}_m)\cdot\dot{\mathbf{x}}_m \end{bmatrix}.$$

For a given eigenvalue λ, the Koopman PDE in (7.70) may be evaluated on data:

$$(\lambda\mathbf{\Theta}(\mathbf{X}) - \mathbf{\Gamma}(\mathbf{X},\dot{\mathbf{X}}))\boldsymbol{\xi} = \mathbf{0}. \tag{7.105}$$

The formulation in (7.105) is implicit, so that $\boldsymbol{\xi}$ will be in the null space of $\lambda\mathbf{\Theta}(\mathbf{X}) - \mathbf{\Gamma}(\mathbf{X},\dot{\mathbf{X}})$. The right null space of (7.105) for a given λ is spanned by the right singular vectors of $\lambda\mathbf{\Theta}(\mathbf{X}) - \mathbf{\Gamma}(\mathbf{X},\dot{\mathbf{X}}) = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^*$ (i.e., columns of $\mathbf{V}$) corresponding to zero-valued singular values. It may be possible to identify the few active terms in an eigenfunction by finding the sparsest vector in the null space [576], as in the implicit-SINDy algorithm [478] described in Section 7.3. In this formulation, the eigenvalues λ are not known *a priori*, and must be learned with the approximate eigenfunction. Koopman eigenfunctions and eigenvalues can also be determined as the solution to the eigenvalue problem $\mathbf{A}_\mathbf{Y}\boldsymbol{\xi}_\alpha = \lambda_\alpha\boldsymbol{\xi}_\alpha$, where $\mathbf{A}_\mathbf{Y} = \mathbf{\Theta}^\dagger\mathbf{\Gamma}$ is obtained via least-squares regression, as in the continuous-time version of eDMD. While many eigenfunctions are spurious, those corresponding to lightly damped eigenvalues can be well approximated.

From a practical standpoint, data in $\mathbf{X}$ does not need to be sampled from full trajectories, but can be obtained using more sophisticated strategies such as Latin hypercube sampling or sampling from a distribution over the phase space. Moreover, reproducing kernel Hilbert spaces (RKHS) can be employed to describe $\varphi(\mathbf{x})$ *locally* in patches of state space.

Example: Duffing System (Kaiser et al. [365]) We demonstrate the sparse identification of Koopman eigenfunctions on the undamped Duffing oscillator:

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} x_2 \\ x_1 - x_1^3 \end{bmatrix},$$

where x_1 is the position and x_2 is the velocity of a particle in a double-well potential with equilibria $(0,0)$ and $(\pm 1,0)$. This system is conservative, with Hamiltonian $\mathcal{H} = \frac{1}{2}x_2^2 - \frac{1}{2}x_1^2 + \frac{1}{4}x_1^4$. The Hamiltonian, and in general any conserved quantity, is a Koopman eigenfunction with zero eigenvalue.

For the eigenvalue $\lambda = 0$, (7.105) becomes $-\mathbf{\Gamma}(\mathbf{X},\dot{\mathbf{X}})\boldsymbol{\xi} = \mathbf{0}$, and hence a sparse $\boldsymbol{\xi}$ is sought in the null space of $-\mathbf{\Gamma}(\mathbf{X},\dot{\mathbf{X}})$. A library of candidate functions is constructed from data, employing polynomials up to fourth order:

$$\mathbf{\Theta}(\mathbf{X}) = \begin{bmatrix} | & | & | & | & & | \\ x_1(t) & x_2(t) & x_1^2(t) & x_1(t)x_2(t) & \cdots & x_2^4(t) \\ | & | & | & | & & | \end{bmatrix}$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

and

$$\mathbf{\Gamma}(\mathbf{X},\dot{\mathbf{X}}) = \begin{bmatrix} | & | & | & | & | \\ \dot{x}_1(t) & \dot{x}_2(t) & 2x_1(t)\dot{x}_1(t) & x_2(t)\dot{x}_1(t)+x_1(t)\dot{x}_2(t) & \cdots & 4x_2(t)^3\dot{x}_2(t) \\ | & | & | & | & | \end{bmatrix}.$$

A sparse vector of coefficients $\boldsymbol{\xi}$ may be identified, with the few non-zero entries determining the active terms in the Koopman eigenfunction. The identified Koopman eigenfunction associated with $\lambda = 0$ is

$$\varphi(\mathbf{x}) = -\tfrac{2}{3}x_1^2 + \tfrac{2}{3}x_2^2 + \tfrac{1}{3}x_1^4. \tag{7.106}$$

This eigenfunction matches the Hamiltonian perfectly up to a constant scaling.

Data-Driven Koopman and Delay Coordinates

Instead of advancing instantaneous linear or nonlinear measurements of the state of a system directly, as in DMD, it may be possible to obtain intrinsic measurement coordinates for Koopman based on time-delayed measurements of the system [25, 126, 190, 681]. This perspective is data-driven, relying on the wealth of information from previous measurements to inform the future. Unlike a linear or weakly nonlinear system, where trajectories may get trapped at fixed points or on periodic orbits, chaotic dynamics are particularly well suited to this analysis: trajectories evolve to densely fill an attractor, so more data provides more information. The use of delay coordinates may be especially important for systems with long-term memory effects, where the Koopman approach has recently been shown to provide a successful analysis tool [685].

The time-delay measurement scheme is shown schematically in Fig. 7.13, as illustrated on the Lorenz system for a single time series of the first variable, $x(t)$. The conditions of the Takens embedding theorem are satisfied [694], so it is possible to obtain a diffeomorphism between a delay-embedded attractor and the attractor in the original coordinates. We then obtain eigen-time-delay coordinates from a time series of a single measurement $x(t)$ by taking the SVD of the Hankel matrix $\mathbf{H}$:

$$\mathbf{H} = \begin{bmatrix} x(t_1) & x(t_2) & \cdots & x(t_p) \\ x(t_2) & x(t_3) & \cdots & x(t_{p+1}) \\ \vdots & \vdots & \ddots & \vdots \\ x(t_q) & x(t_{q+1}) & \cdots & x(t_m) \end{bmatrix} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^*. \tag{7.107}$$

The columns of $\mathbf{U}$ and $\mathbf{V}$ from the SVD are arranged hierarchically by their ability to model the columns and rows of $\mathbf{H}$, respectively. Often, $\mathbf{H}$ may admit a low-rank approximation by the first r columns of $\mathbf{U}$ and $\mathbf{V}$. Note that the Hankel matrix in (7.107) is the basis of the eigensystem realization algorithm [358]

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

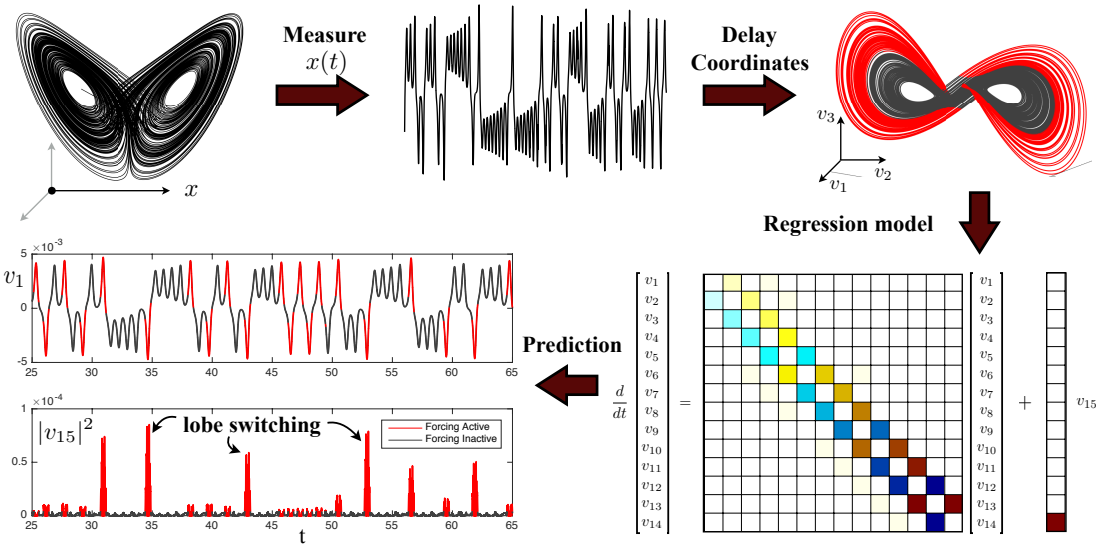


Figure 7.13: Decomposition of chaos into a linear system with forcing. A time series $x(t)$ is stacked into a Hankel matrix $\mathbf{H}$. The SVD of $\mathbf{H}$ yields a hierarchy of *eigen*-time series that produce a delay-embedded attractor. A best-fit linear regression model is obtained on the delay coordinates $\mathbf{v}$; the linear fit for the first $r-1$ variables is excellent, but the last coordinate v_r is not well modeled as linear. Instead, v_r is an input that forces the first $r-1$ variables. Rare forcing events correspond to lobe switching in the chaotic dynamics. This architecture is called the Hankel alternative view of Koopman (HAVOK) analysis, from [126]. Modified from Brunton et al. [126].

in linear system identification (see Section 9.3) and singular spectrum analysis (SSA) [120] in climate time-series analysis.

The low-rank approximation to (7.107) provides a *data-driven* measurement system that is approximately invariant to the Koopman operator for states on the attractor. By definition, the dynamics map the attractor into itself, making it *invariant* to the flow. In other words, the columns of $\mathbf{U}$ form a Koopman-invariant subspace. We may rewrite (7.107) with the Koopman operator $\mathcal{K} \triangleq \mathcal{K}_{\Delta t}$:

$$\mathbf{H} = \begin{bmatrix} x(t_1) & \mathcal{K}x(t_1) & \cdots & \mathcal{K}^{p-1}x(t_1) \\ \mathcal{K}x(t_1) & \mathcal{K}^2x(t_1) & \cdots & \mathcal{K}^{p}x(t_1) \\ \vdots & \vdots & \ddots & \vdots \\ \mathcal{K}^{q-1}x(t_1) & \mathcal{K}^{q}x(t_1) & \cdots & \mathcal{K}^{m-1}x(t_1) \end{bmatrix}. \tag{7.108}$$

The columns of (7.107) are well approximated by the first r columns of $\mathbf{U}$. The first r columns of $\mathbf{V}$ provide a time series of the magnitude of each of the

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

columns of $\mathbf{U\Sigma}$ in the data. By plotting the first three columns of $\mathbf{V}$, we obtain an embedded attractor for the Lorenz system (see Fig. 7.13).

The connection between eigen-time-delay coordinates from (7.107) and the Koopman operator motivates a linear regression model on the variables in $\mathbf{V}$. Even with an approximately Koopman-invariant measurement system, there remain challenges to identifying a linear model for a chaotic system. A linear model, however detailed, cannot capture multiple fixed points or the unpredictable behavior characteristic of chaos with a positive Lyapunov exponent [127]. Instead of constructing a closed linear model for the first r variables in $\mathbf{V}$, we build a linear model on the first $r-1$ variables and recast the last variable, v_r, as a forcing term:

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{v}(t) = \mathbf{A}\mathbf{v}(t) + \mathbf{B}v_r(t), \tag{7.109}$$

where $\mathbf{v} = \begin{bmatrix} v_1 & v_2 & \cdots & v_{r-1} \end{bmatrix}^T$ is a vector of the first $r-1$ eigen-time-delay coordinates. Other work has investigated the splitting of dynamics into deterministic linear and chaotic stochastic dynamics [497].

In all of the examples explored in [126], the linear model on the first $r-1$ terms is accurate, while no linear model represents v_r. Instead, v_r is an input forcing to the linear dynamics in (7.109), which approximates the nonlinear dynamics. The statistics of $v_r(t)$ are non-Gaussian, with long tails corresponding to rare-event forcing that drives lobe switching in the Lorenz system; this is related to rare-event forcing distributions observed and modeled by others [472, 473, 618]. The forced linear system in (7.109) was discovered after applying the SINDy algorithm [132] to delay coordinates of the Lorenz system. Continuing to develop Koopman on delay coordinates has significant promise in the context of closed-loop feedback control, where it may be possible to manipulate the behavior of a chaotic system by treating v_r as a disturbance.

In addition, the use of delay coordinates as intrinsic measurements for Koopman analysis suggests that Koopman theory may also be used to improve spatially distributed sensor technologies. A spatial array of sensors, for example the $\mathcal{O}(100)$ strain sensors on the wings of flying insects, may use phase delay coordinates to provide nearly optimal embeddings to detect and control convective structures (e.g., stall from a gust, leading-edge vortex formation and convection, etc.).

History of Delay Embeddings for Dynamics

The Hankel matrix has been used for decades in system identification, for example in the eigensystem realization algorithm (ERA) [358] and the singular spectrum analysis (SSA) [120]. These early algorithms were developed specifically for linear systems, and although they were often applied to weakly non-

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

linear systems, it was unclear how to interpret the resulting models and decompositions. Modern Koopman operator theory has provided a valuable new perspective for how to interpret the results of these classical Hankel-based approaches when applied to nonlinear systems. Computing DMD on a Hankel matrix was first introduced by Tu et al. [727] and was used by B. Brunton et al. [123] in the field of neuroscience. The connection between the Hankel matrix and the Koopman operator, along with the linear regression models in (7.109), was established by Brunton et al. [126] in the Hankel alternative view of Koopman (HAVOK) framework. Several subsequent works have provided additional theoretical foundations for this approach [25, 167, 190, 328, 371]. Hirsh et al. [328] established connections between HAVOK and the Frenet–Serret frame from differential geometry, motivating a more accurate computational modeling approach. The HAVOK approach is also often referred to as delay-DMD [727] or Hankel-DMD [25]. A connection between delay embeddings and the Koopman operator was established as early as 2004 by Mezić and Banaszuk [500], where a stochastic Koopman operator is defined and a statistical Takens theorem is proven. Other work has investigated the splitting of dynamics into deterministic linear and chaotic stochastic dynamics [497]. The use of delay coordinates may be especially important for systems with long-term memory effects and where the Koopman approach has recently been shown to provide a successful analysis tool [685].

HAVOK Code for Lorenz System

Code 7.6 below generates a HAVOK model for the same Lorenz system data generated in Code 7.2. Here we use $\Delta t = 0.01$, $m_o = 10$, and $r = 10$, although the results would be more accurate for $\Delta t = 0.001$, $m_o = 100$, and $r = 15$.

Code 7.6: [MATLAB] HAVOK code for Lorenz data generated in Section 7.1.

```
%% EIGEN-TIME DELAY COORDINATES
stackmax = 10;    %  Number of shift-stacked rows
r=10;             %  Rank of HAVOK Model
H = zeros(stackmax,size(x,1)-stackmax);
for k=1:stackmax
    H(k,:) = x(k:end-stackmax-1+k,1);
end
[U,S,V] = svd(H,'econ'); % Eigen delay coordinates

%% COMPUTE DERIVATIVES (4TH ORDER CENTRAL DIFFERENCE)
dV = zeros(length(V)-5,r);
for i=3:length(V)-3
    for k=1:r
        dV(i-2,k) = (1/(12*dt))*(-V(i+2,k)+8*V(i+1,k)-8*V(i
            -1,k)+V(i-2,k));
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

```
    end
end
% trim first and last two that are lost in derivative
V = V(3:end-3,1:r);

%%  BUILD HAVOK REGRESSION MODEL ON TIME DELAY COORDINATES
Xi = V\dV;
A = Xi(1:r-1,1:r-1)';
B = Xi(end,1:r-1)';
```

Code 7.6: [Python] HAVOK code for Lorenz data generated in Section 7.1.

```
## Eigen-time delay coordinates
stackmax = 10 # Number of shift-stacked rows
r = 10        # rank of HAVOK model
H = np.zeros((stackmax,x.shape[0]-stackmax))

for k in range(stackmax):
    H[k,:] = x[k:-(stackmax-k),0]
U,S,VT = np.linalg.svd(H,full_matrices=0)
V = VT.T

## Compute Derivatives (4th Order Central Difference)
dV = (1/(12*dt))*(-V[4:,:]+8*V[3:-1,:]-8*V[1:-3,:]+V[:-4,:])
# trim first and last two that are lost in derivative
V = V[2:-2]

## Build HAVOK Regression Model on Time Delay Coordinates
Xi = np.linalg.lstsq(V,dV,rcond=None)[0]
A = Xi[:(r-1),:(r-1)].T
B = Xi[-1,:(r-1)].T
```

Neural Networks for Koopman Embeddings

Despite the promise of Koopman embeddings, obtaining tractable representations has remained a central challenge. Recall that, even for relatively simple dynamical systems, the eigenfunctions of the Koopman operator may be arbitrarily complex. Deep learning, which is well suited for representing arbitrary functions, has recently emerged as a promising approach for discovering and representing Koopman eigenfunctions [440, 465, 485, 540, 692, 747, 766], providing a data-driven embedding of strongly nonlinear systems into intrinsic linear coordinates. In particular, the Koopman perspective fits naturally with the deep autoencoder structure discussed in Chapter 6, where a few key latent variables $\mathbf{y} = \boldsymbol{\varphi}(\mathbf{x})$ are discovered to parameterize the dynamics. In a Koopman network, an additional constraint is enforced so that the dynamics must be linear

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

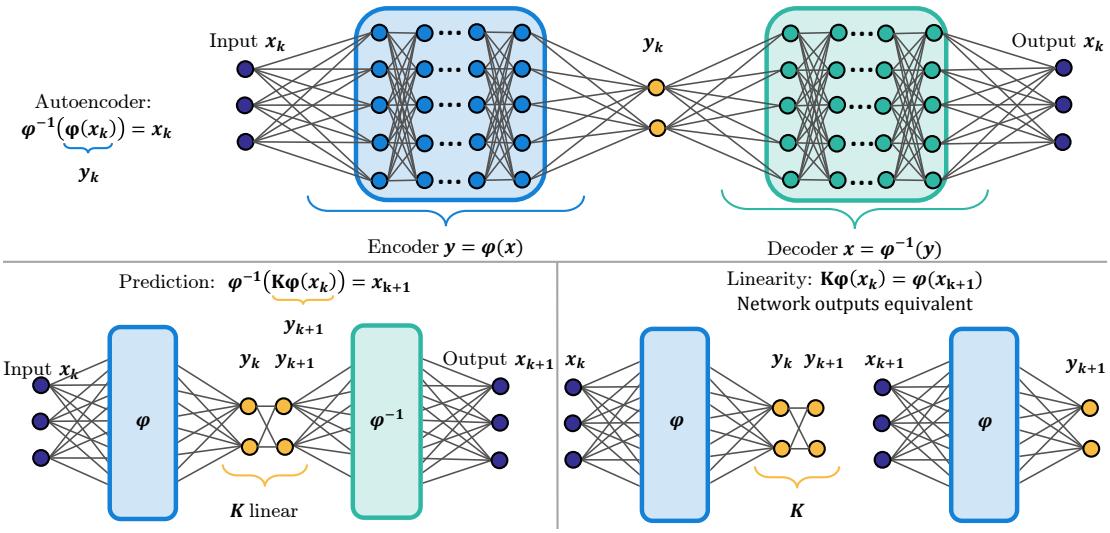


Figure 7.14: Deep neural network architecture used to identify Koopman eigenfunctions $\varphi(\mathbf{x})$. The network is based on a deep autoencoder (top), which identifies intrinsic coordinates $\mathbf{y} = \varphi(\mathbf{x})$. Additional loss functions are included to enforce linear dynamics in the autoencoder variables (bottom). Reproduced with permission from Lusch et al. [465].

on these latent variables, forcing the functions $\varphi(\mathbf{x})$ to be Koopman eigenfunctions, as illustrated in Fig. 7.14. The constraint of linear dynamics is enforced by the loss function $\|\varphi(\mathbf{x}_{k+1}) - \mathbf{K}\varphi(\mathbf{x}_k)\|$, where $\mathbf{K}$ is a matrix. In general, linearity is enforced over multiple time-steps, so that a trajectory is captured by iterating $\mathbf{K}$ on the latent variables. In addition, it is important to be able to map back to physical variables $\mathbf{x}$, which is why the autoencoder structure is favorable [465]. Variational autoencoders are also used for stochastic dynamical systems, such as molecular dynamics, where the map back to physical configuration space from the latent variables is probabilistic [485, 747].

For simple systems with a discrete eigenvalue spectrum, a compact representation may be obtained in terms of a few autoencoder variables. However, dynamical systems with continuous eigenvalue spectra defy low-dimensional representations using many existing neural network or Koopman representations. Continuous spectrum dynamics are ubiquitous, ranging from the simple pendulum to nonlinear optics and broadband turbulence. For example, the classical pendulum, given by

$$\ddot{x} = -\sin(\omega x), \tag{7.110}$$

exhibits a continuous range of frequencies, from ω to 0, as the amplitude of the pendulum oscillation is increased. Thus, the continuous spectrum confounds a simple description in terms of a few Koopman eigenfunctions [499]. Indeed, away from the linear regime, an infinite Fourier sum is required to approximate the shift in frequency.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

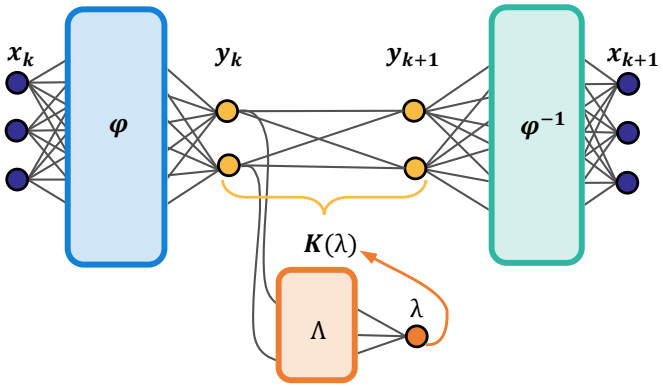


Figure 7.15: Modified network architecture with auxiliary network to parameterize the continuous eigenvalue spectrum. A continuous eigenvalue λ enables aggressive dimensionality reduction in the autoencoder, avoiding the need for higher harmonics of the fundamental frequency that are generated by the nonlinearity. Reproduced with permission from Lusch et al. [465].

In a recent work by Lusch et al. [465], an auxiliary network is used to parameterize the continuously varying eigenvalue, enabling a network structure that is both parsimonious and interpretable. This parameterized network is depicted schematically in Fig. 7.15 and illustrated on the simple pendulum in Fig. 7.16. In contrast to other network structures, which require a large autoencoder layer to encode the continuous frequency shift with an asymptotic expansion in terms of harmonics of the natural frequency, the parameterized network is able to identify a single complex conjugate pair of eigenfunctions with a varying imaginary eigenvalue pair. If this explicit frequency dependence is unaccounted for, then a high-dimensional network is necessary to account for the shifting frequency and eigenvalues.

It is expected that neural network representations of dynamical systems, and Koopman embeddings in particular, will remain a growing area of interest in data-driven dynamics. Combining the representational power of deep learning with the elegance and simplicity of Koopman embeddings has the potential to transform the analysis and control of complex systems.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

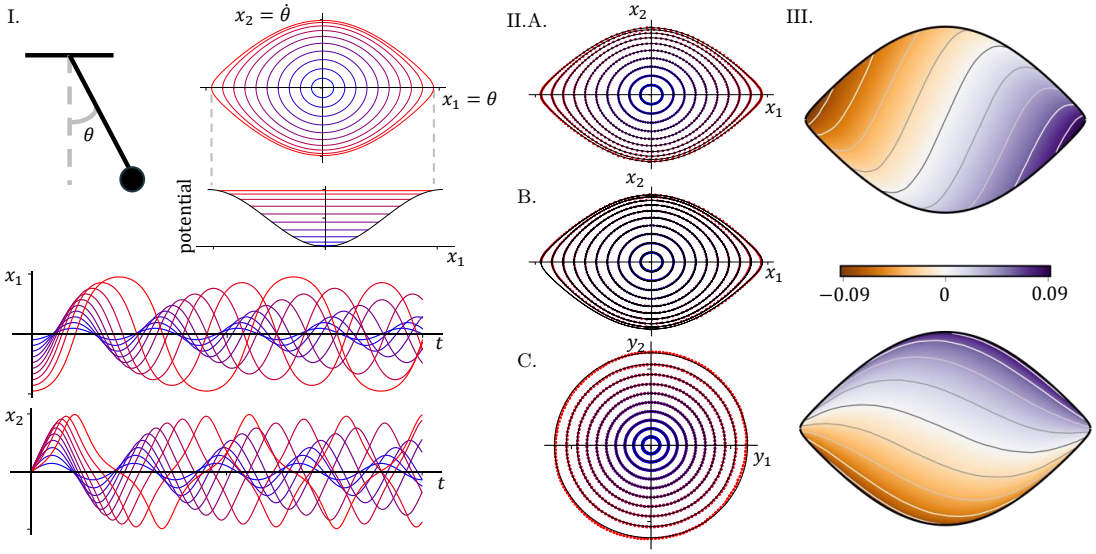


Figure 7.16: Neural network embedding of the nonlinear pendulum, using the parameterized network in Fig. 7.15. As the pendulum amplitude increases, the frequency continuously changes (I). In the Koopman eigenfunction coordinates (III), the dynamics become linear, given by perfect circles (II.C). Reproduced with permission from Lusch et al. [**?**]

Suggested Reading

Texts

(1) Nonlinear oscillations, dynamical systems, and bifurcations of vector fields, by P. Holmes and J. Guckenheimer, 1983 [334].

(2) Dynamic mode decomposition: Data-driven modeling of complex systems, by J. N. Kutz, S. L. Brunton, B. W. Brunton, and J. L. Proctor, 2016 [422].

(3) Differential equations and dynamical systems, by L. Perko, 2013 [562].

Papers and reviews

(1) Distilling free-form natural laws from experimental data, by M. Schmidt and H. Lipson, *Science*, 2009 [640].

(2) Discovering governing equations from data by sparse identification of nonlinear dynamical systems, by S. L. Brunton, J. L. Proctor, and J. N. Kutz, *Proceedings of the National Academy of Sciences*, 2016 [132].

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

(3) On dynamic mode decomposition: Theory and applications, by J. H. Tu, C. W. Rowley, D. M. Luchtenburg, S. L. Brunton, and J. N. Kutz, *Journal of Computational Dynamics*, 2014 [727].

(4) Hamiltonian systems and transformation in Hilbert space, by B. O. Koopman, *Proceedings of the National Academy of Sciences*, 1931 [402].

(5) Spectral properties of dynamical systems, model reduction and decompositions, by I. Mezić, *Nonlinear Dynamics*, 2005 [497].

(6) Data-driven model reduction and transfer operator approximation, by S. Klus, F. Nuske, P. Koltai, H. Wu, I. Kevrekidis, C. Schutte, and F. Noe, *Journal of Nonlinear Dynamics*, 2018 [392].

(7) Hidden physics models: Machine learning of nonlinear partial differential equations, by M. Raissi and G. E. Karniadakis, *Journal of Computational Physics*, 2018 [583].

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Homework

Exercise 7-1. Create a similar bifurcation diagram to the logistic map, but for the *hat* map, given by

$$x_{k+1} = \begin{cases} \beta x_k & \text{for } x_k \in [0, \frac{1}{2}), \\ \beta/2 - \beta x_k & \text{for } x_k \in [\frac{1}{2}, 1]. \end{cases}$$

For what values of β is this map chaotic?

Exercise 7-2. This exercise will explore how to compare trajectories of chaotic systems. For chaotic systems, simply comparing the solutions becomes difficult, because even minuscule changes in the initial conditions can give rise to entirely different solutions because of exponential growth of these small changes in time. Instead, it is often more natural to compare the probability distributions of the chaotic attractor.

(a) Generate two trajectories with nearby initial conditions, within an initial distance of 1×10^{-6}, for the Lorenz system with the standard parameters. Plot these two time series. Compute the error between the two trajectories as a function of time and explain the trend.

(b) For a given point on the attractor, find the perturbation direction that gives the largest error for $T = 1$ between the two trajectories.

(c) Now, we will compare the distribution of these trajectories using the Kullback–Leibler (KL) divergence, also known as the relative entropy. In this simple example, we will only compare the x coordinate of the two trajectories. The KL divergence between two discrete distributions $P(x)$ and $Q(x)$ is given by

$$D_{\mathrm{KL}}(P, Q) = \sum_{x \in \mathcal{X}} P(x) \log \left(\frac{P(x)}{Q(x)} \right), \tag{7.111}$$

where $\mathcal{X}$ is the set of discrete states. For our Lorenz example, we will compute a binned histogram of the x variable from -20 to 20 with bins of width 0.2, and this will be our discrete distribution. Compute the KL divergence of the two trajectories as a function of time. How does this trend differ from the error plot above?

(d) Now generate two trajectories starting from the same initial condition but with different parameters of the Lorenz system. Compute the KL divergence of these two solutions.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Exercise 7-3. This exercise will explore the finite-time Lyapunov exponent (FTLE) as a measure of local stretching between neighboring particles in a chaotic dynamical system.

(a) First, we will generate a set of approximately equally spaced points on the Lorenz attractor. Generate a long-time trajectory (at least $T = 100$) for the Lorenz system, starting from a standard initial condition and using standard parameters. Discard the first $t = 1$ portion of the trajectory. Create a library of points, initialized with the first point on the trajectory after $t = 1$. For each subsequent point on the trajectory, add it to the library if and only if it is a distance of greater than $\delta = 0.1$ from all other points in the library. This will be a sampling of the attractor.

(b) Now, for each point $\mathbf{x}_0$ in this set, find the nearest point $\mathbf{x}'$ on the attractor and compute the difference $\mathbf{x}_\delta = \mathbf{x}_0 - \mathbf{x}'$. Simulate $\mathbf{x}_0$ and $\mathbf{x}_\epsilon = \mathbf{x}_0 + \epsilon\mathbf{x}_\delta$ for $T = 1$, using $\epsilon = 0.01$. Repeat this for every point on the attractor. For each point, compute the finite-time Lyapunov exponent using the following formula:

$$\sigma = \frac{1}{T}\log\left(\frac{\|\mathbf{x}_\epsilon(T) - \mathbf{x}_0(T)\|_2}{\epsilon}\right).$$

Plot each point on the attractor, color-coded by the FTLE σ. Which points are the most sensitive? Is this consistent with your intuition?

(c) Repeat this experiment for various ϵ and T and explain the results.

(d) Finally, it is possible to create a proxy for this sensitivity using an adaptive step integrator, such as `rk45`. Simulate a long-time trajectory with a very small minimum Δt and a small error tolerance. Plot the trajectory color-coded by the Δt selected by the integrator. Is this consistent with the FTLE plots above?

Exercise 7-4. This exercise will test the data requirements for identifying an accurate SINDy model for the Lorenz system, following the work of Champion et al. [167]. First, we will use clean data without any measurement noise.

(a) Generate a long trajectory with a fine sampling rate of $\Delta t = 0.0001$; discard the first $t = 1$ of the data. Use SINDy to generate models using an increasing length of data T. At what T is it possible to identify the correct Lorenz mode? Plot the data up until T. Do the results surprise you? (Explain why.)

(b) Now, repeat this experiment for different sampling rates from $t = 0.0001$ to 0.1. How does the minimum data length T change? How does the number of samples $T/\Delta t$ change?

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

(c) Repeat the experiment with small additive Gaussian noise on the trajectory. With noise, repeat the identification for several noise realizations to obtain an average success rate. Also try different noise magnitudes. Repeat the main figure in the Champion et al. [167] paper.

Exercise 7-5. Generate and plot a trajectory for the Rossler system, given by

$$\begin{aligned}\dot{x} &= -y - z,\\ \dot{y} &= x + ay,\\ \dot{z} &= b + z(x - c),\end{aligned}$$

with parameters $a = 0.2$, $b = 0.2$, and $c = 14$. First, let us marvel at how a chaotic system can be generated with a single quadratic term (the xz term in the third equation). Use this trajectory to identify a SINDy model. Explore different sparsifying thresholds and different trajectory lengths. Now add a small amount of noise to the trajectory and re-identify the model, again with different thresholds and lengths. When there is sufficiently long trajectory data, what happens when the threshold is too large? When it is too small?

Exercise 7-6. This example will explore and compare the SINDy method and genetic programming to learn the equations of motion for a challenging system.

(a) Derive the equations of motion for a double pendulum and simulate for an initial condition near the double-inverted configuration.

(b) Use this data to generate a model using the SINDy method as in Kaheman et al. [364].

(c) Use this data to generate a model using the genetic programming approach as in Schmidt and Lipson [640].

(d) Repeat the above experiments with and without friction. Compare the results and discuss.

(e) Try using both approaches to learn conserved quantities for the double pendulum system in the absence of friction. Note that you will need to use a very accurate integrator to avoid numerical issues due to chaos.

Exercise 7-7. This exercise will explore identifying a PDE using PDE-FIND based on data from the Korteweg–de Vries (KdV) system

$$u_t + u_{xxx} - 6uu_x = 0.$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

This system gives rise to coherent traveling wave solutions, known as solitons, that retain their shape as they travel at a constant wave speed, despite the nonlinearity in the system. In general, a soliton solution will take the form

$$u(x,t) = -\frac{c}{2}\text{sech}^2\left(\frac{\sqrt{c}}{2}(x-ct)\right)$$

for an arbitrary positive wave speed c. Notice that the wave speed c is linked to the amplitude and width of the wave.

(a) Generate data by initializing a single-soliton solution

$$u(x,0) = -\tfrac{1}{2}\text{sech}^2(\tfrac{1}{2}(x)),$$

with a wave speed corresponding to $c = 1$ above. Try to identify a PDE model using PDE-FIND based on this data. What is the sparsest model that supports the data?

(b) Next, use data beginning at two initial conditions. The first initial condition is the one above with $c = 1$, and the second initial condition is a soliton solution with $c = 4$. Concatenate this data and identify a PDE model using PDE-FIND. What is the sparsest model that supports the data?

(c) Discuss the two models that are identified depending on the initial data, and explain any differences.

Exercise 7-8. This exercise will explore the connection between DMD and hidden Markov models (HMMs). A Markov model describes the probabilities of transitioning from one of finitely many states to another. Typically this information is encoded in a transition probability matrix $\mathbf{P}$ that defines a probabilistic dynamical system

$$\mathbf{x}_{k+1} = \mathbf{P}\mathbf{x}_k.$$

The vector $\mathbf{x}_k \in \mathbb{R}^n$ is a vector of probabilities[3] of being in one of n states at time-step k. One way to simulate a Markov model forward in time is to evolve the probabilities in $\mathbf{x}$ until they reach a steady state (given by the eigenvector of $\mathbf{P}$ corresponding to unit eigenvalue). Alternatively, it is possible to make an *observation* at each time-step k, whereby the state $\mathbf{x}_{k+1}$ is chosen based on the probability vector $\mathbf{P}\mathbf{x}_k$ so that $\mathbf{x}_{k+1}$ has a 1 in exactly one position and 0s everywhere else. This technically corresponds to a modified system

$$\mathbf{x}_{k+1} = \mathcal{O}(\mathbf{P}\mathbf{x}_k),$$

[3]Note that this is the transpose of the Russian standard notation for Markov models to be consistent with the rest of the book.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

where the operator $\mathcal{O}$ is the observation operator that samples from the probability distribution given by $\mathbf{P}\mathbf{x}_k$.

In this example, consider a Markov process determining a weather model for the transition between sunny, rainy, or cloudy weather, which are the three states in $\mathbf{x} \in \mathbb{R}^3$. The transition probability matrix is given by

$$\mathbf{P} = \begin{bmatrix} 0 & 0.25 & 0.25 \\ 0.40 & 0.50 & 0.25 \\ 0.60 & 0.25 & 0.50 \end{bmatrix}.$$

First, what is the long-time expected probability distribution?

Now, simulate a random instance of this process, using the observation operator $\mathcal{O}$ at every step. Create a data matrix using this process, and identify a DMD model. What is the structure of the model? Does it agree with the transition matrix $\mathbf{P}$? Does it satisfy conservation of probability, meaning that the columns each sum to 1?

Exercise 7-9. This exercise will develop a probabilistic model for the chaotic dynamics in the Lorenz system, following the seminal paper by Kaiser et al. [367] on cluster reduced-order modeling (CROM).

(a) First, generate a long trajectory of the Lorenz system, starting with the standard parameters and integrating until $T = 500$ with a time-step of $\Delta t = 0.005$. Next, use k-means clustering on the data to segment it into $k = 10$ clusters and plot the data, color-coded by which cluster it belongs to.

(b) The CROM approach creates a $k \times k$ Markov model $\mathbf{P}$ for the probability of transitioning from one cluster to another. To compute this transition matrix, begin by creating a $k \times k$ matrix initialized with all zeros. Next, go through the trajectory data, and for every point keep track of which cluster the point belongs to and what cluster the next point belongs to. If the current point belongs to cluster j and the next point belongs to cluster i, add a 1 to the P_{ij} location in the matrix. After all transitions from the entire trajectory have been recorded, normalize each column of $\mathbf{P}$ by the sum of the column, so that all columns add up to 1.

(c) Now simulate the evolution of the CROM model starting with the final data point at $T = 500$ and plot the evolution. How does this compare with a trajectory of the Lorenz system initialized at this same location?

(d) Reproduce Figs. 4 and 5 from the Kaiser et al. [367] paper and explain these results.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

(e) Now, repeat the above with different cluster sizes k. Do the results improve or worsen for fewer clusters? For more clusters?

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Chapter 8

Linear Control Theory

The focus of this book has largely been on characterizing complex systems through dimensionality reduction, sparse sampling, and dynamical systems modeling. However, an overarching goal for many systems is the ability to actively manipulate their behavior for a given engineering objective. The study and practice of manipulating dynamical systems is broadly known as control theory, and it is one of the most successful fields at the interface of applied mathematics and practical engineering. Control theory is inseparable from data science, as it relies on sensor measurements (data) obtained from a system to achieve a given objective. In fact, control theory deals with living data, as successful application modifies the dynamics of the system, thus changing the characteristics of the measurements. Control theory forces the reader to confront reality, as simplifying assumptions and model approximations are tested.

Control theory has helped shape the modern technological and industrial landscape. Examples abound, including cruise control in automobiles, position control in construction equipment, fly-by-wire autopilots in aircraft, industrial automation, packet routing in the Internet, commercial HVAC (heating, ventilation, and air-conditioning) systems, stabilization of rockets, and PID (proportional–integral–derivative) temperature and pressure control in modern espresso machines, to name only a few of the many applications. In the future, control will be increasingly applied to high-dimensional, strongly nonlinear and multi-scale problems, such as turbulence, neuroscience, finance, epidemiology, autonomous robots, and self-driving cars. In these future applications, data-driven modeling and control will be vitally important; this is the subject of Chapters 7 and 10.

This chapter will introduce the key concepts from closed-loop feedback control. The goal is to build intuition for how and when to use feedback control, motivated by practical real-world challenges. Most of the theory will be developed for linear systems, where a wealth of powerful techniques exist [222, 665]. This theory will then be demonstrated on simple and intuitive examples, such as to develop a cruise controller for an automobile or to stabilize

an inverted pendulum on a moving cart. Code will be provided in MATLAB and Python. Historically, control was typically implemented in MATLAB because of the extensive control toolboxes and functionality. However, advanced control is now possible in Python through the Python Control Systems Library (`python-control`), available at `https://python-control.readthedocs.io/`. This toolbox provides, among other functionality, Python wrappers for the same SLICOT optimization libraries [77] that are used in MATLAB's control toolboxes. In all of the Python codes, it is assumed that the following is added to the preamble:

```
from control.matlab import *
import slycot
```

This will make the Python code very similar to, and in some cases identical to, the corresponding MATLAB code. If code is not duplicated for MATLAB and Python, then it may be assumed that the python-control implementation is nearly identical.

Types of Control

There are many ways to manipulate the behavior of a dynamical system, and these control approaches are organized schematically in Fig. 8.1. Passive control does not require input energy, and, when sufficient, it is desirable because of its simplicity, reliability, and low cost. For example, stop signs at a traffic intersection regulate the flow of traffic. Active control requires input energy, and these controllers are divided into two broad categories based on whether or not sensors are used to inform the controller. In the first category, open-loop control relies on a pre-programmed control sequence; in the traffic example, signals may be pre-programmed to regulate traffic dynamically at different times of day. In the second category, active control uses sensors to inform the control law. Disturbance feedforward control measures exogenous disturbances to the system and then feeds this into an open-loop control law; an example of feedforward control would be to pre-emptively change the direction of the flow of traffic near a stadium when a large crowd of people are expected to leave. Finally, the last category is closed-loop feedback control, which will be the main focus of this chapter. Closed-loop control uses sensors to measure the system directly and then shapes the control in response to whether the system is actually achieving the desired goal. Many modern traffic systems have smart traffic lights with a control logic informed by inductive sensors in the roadbed that measure traffic density.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

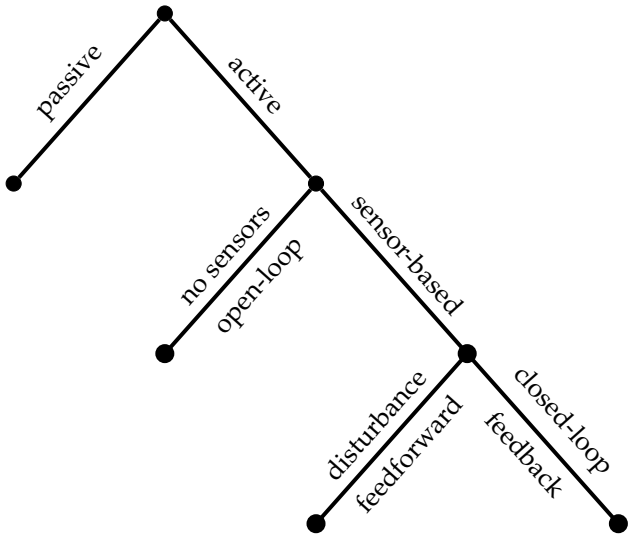


Figure 8.1: Schematic illustrating the various types of control. Most of this chapter will focus on closed-loop feedback control.

8.1 Closed-Loop Feedback Control

The main focus of this chapter is closed-loop feedback control, which is the method of choice for systems with uncertainty, instability, and/or external disturbances. Figure 8.2 depicts the general feedback control framework, where sensor measurements, $\mathbf{y}$, of a system are fed back into a controller, which then decides on an actuation signal, $\mathbf{u}$, to manipulate the dynamics and provide robust performance despite model uncertainty and exogenous disturbances. In all of the examples discussed in this chapter, the vector of exogenous disturbances may be decomposed as $\mathbf{w} = \begin{bmatrix}\mathbf{w}_d^T & \mathbf{w}_n^T & \mathbf{w}_r^T\end{bmatrix}^T$, where $\mathbf{w}_d$ are disturbances to the state of the system, $\mathbf{w}_n$ is measurement noise, and $\mathbf{w}_r$ is a reference trajectory that should be tracked by the closed-loop system.

Mathematically, the system and measurements are typically described by a dynamical system:

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{x} = \mathbf{f}(\mathbf{x}, \mathbf{u}, \mathbf{w}_d), \tag{8.1a}$$

$$\mathbf{y} = \mathbf{g}(\mathbf{x}, \mathbf{u}, \mathbf{w}_n). \tag{8.1b}$$

The goal is to construct a control law,

$$\mathbf{u} = \mathbf{k}(\mathbf{y}, \mathbf{w}_r), \tag{8.2}$$

that minimizes a cost function,

$$J \triangleq J(\mathbf{x}, \mathbf{u}, \mathbf{w}_r). \tag{8.3}$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

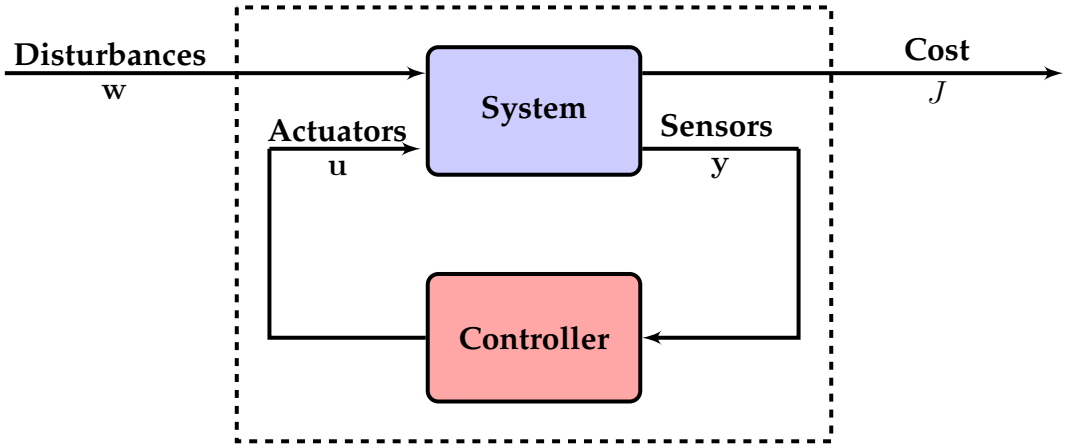


Figure 8.2: Standard framework for feedback control. Measurements of the system, $\mathbf{y}(t)$, are fed back into a controller, which then decides on the appropriate actuation signal $\mathbf{u}(t)$ to control the system. The control law is designed to modify the system dynamics and provide good performance, quantified by the cost J, despite exogenous disturbances and noise in $\mathbf{w}$. The exogenous input $\mathbf{w}$ may also include a reference trajectory $\mathbf{w}_r$ that should be tracked.

Thus, modern control relies heavily on techniques from optimization [101]. In general, the controller in (8.2) will be a dynamical system, rather than a static function of the inputs. For example, the Kalman filter in Section 8.5 dynamically estimates the full state $\mathbf{x}$ from measurements of $\mathbf{u}$ and $\mathbf{y}$. In this case, the control law will become $\mathbf{u} = \mathbf{k}(\mathbf{y}, \hat{\mathbf{x}}, \mathbf{w}_r)$, where $\hat{\mathbf{x}}$ is the full-state estimate.

To motivate the added cost and complexity of sensor-based feedback control, it is helpful to compare with open-loop control. For reference tracking problems, the controller is designed to steer the output of a system towards a desired reference output value $\mathbf{w}_r$, thus minimizing the error $\epsilon = \mathbf{y} - \mathbf{w}_r$. Open-loop control, shown in Fig. 8.3, uses a model of the system to design an actuation signal $\mathbf{u}$ that produces the desired reference output. However, this pre-planned strategy cannot correct for external disturbances to the system and is fundamentally incapable of changing the dynamics. Thus, it is impossible to stabilize an unstable system, such as an inverted pendulum, with open-loop control, since the system model would have to be known perfectly and the system would need to be perfectly isolated from disturbances. Moreover, any model uncertainty will directly contribute to open-loop tracking error.

In contrast, closed-loop feedback control, shown in Fig. 8.4, uses sensor measurements of the system to inform the controller about how the system is actually responding. These sensor measurements provide information about unmodeled dynamics and disturbances that would degrade the performance

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

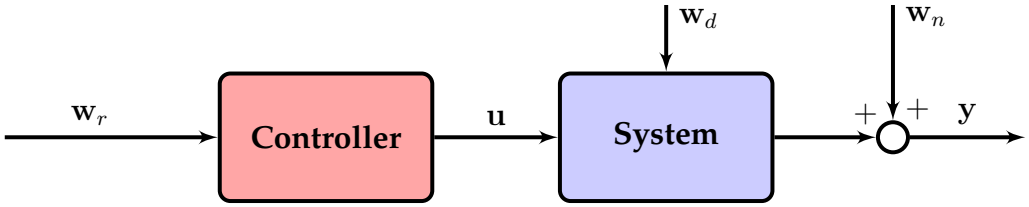


Figure 8.3: Open-loop control diagram. Given a desired reference signal $\mathbf{w}_r$, the open-loop control law constructs a control protocol $\mathbf{u}$ to drive the system based on a model. External disturbances ($\mathbf{w}_d$) and sensor noise ($\mathbf{w}_n$), as well as unmodeled system dynamics and uncertainty, are not accounted for and degrade performance.

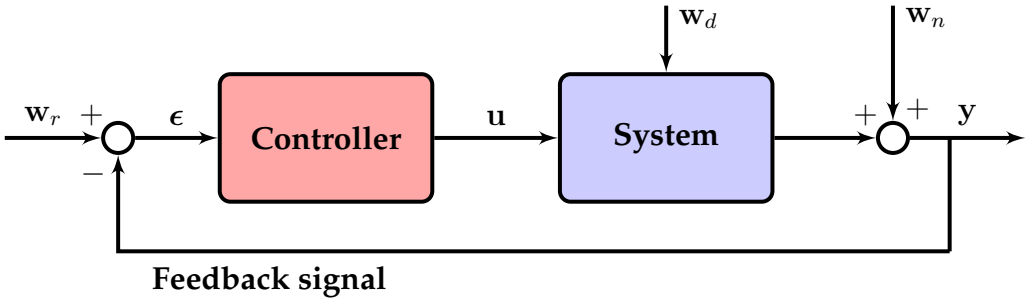


Figure 8.4: Closed-loop feedback control diagram. The sensor signal $\mathbf{y}$ is fed back and subtracted from the reference signal $\mathbf{w}_r$, providing information about how the system is responding to actuation and external disturbances. The controller uses the resulting error $\boldsymbol{\epsilon}$ to determine the correct actuation signal $\mathbf{u}$ for the desired response. Feedback is often able to stabilize unstable dynamics while effectively rejecting disturbances $\mathbf{w}_d$ and attenuating noise $\mathbf{w}_n$.

in open-loop control. Further, with feedback it is often possible to modify and stabilize the dynamics of the closed-loop system, something that is not possible with open-loop control. Thus, closed-loop feedback control is often able to maintain high-performance operation for systems with unstable dynamics, model uncertainty, and external disturbances.

Examples of the Benefits of Feedback Control

To summarize, closed-loop feedback control has several benefits over open-loop control:

- It may be possible to stabilize an unstable system.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

- It may be possible to compensate for external disturbances.
- It may be possible to correct for unmodeled dynamics and model uncertainty.

These issues are illustrated in the following two simple examples.

Inverted Pendulum. Consider the unstable inverted pendulum equations, which will be derived later in Section 8.2. The linearized equations are

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ g/L & d \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u, \tag{8.4}$$

where $x_1 = \theta$, $x_2 = \dot{\theta}$, u is a torque applied to the pendulum arm, g is gravitational acceleration, L is the length of the pendulum arm, and d is damping. We may write this system in standard form as

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{x} = \mathbf{A}\mathbf{x} + \mathbf{B}u.$$

If we choose constants so that the natural frequency is $\omega_n = \sqrt{g/L} = 1$ and $d = 0$, then the system has eigenvalues $\lambda = \pm 1$, corresponding to an unstable saddle-type fixed point.

No open-loop control strategy can change the dynamics of the system, given by the eigenvalues of $\mathbf{A}$. However, with full-state feedback control, given by $u = -\mathbf{K}\mathbf{x}$, the closed-loop system becomes

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{x} = \mathbf{A}\mathbf{x} + \mathbf{B}u = (\mathbf{A} - \mathbf{B}\mathbf{K})\mathbf{x}.$$

Choosing $\mathbf{K} = \begin{bmatrix} 4 & 4 \end{bmatrix}$, corresponding to a control law $u = -4x_1 - 4x_2 = -4\theta - 4\dot{\theta}$, the closed-loop system $(\mathbf{A} - \mathbf{B}\mathbf{K})$ has stable eigenvalues $\lambda = -1$ and $\lambda = -3$.

Determining when it is possible to change the eigenvalues of the closed-loop system, and determining the appropriate control law $\mathbf{K}$ to achieve this, will be the subject of future sections.

Cruise Control. To appreciate the ability of closed-loop control to compensate for unmodeled dynamics and disturbances, we will consider a simple model of cruise control in an automobile. Let u be the rate of fuel fed into the engine, and let y be the car's speed. Neglecting transients, a crude model[1] is

$$y = u. \tag{8.5}$$

[1] A more realistic model would have acceleration dynamics, so that $\dot{x} = -x + u$ and $y = x$.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Thus, if we double the gas input, we double the automobile's speed.

Based on this model, we may design an open-loop cruise controller to track a reference speed w_r by simply commanding an input of $u = w_r$. However, an incorrect automobile model (i.e., in actuality $y = 2u$), or external disturbances, such as rolling hills (i.e., if $y = u + \sin(t)$), are not accounted for in the simple open-loop design.

In contrast, a closed-loop control law, based on measurements of the speed, is able to compensate for unmodeled dynamics and disturbances. Consider the closed-loop control law $u = K(w_r - y)$, so that gas is increased when the measured velocity is too low, and decreased when it is too high. Then if the dynamics are actually $y = 2u$ instead of $y = u$, the open-loop system will have 50% steady-state tracking error, while the performance of the closed-loop system can be significantly improved for large K:

$$y = 2K(w_r - y) \quad \Longrightarrow \quad (1 + 2K)y = 2Kw_r \quad \Longrightarrow \quad y = \frac{2K}{1 + 2K} w_r. \tag{8.6}$$

For $K = 50$, the closed-loop system only has 1% steady-state tracking error. Similarly, an added disturbance w_d will be attenuated by a factor of $1/(2K + 1)$.

As a concrete example, consider a reference tracking problem with a desired reference speed of 60 mph miles per hour). The model is $y = u$, and the true system is $y = 0.5u$. In addition, there is a disturbance in the form of rolling hills that increase and decrease the speed by ± 10 mph at a frequency of 0.5 Hz. An open-loop controller is compared with a closed-loop proportional controller with $K = 50$ in Fig. 8.5 and Code 8.1. Although the closed-loop controller has significantly better performance, we will see later that a large proportional gain may come at the cost of robustness. Adding an integral term will improve performance.

Code 8.1: [MATLAB] Compare open-loop and closed-loop cruise control.

```
t = 0:.01:10;              % time

wr = 60*ones(size(t));     % reference speed
d = 10*sin(pi*t);          % disturbance

aModel = 1;                % y = aModel*u
aTrue = .5;                % y = aTrue*u

uOL = wr/aModel;           % Open-loop u based on model
yOL = aTrue*uOL + d;       % Open-loop response

K = 50;                    % control gain, u=K(wr-y);
yCL = aTrue*K/(1+aTrue*K)*wr + d/(1+aTrue*K);
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

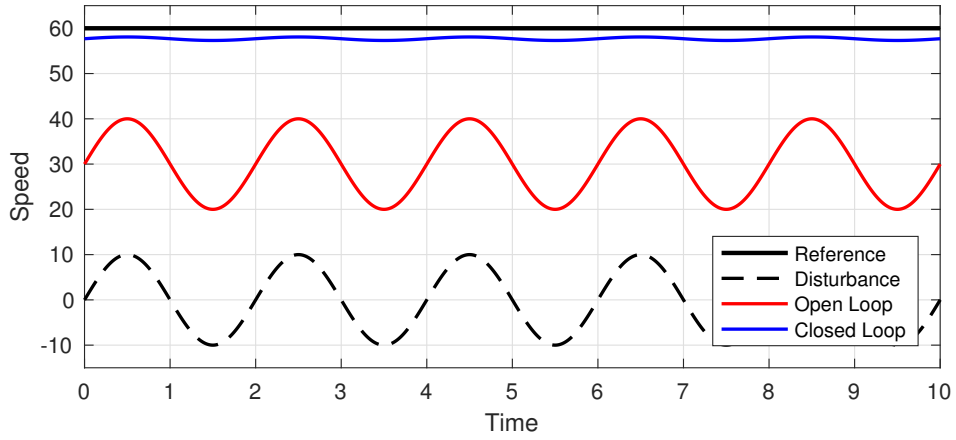


Figure 8.5: Open-loop versus closed-loop cruise control.

Code 8.1: [Python] Compare open-loop and closed-loop cruise control.

```
t = np.arange(0,10,0.01)    # time

wr = 60 * np.ones_like(t)   # reference speed
d = 10*np.sin(np.pi*t)      # disturbance

aModel = 1                  # y = aModel*u
aTrue = 0.5                 # y = aTrue*u

uOL = wr/aModel             # Open-loop u based on model
yOL = aTrue*uOL + d         # Open-loop response

K = 50                      # control gain, u=K(wr-y)
yCL = (aTrue*K/(1+aTrue*K))*wr + d/(1+aTrue*K)
```

8.2 Linear Time-Invariant Systems

The most complete theory of control has been developed for linear systems [30, 222, 665]. Linear systems are generally obtained by linearizing a nonlinear system about a fixed point or a periodic orbit. However, instability may quickly take a trajectory far away from the fixed point. Fortunately, an effective stabilizing controller will keep the state of the system in a small neighborhood of the fixed point where the linear approximation is valid. For example, in the case of the inverted pendulum, feedback control may keep the pendulum stabilized in the vertical position where the dynamics behave linearly.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Linearization of Nonlinear Dynamics

Given a nonlinear input–output system

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{x} = \mathbf{f}(\mathbf{x}, \mathbf{u}), \tag{8.7a}$$

$$\mathbf{y} = \mathbf{g}(\mathbf{x}, \mathbf{u}), \tag{8.7b}$$

it is possible to linearize the dynamics near a fixed point $(\bar{\mathbf{x}}, \bar{\mathbf{u}})$ where $\mathbf{f}(\bar{\mathbf{x}}, \bar{\mathbf{u}}) = \mathbf{0}$. For small $\Delta\mathbf{x} = \mathbf{x} - \bar{\mathbf{x}}$ and $\Delta\mathbf{u} = \mathbf{u} - \bar{\mathbf{u}}$, the dynamics $\mathbf{f}$ may be expanded in a Taylor series about the point $(\bar{\mathbf{x}}, \bar{\mathbf{u}})$ as

$$\mathbf{f}(\bar{\mathbf{x}} + \Delta\mathbf{x}, \bar{\mathbf{u}} + \Delta\mathbf{u}) = \mathbf{f}(\bar{\mathbf{x}}, \bar{\mathbf{u}}) + \underbrace{\left.\frac{\mathrm{d}\mathbf{f}}{\mathrm{d}\mathbf{x}}\right|_{(\bar{\mathbf{x}},\bar{\mathbf{u}})}}_{\mathbf{A}} \cdot \Delta\mathbf{x} + \underbrace{\left.\frac{\mathrm{d}\mathbf{f}}{\mathrm{d}\mathbf{u}}\right|_{(\bar{\mathbf{x}},\bar{\mathbf{u}})}}_{\mathbf{B}} \cdot \Delta\mathbf{u} + \cdots . \tag{8.8}$$

Similarly, the output equation $\mathbf{g}$ may be expanded as

$$\mathbf{g}(\bar{\mathbf{x}} + \Delta\mathbf{x}, \bar{\mathbf{u}} + \Delta\mathbf{u}) = \mathbf{g}(\bar{\mathbf{x}}, \bar{\mathbf{u}}) + \underbrace{\left.\frac{\mathrm{d}\mathbf{g}}{\mathrm{d}\mathbf{x}}\right|_{(\bar{\mathbf{x}},\bar{\mathbf{u}})}}_{\mathbf{C}} \cdot \Delta\mathbf{x} + \underbrace{\left.\frac{\mathrm{d}\mathbf{g}}{\mathrm{d}\mathbf{u}}\right|_{(\bar{\mathbf{x}},\bar{\mathbf{u}})}}_{\mathbf{D}} \cdot \Delta\mathbf{u} + \cdots . \tag{8.9}$$

For small displacements around the fixed point, the higher-order terms are negligibly small. Dropping the Δ and shifting to a coordinate system where $\bar{\mathbf{x}}$, $\bar{\mathbf{u}}$, and $\bar{\mathbf{y}}$ are at the origin, the linearized dynamics may be written as

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{x} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}, \tag{8.10a}$$

$$\mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u}. \tag{8.10b}$$

Note that we have neglected the disturbance and noise inputs, $\mathbf{w}_d$ and $\mathbf{w}_n$, respectively; these will be added back in the discussion on Kalman filtering in Section 8.5.

Unforced Linear System

In the absence of control (i.e., $\mathbf{u} = \mathbf{0}$), and with measurements of the full state (i.e., $\mathbf{y} = \mathbf{x}$), the dynamical system in (8.10) becomes

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{x} = \mathbf{A}\mathbf{x}. \tag{8.11}$$

The solution $\mathbf{x}(t)$ is given by

$$\mathbf{x}(t) = e^{\mathbf{A}t}\mathbf{x}(0), \tag{8.12}$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

where the matrix exponential is defined by

$$e^{\mathbf{A}t} = \mathbf{I} + \mathbf{A}t + \frac{\mathbf{A}^2t^2}{2!} + \frac{\mathbf{A}^3t^3}{3!} + \cdots . \tag{8.13}$$

The solution in (8.12) is determined entirely by the eigenvalues and eigenvectors of the matrix $\mathbf{A}$. Consider the eigendecomposition of $\mathbf{A}$:

$$\mathbf{AT} = \mathbf{T}\boldsymbol{\Lambda}. \tag{8.14}$$

In the simplest case, $\boldsymbol{\Lambda}$ is a diagonal matrix of distinct eigenvalues and $\mathbf{T}$ is a matrix whose columns are the corresponding linearly independent eigenvectors of $\mathbf{A}$. For repeated eigenvalues, $\boldsymbol{\Lambda}$ may be written in Jordan form, with entries above the diagonal for degenerate eigenvalues of multiplicity ≥ 2; the corresponding columns of $\mathbf{T}$ will be generalized eigenvectors.

In either case, it is easier to compute the matrix exponential $e^{\boldsymbol{\Lambda}t}$ than $e^{\mathbf{A}t}$. For diagonal $\boldsymbol{\Lambda}$, the matrix exponential is given by

$$e^{\boldsymbol{\Lambda}t} = \begin{bmatrix} e^{\lambda_1 t} & 0 & \cdots & 0 \\ 0 & e^{\lambda_2 t} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & e^{\lambda_n t} \end{bmatrix}. \tag{8.15}$$

In the case of a non-trivial Jordan block in $\boldsymbol{\Lambda}$ with entries above the diagonal, simple extensions exist related to nilpotent matrices (for details, see Perko [562]).

Rearranging the terms in (8.14), we find that it is simple to represent powers of $\mathbf{A}$ in terms of the eigenvectors and eigenvalues:

$$\mathbf{A} = \mathbf{T}\boldsymbol{\Lambda}\mathbf{T}^{-1}, \tag{8.16a}$$
$$\mathbf{A}^2 = (\mathbf{T}\boldsymbol{\Lambda}\mathbf{T}^{-1})(\mathbf{T}\boldsymbol{\Lambda}\mathbf{T}^{-1}) = \mathbf{T}\boldsymbol{\Lambda}^2\mathbf{T}^{-1}, \tag{8.16b}$$
$$\vdots$$
$$\mathbf{A}^k = (\mathbf{T}\boldsymbol{\Lambda}\mathbf{T}^{-1})(\mathbf{T}\boldsymbol{\Lambda}\mathbf{T}^{-1})\cdots(\mathbf{T}\boldsymbol{\Lambda}\mathbf{T}^{-1}) = \mathbf{T}\boldsymbol{\Lambda}^k\mathbf{T}^{-1}. \tag{8.16c}$$

Finally, substituting these expressions into (8.13) yields

$$e^{\mathbf{A}t} = e^{\mathbf{T}\boldsymbol{\Lambda}\mathbf{T}^{-1}t} = \mathbf{T}\mathbf{T}^{-1} + \mathbf{T}\boldsymbol{\Lambda}\mathbf{T}^{-1}t + \frac{\mathbf{T}\boldsymbol{\Lambda}^2\mathbf{T}^{-1}t^2}{2!} + \frac{\mathbf{T}\boldsymbol{\Lambda}^3\mathbf{T}^{-1}t^3}{3!} + \cdots \tag{8.17a}$$
$$= \mathbf{T}\left[\mathbf{I} + \boldsymbol{\Lambda}t + \frac{\boldsymbol{\Lambda}^2t^2}{2!} + \frac{\boldsymbol{\Lambda}^3t^3}{3!} + \cdots\right]\mathbf{T}^{-1} \tag{8.17b}$$
$$= \mathbf{T}e^{\boldsymbol{\Lambda}t}\mathbf{T}^{-1}. \tag{8.17c}$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Thus, we see that it is possible to compute the matrix exponential efficiently in terms of the eigendecomposition of $\mathbf{A}$. Moreover, the matrix of eigenvectors $\mathbf{T}$ defines a change of coordinates that dramatically simplifies the dynamics:

$$\mathbf{x} = \mathbf{T}\mathbf{z} \quad \Longrightarrow \quad \dot{\mathbf{z}} = \mathbf{T}^{-1}\dot{\mathbf{x}} = \mathbf{T}^{-1}\mathbf{A}\mathbf{x} = \mathbf{T}^{-1}\mathbf{A}\mathbf{T}\mathbf{z} \quad \Longrightarrow \quad \dot{\mathbf{z}} = \mathbf{\Lambda}\mathbf{z}. \tag{8.18}$$

In other words, changing to eigenvector coordinates, the dynamics become diagonal. Combining (8.12) with (8.17c), it is possible to write the solution $\mathbf{x}(t)$ as

$$\mathbf{x}(t) = \underbrace{\mathbf{T}\underbrace{e^{\mathbf{\Lambda}t}\underbrace{\mathbf{T}^{-1}\mathbf{x}(0)}_{\mathbf{z}(0)}}_{\mathbf{z}(t)}}_{\mathbf{x}(t)}. \tag{8.19}$$

In the first step, $\mathbf{T}^{-1}$ maps the initial condition in physical coordinates, $\mathbf{x}(0)$, into eigenvector coordinates, $\mathbf{z}(0)$. The next step advances these initial conditions using the diagonal update $e^{\mathbf{\Lambda}t}$, which is considerably simpler in eigenvector coordinates $\mathbf{z}$. Finally, multiplying by $\mathbf{T}$ maps $\mathbf{z}(t)$ back to physical coordinates, $\mathbf{x}(t)$.

In addition to making it possible to compute the matrix exponential, and hence the solution $\mathbf{x}(t)$, the eigendecomposition of $\mathbf{A}$ is even more useful to understand the dynamics and stability of the system. We see from (8.19) that the only time-varying portion of the solution is $e^{\mathbf{\Lambda}t}$. In general, these eigenvalues $\lambda = a + ib$ may be complex numbers, so that the solutions are given by $e^{\lambda t} = e^{at}(\cos(bt) + i\sin(bt))$. Thus, if all of the eigenvalues λ_k have negative real part (i.e., $\text{Re}(\lambda) = a < 0$), then the system is stable, and solutions all decay to $\mathbf{x} = \mathbf{0}$ as $t \to \infty$. However, if even a single eigenvalue has positive real part, then the system is unstable and will diverge from the fixed point along the corresponding unstable eigenvector direction. Any random initial condition is likely to have a component in this unstable direction, and, moreover, disturbances will likely excite all eigenvectors of the system.

Forced Linear System

With forcing, and for zero initial condition, $\mathbf{x}(0) = \mathbf{0}$, the solution to (8.10a) is

$$\mathbf{x}(t) = \int_0^t e^{\mathbf{A}(t-\tau)}\mathbf{B}\mathbf{u}(\tau)\,\mathrm{d}\tau \triangleq e^{\mathbf{A}t}\mathbf{B} * \mathbf{u}(t). \tag{8.20}$$

The control input $\mathbf{u}(t)$ is convolved with the kernel $e^{\mathbf{A}t}\mathbf{B}$. With an output $\mathbf{y} = \mathbf{C}\mathbf{x}$, we have $\mathbf{y}(t) = \mathbf{C}e^{\mathbf{A}t}\mathbf{B} * \mathbf{u}(t)$. This convolution is illustrated in Fig. 8.6 for a single-input, single-output (SISO) system in terms of the impulse response $g(t) = \mathbf{C}e^{\mathbf{A}t}\mathbf{B} = \int_0^t \mathbf{C}e^{\mathbf{A}(t-\tau)}\mathbf{B}\delta(\tau)\,\mathrm{d}\tau$ given a Dirac delta input $u(t) = \delta(t)$.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

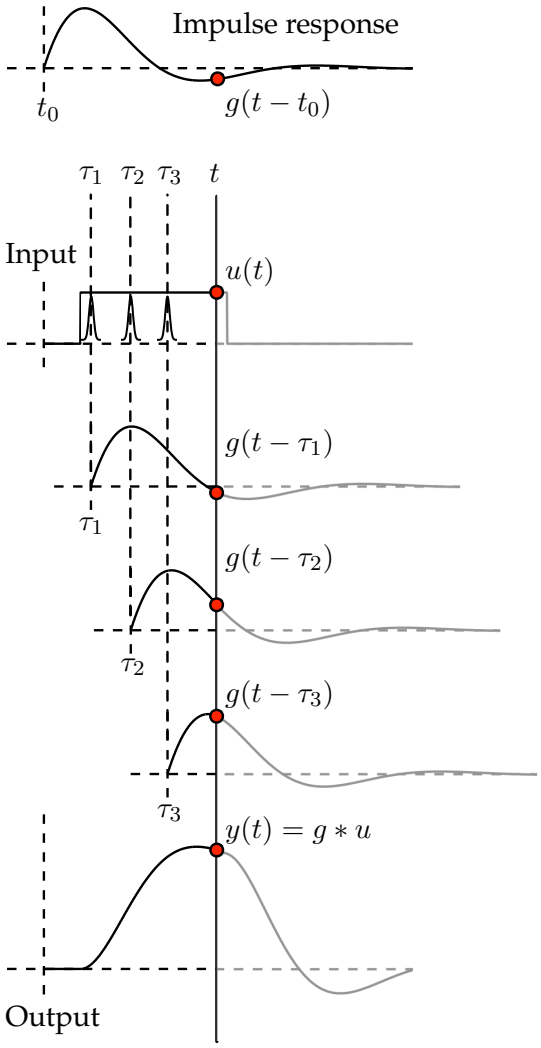


Figure 8.6: Convolution for a single-input, single-output (SISO) system.

Discrete-Time Systems

In many real-world applications, systems are sampled at discrete instants in time. Thus, digital control systems are typically formulated in terms of discrete-time dynamical systems:

$$\mathbf{x}_{k+1} = \mathbf{A}_d\mathbf{x}_k + \mathbf{B}_d\mathbf{u}_k, \tag{8.21a}$$

$$\mathbf{y}_k = \mathbf{C}_d\mathbf{x}_k + \mathbf{D}_d\mathbf{u}_k, \tag{8.21b}$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

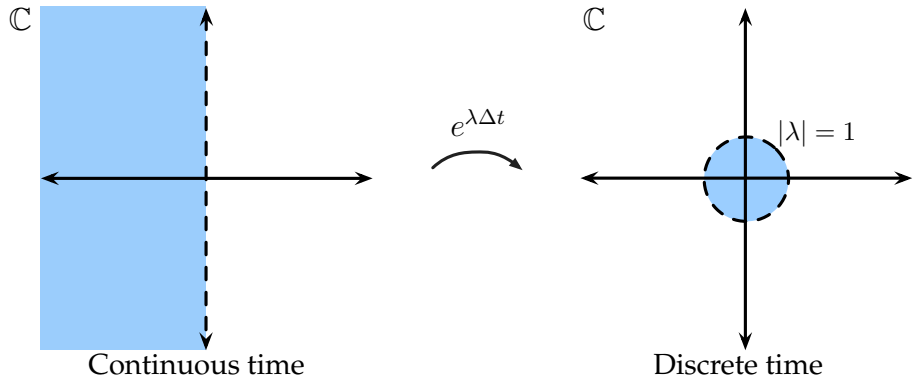


Figure 8.7: The matrix exponential defines a conformal map on the complex plane, mapping stable eigenvalues in the left half-plane into eigenvalues inside the unit circle.

where $\mathbf{x}_k = \mathbf{x}(k\Delta t)$. The system matrices in (8.21) can be obtained from the continuous-time system in (8.10) as

$$\mathbf{A}_d = e^{\mathbf{A}\Delta t}, \tag{8.22a}$$

$$\mathbf{B}_d = \int_0^{\Delta t} e^{\mathbf{A}\tau}\mathbf{B}\,\mathrm{d}\tau, \tag{8.22b}$$

$$\mathbf{C}_d = \mathbf{C}, \tag{8.22c}$$

$$\mathbf{D}_d = \mathbf{D}. \tag{8.22d}$$

The stability of the discrete-time system in (8.21) is still determined by the eigenvalues of $\mathbf{A}_d$, although now a system is stable if and only if all discrete-time eigenvalues are inside the unit circle in the complex plane. Thus, $\exp(\mathbf{A}\Delta t)$ defines a conformal mapping on the complex plane from continuous time to discrete time, where eigenvalues in the left half-plane map to eigenvalues inside the unit circle.

Example: Inverted Pendulum

Consider the inverted pendulum in Fig. 8.8 with a torque input u at the base. The equation of motion, derived using the Euler–Lagrange equations,[2] is

$$\ddot{\theta} = -\frac{g}{L}\sin(\theta) + u. \tag{8.23}$$

[2]The Lagrangian is $\mathcal{L} = (m/2)L^2\dot{\theta}^2 - mgL\cos(\theta)$, and the Euler–Lagrange equation is $(\mathrm{d}/\mathrm{d}t)\partial\mathcal{L}/\partial\dot{\theta} - \partial\mathcal{L}/\partial\theta = \tau$, where τ is the input torque.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

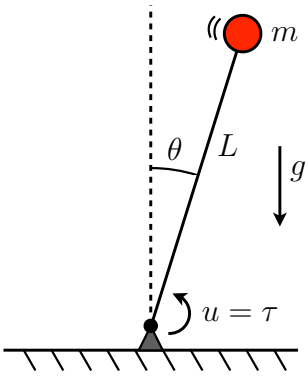


Figure 8.8: Schematic of inverted pendulum system.

Introducing the state $\mathbf{x}$, given by the angular position and velocity, we can write this second-order differential equation as a system of first-order equations:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} \theta \\ \dot{\theta} \end{bmatrix} \quad \Longrightarrow \quad \frac{\mathrm{d}}{\mathrm{d}t} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} x_2 \\ -(g/L)\sin(x_1) + u \end{bmatrix}. \tag{8.24}$$

Taking the Jacobian of $\mathbf{f}(\mathbf{x}, \mathbf{u})$ yields

$$\frac{\mathrm{d}\mathbf{f}}{\mathrm{d}\mathbf{x}} = \begin{bmatrix} 0 & 1 \\ -(g/L)\cos(x_1) & 0 \end{bmatrix}, \quad \frac{\mathrm{d}\mathbf{f}}{\mathrm{d}\mathbf{u}} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}. \tag{8.25}$$

Linearizing at the pendulum-up ($x_1 = \pi$, $x_2 = 0$) and pendulum-down ($x_1 = 0$, $x_2 = 0$) equilibria gives

$$\underbrace{\frac{\mathrm{d}}{\mathrm{d}t} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ g/L & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u}_{\text{pendulum up, } \lambda = \pm\sqrt{g/L}} \quad \text{and} \quad \underbrace{\frac{\mathrm{d}}{\mathrm{d}t} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -g/L & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u}_{\text{pendulum down, } \lambda = \pm i\sqrt{g/L}}.$$

Thus, we see that the down position is a stable center with eigenvalues $\lambda = \pm i\sqrt{g/L}$ corresponding to oscillations at a natural frequency of $\sqrt{g/L}$. The pendulum-up position is an unstable saddle with eigenvalues $\lambda = \pm\sqrt{g/L}$.

8.3 Controllability and Observability

A natural question arises in linear control theory: To what extent can closed-loop feedback $\mathbf{u} = -\mathbf{K}\mathbf{x}$ manipulate the behavior of the system in (8.10a)? We already saw in Section 8.1 that it was possible to modify the eigenvalues of the unstable inverted pendulum system via closed-loop feedback, resulting in a new system matrix $(\mathbf{A} - \mathbf{B}\mathbf{K})$ with stable eigenvalues. This section will provide concrete conditions on when and how the system dynamics may be manipulated through feedback control. The dual question, of when it is possible to estimate the full state $\mathbf{x}$ from measurements $\mathbf{y}$, will also be addressed.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Controllability

The ability to design the eigenvalues of the closed-loop system with the choice of $\mathbf{K}$ relies on the system in (8.10a) being *controllable*. The controllability of a linear system is determined entirely by the column space of the *controllability* matrix $\mathcal{C}$:

$$\mathcal{C} = \begin{bmatrix} \mathbf{B} & \mathbf{AB} & \mathbf{A}^2\mathbf{B} & \cdots & \mathbf{A}^{n-1}\mathbf{B} \end{bmatrix}. \tag{8.26}$$

If the matrix $\mathcal{C}$ has n linearly independent columns, so that it spans all of $\mathbb{R}^n$, then the system in (8.10a) is controllable. The span of the columns of the controllability matrix $\mathcal{C}$ forms a Krylov subspace that determines which state vector directions in $\mathbb{R}^n$ may be manipulated with control. Thus, in addition to controllability implying arbitrary eigenvalue placement, it also implies that any state $\boldsymbol{\xi} \in \mathbb{R}^n$ is reachable in a finite time with some actuation signal $\mathbf{u}(t)$.

The following three conditions are equivalent:

(a) *Controllability*. The span of $\mathcal{C}$ is $\mathbb{R}^n$. The matrix $\mathcal{C}$ may be generated by

```
>> ctrb(A,B)
```

and the rank may be tested to see if it is equal to n by

```
>> rank(ctrb(A,B))
```

In Python, the rank is computed by

```
>>> numpy.linalg.matrix_rank(ctrb(A,B))
```

(b) *Arbitrary eigenvalue placement*. It is possible to design the eigenvalues of the closed-loop system through choice of feedback $\mathbf{u} = -\mathbf{K}\mathbf{x}$:

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{x} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} = (\mathbf{A} - \mathbf{B}\mathbf{K})\mathbf{x}. \tag{8.27}$$

Given a set of desired eigenvalues, the gain $\mathbf{K}$ can be determined by

```
>> K = place(A,B,neweigs)
```

Designing $\mathbf{K}$ for the best performance will be discussed in Section 8.4.

(c) *Reachability of* $\mathbb{R}^n$. It is possible to steer the system to any arbitrary state $\mathbf{x}(t) = \boldsymbol{\xi} \in \mathbb{R}^n$ in a finite time with some actuation signal $\mathbf{u}(t)$.

Note that reachability also applies to open-loop systems. In particular, if a direction $\boldsymbol{\xi}$ is not in the span of $\mathcal{C}$, then it is impossible for control to push in this direction in either open-loop or closed-loop systems.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Examples. The notion of controllability is more easily understood by investigating a few simple examples. First, consider the following system:

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u \quad \Longrightarrow \quad \mathcal{C} = \begin{bmatrix} 0 & 0 \\ 1 & 2 \end{bmatrix}. \tag{8.28}$$

This system is not controllable, because the controllability matrix $\mathcal{C}$ consists of two linearly dependent vectors and does not span $\mathbb{R}^2$. Even before checking the rank of the controllability matrix, it is easy to see that the system will not be controllable since the states x_1 and x_2 are completely decoupled and the actuation input u only affects the second state.

Modifying this example to include two actuation inputs makes the system controllable by increasing the control authority:

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \quad \Longrightarrow \quad \mathcal{C} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 2 \end{bmatrix}. \tag{8.29}$$

This *fully actuated* system is clearly controllable because x_1 and x_2 may be independently controlled with u_1 and u_2. The controllability of this system is confirmed by checking that the columns of $\mathcal{C}$ do span $\mathbb{R}^2$.

The most interesting cases are less obvious than these two examples. Consider the system

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & 2 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u \quad \Longrightarrow \quad \mathcal{C} = \begin{bmatrix} 0 & 1 \\ 1 & 2 \end{bmatrix}. \tag{8.30}$$

This two-state system is controllable with a single actuation input because the states x_1 and x_2 are now coupled through the dynamics. Similarly,

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} u \quad \Longrightarrow \quad \mathcal{C} = \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix} \tag{8.31}$$

is controllable even though the dynamics of x_1 and x_2 are decoupled, because the actuator $\mathbf{B} = \begin{bmatrix} 1 & 1 \end{bmatrix}^T$ is able to simultaneously affect both states and they have different timescales.

We will see in Section 8.3 that controllability is intimately related to the alignment of the columns of $\mathbf{B}$ with the eigenvector directions of $\mathbf{A}$.

Observability

Mathematically, observability of the system in (8.10) is nearly identical to controllability, although the physical interpretation differs somewhat. A system is *observable* if it is possible to estimate any state $\boldsymbol{\xi} \in \mathbb{R}^n$ from a time history of the measurements $\mathbf{y}(t)$.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Again, the observability of a system is entirely determined by the row space of the *observability* matrix $\mathcal{O}$:

$$\mathcal{O} = \begin{bmatrix} \mathbf{C} \\ \mathbf{CA} \\ \mathbf{CA}^2 \\ \vdots \\ \mathbf{CA}^{n-1} \end{bmatrix}. \tag{8.32}$$

In particular, if the rows of the matrix $\mathcal{O}$ span $\mathbb{R}^n$, then it is possible to estimate any full-dimensional state $\mathbf{x} \in \mathbb{R}^n$ from the time history of $\mathbf{y}(t)$. The matrix $\mathcal{O}$ may be generated by

```
>> obsv(A,C)
```

The motivation for full-state estimation is relatively straightforward. We have already seen that, with full-state feedback, $\mathbf{u} = -\mathbf{K}\mathbf{x}$, it is possible to modify the behavior of a controllable system. However, if full-state measurements of $\mathbf{x}$ are not available, it is necessary to *estimate* $\mathbf{x}$ from the measurements. This is possible when the system is observable. In Section 8.5, we will see that it is possible to design an observer dynamical system to estimate the full state from noisy measurements. As in the case of a controllable system, if a system is observable, it is possible to design the eigenvalues of the estimator dynamical system to have desirable characteristics, such as fast estimation and effective noise attenuation.

Interestingly, the observability criterion is mathematically the dual of the controllability criterion. In fact, the observability matrix is the transpose of the controllability matrix for the pair $(\mathbf{A}^T, \mathbf{C}^T)$:

```
>> O = ctrb(A',C')'; % 'obsv' is dual of 'crtb'
```

The PBH Test for Controllability

There are many tests to determine whether or not a system is controllable. One of the most useful and illuminating is the Popov–Belevitch–Hautus (PBH) test. The PBH test states that the pair $(\mathbf{A}, \mathbf{B})$ is controllable if and only if the column rank of the matrix $\begin{bmatrix}(\mathbf{A} - \lambda\mathbf{I}) & \mathbf{B}\end{bmatrix}$ is equal to n for all $\lambda \in \mathbb{C}$. This test is particularly fascinating because it connects controllability[3] to a relationship between the columns of $\mathbf{B}$ and the eigenspace of $\mathbf{A}$.

First, the PBH test only needs to be checked at λ that are eigenvalues of $\mathbf{A}$, since the rank of $\mathbf{A}-\lambda\mathbf{I}$ is equal to n except when λ is an eigenvalue of $\mathbf{A}$. In fact,

[3]There is an equivalent PBH test for observability that states that $\begin{bmatrix}(\mathbf{A} - \lambda\mathbf{I}) \\ \mathbf{C}\end{bmatrix}$ must have row rank n for all $\lambda \in \mathbb{C}$ for the system to be observable.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

the characteristic equation $\det(\mathbf{A}-\lambda\mathbf{I})=0$ is used to determine the eigenvalues of $\mathbf{A}$ as exactly those values where the matrix $\mathbf{A}-\lambda\mathbf{I}$ becomes rank-deficient or degenerate.

Now, given that $(\mathbf{A}-\lambda\mathbf{I})$ is only rank-deficient for eigenvalues λ, it also follows that the null space, or kernel, of $\mathbf{A}-\lambda\mathbf{I}$ is given by the span of the eigenvectors corresponding to that particular eigenvalue. Thus, for $\begin{bmatrix}(\mathbf{A}-\lambda\mathbf{I}) & \mathbf{B}\end{bmatrix}$ to have rank n, the columns in $\mathbf{B}$ must have some component in each of the eigenvector directions associated with $\mathbf{A}$ to complement the null space of $\mathbf{A}-\lambda\mathbf{I}$.

If $\mathbf{A}$ has n distinct eigenvalues, then the system will be controllable with a single actuation input, since the matrix $\mathbf{A}-\lambda\mathbf{I}$ will have at most one eigenvector direction in the null space. In particular, we may choose $\mathbf{B}$ as the sum of all of the n linearly independent eigenvectors, and it will be guaranteed to have some component in each direction. It is also interesting to note that if $\mathbf{B}$ is a random vector (>>**B=randn(n,1);**), then $(\mathbf{A},\mathbf{B})$ will be controllable with high probability, since it will be exceedingly unlikely that $\mathbf{B}$ will be randomly chosen so that it has zero contribution from any given eigenvector.

If there are degenerate eigenvalues with multiplicity ≥ 2, so that the null space of $\mathbf{A}-\lambda\mathbf{I}$ is multi-dimensional, then the actuation input must have as many degrees of freedom. In other words, the only time that multiple actuators (columns of $\mathbf{B}$) are strictly required is for systems that have degenerate eigenvalues. However, if a system is highly non-normal, it may be helpful to have multiple actuators in practice for better control authority. Such non-normal systems are characterized by large transient growth due to destructive interference between nearly parallel eigenvectors, often with similar eigenvalues.

The Cayley–Hamilton Theorem and Reachability

To provide insight into the relationship between the controllability of the pair $(\mathbf{A},\mathbf{B})$ and the reachability of any vector $\boldsymbol{\xi}\in\mathbb{R}^n$ via the actuation input $\mathbf{u}(t)$, we will leverage the Cayley–Hamilton theorem. This is a gem of linear algebra that provides an elegant way to represent solutions of $\dot{\mathbf{x}}=\mathbf{A}\mathbf{x}$ in terms of a finite sum of powers of $\mathbf{A}$, rather than the infinite sum required for the matrix exponential in (8.13).

The Cayley–Hamilton theorem states that every matrix $\mathbf{A}$ satisfies its own characteristic (eigenvalue) equation, $\det(\mathbf{A}-\lambda\mathbf{I})=0$:

$$\det(\mathbf{A}-\lambda\mathbf{I})=\lambda^n+a_{n-1}\lambda^{n-1}+\cdots+a_2\lambda^2+a_1\lambda+a_0=0 \tag{8.33a}$$

$$\Longrightarrow \quad \mathbf{A}^n+a_{n-1}\mathbf{A}^{n-1}+\cdots+a_2\mathbf{A}^2+a_1\mathbf{A}+a_0\mathbf{I}=\mathbf{0}. \tag{8.33b}$$

Although this is relatively simple to state, it has profound consequences. In particular, it is possible to express $\mathbf{A}^n$ as a linear combination of smaller powers of $\mathbf{A}$:

$$\mathbf{A}^n=-a_0\mathbf{I}-a_1\mathbf{A}-a_2\mathbf{A}^2-\cdots-a_{n-1}\mathbf{A}^{n-1}. \tag{8.34}$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

It is straightforward to see that this also implies that any higher power $\mathbf{A}^{k\geq n}$ may also be expressed as a sum of the matrices $\{\mathbf{I}, \mathbf{A}, \dots, \mathbf{A}^{n-1}\}$:

$$\mathbf{A}^{k\geq n} = \sum_{j=0}^{n-1} \alpha_j \mathbf{A}^j. \tag{8.35}$$

Thus, it is possible to express the infinite sum in the exponential $e^{\mathbf{A}t}$ as

$$\begin{aligned} e^{\mathbf{A}t} &= \mathbf{I} + \mathbf{A}t + \frac{\mathbf{A}^2t^2}{2!} + \cdots && \text{(8.36a)} \\ &= \beta_0(t)\mathbf{I} + \beta_1(t)\mathbf{A} + \beta_2(t)\mathbf{A}^2 + \cdots + \beta_{n-1}(t)\mathbf{A}^{n-1}. && \text{(8.36b)} \end{aligned}$$

We are now equipped to see how controllability relates to the reachability of an arbitrary vector $\boldsymbol{\xi} \in \mathbb{R}^n$. From (8.20), we see that a state $\boldsymbol{\xi}$ is reachable if there is some $\mathbf{u}(t)$ so that

$$\boldsymbol{\xi} = \int_0^t e^{\mathbf{A}(t-\tau)}\mathbf{B}\mathbf{u}(\tau)\,\mathrm{d}\tau. \tag{8.37}$$

Expanding the exponential on the right-hand side in terms of (8.36b), we have

$$\begin{aligned} \boldsymbol{\xi} &= \int_0^t [\beta_0(t-\tau)\mathbf{I}\mathbf{B}\mathbf{u}(\tau) + \beta_1(t-\tau)\mathbf{A}\mathbf{B}\mathbf{u}(\tau) + \cdots \\ &\quad \cdots + \beta_{n-1}(t-\tau)\mathbf{A}^{n-1}\mathbf{B}\mathbf{u}(\tau)]\,\mathrm{d}\tau \\ &= \mathbf{B}\int_0^t \beta_0(t-\tau)\mathbf{u}(\tau)\,\mathrm{d}\tau + \mathbf{A}\mathbf{B}\int_0^t \beta_1(t-\tau)\mathbf{u}(\tau)\,\mathrm{d}\tau + \cdots \\ &\quad \cdots + \mathbf{A}^{n-1}\mathbf{B}\int_0^t \beta_{n-1}(t-\tau)\mathbf{u}(\tau)\,\mathrm{d}\tau \\ &= \begin{bmatrix}\mathbf{B} & \mathbf{A}\mathbf{B} & \cdots & \mathbf{A}^{n-1}\mathbf{B}\end{bmatrix} \begin{bmatrix} \displaystyle\int_0^t \beta_0(t-\tau)\mathbf{u}(\tau)\,\mathrm{d}\tau \\ \displaystyle\int_0^t \beta_1(t-\tau)\mathbf{u}(\tau)\,\mathrm{d}\tau \\ \vdots \\ \displaystyle\int_0^t \beta_{n-1}(t-\tau)\mathbf{u}(\tau)\,\mathrm{d}\tau \end{bmatrix}. \end{aligned}$$

Note that the matrix on the left is the controllability matrix $\boldsymbol{\mathcal{C}}$, and we see that the only way that all of $\mathbb{R}^n$ is reachable is if the column space of $\boldsymbol{\mathcal{C}}$ spans all of $\mathbb{R}^n$. It is somewhat more difficult to see that if $\boldsymbol{\mathcal{C}}$ has rank n then it is possible to design a $\mathbf{u}(t)$ to reach any arbitrary state $\boldsymbol{\xi} \in \mathbb{R}^n$, but this relies on the fact that the n functions $\{\beta_j(t)\}_{j=0}^{n-1}$ are linearly independent functions. It is also the case that there is not a *unique* actuation input $\mathbf{u}(t)$ to reach a given state $\boldsymbol{\xi}$, as there are many different paths one may take.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Gramians and Degrees of Controllability/Observability

The previous tests for controllability and observability are binary, in the sense that the rank of $\mathcal{C}$ (respectively, $\mathcal{O}$) is either n, or it is not. However, there are *degrees* of controllability and observability, as some states $\mathbf{x}$ may be easier to control or estimate than others.

To identify which states are more or less controllable, one must analyze the eigendecomposition of the controllability Gramian:

$$\mathbf{W}_c(t) = \int_0^t e^{\mathbf{A}\tau}\mathbf{B}\mathbf{B}^* e^{\mathbf{A}^*\tau}\,\mathrm{d}\tau. \tag{8.38}$$

Similarly, the observability Gramian is given by

$$\mathbf{W}_o(t) = \int_0^t e^{\mathbf{A}^*\tau}\mathbf{C}^*\mathbf{C} e^{\mathbf{A}\tau}\,\mathrm{d}\tau. \tag{8.39}$$

These Gramians are often evaluated at infinite time, and, unless otherwise stated, we refer to $\mathbf{W}_c = \lim_{t\to\infty}\mathbf{W}_c(t)$ and $\mathbf{W}_o = \lim_{t\to\infty}\mathbf{W}_o(t)$.

The controllability of a state $\mathbf{x}$ is measured by $\mathbf{x}^*\mathbf{W}_c\mathbf{x}$, which will be larger for more controllable states. If the value of $\mathbf{x}^*\mathbf{W}_c\mathbf{x}$ is large, then it is possible to navigate the system far in the $\mathbf{x}$ direction with a unit control input. The observability of a state is similarly measured by $\mathbf{x}^*\mathbf{W}_o\mathbf{x}$. Both Gramians are symmetric and positive semi-definite, having non-negative eigenvalues. Thus, the eigenvalues and eigenvectors may be ordered hierarchically, with eigenvectors corresponding to large eigenvalues being more easily controllable or observable. In this way, the Gramians induce a new inner product over state space in terms of the controllability or observability of the states.

Gramians may be visualized by ellipsoids in state space, with the principal axes given by directions that are hierarchically ordered in terms of controllability or observability. An example of this visualization is shown in Fig. 9.2 in Chapter 9. In fact, Gramians may be used to design reduced-order models for high-dimensional systems. Through a balancing transformation, a key subspace is identified with the most jointly controllable and observable modes. These modes then define a good projection basis to define a model that captures the dominant input–output dynamics. This form of balanced model reduction will be investigated further in Section 9.2.

Gramians are also useful to determine the minimum-energy control $\mathbf{u}(t)$ required to navigate the system to $\mathbf{x}(t_f)$ at time t_f from $\mathbf{x}(0) = \mathbf{0}$:

$$\mathbf{u}(t) = \mathbf{B}^*(e^{\mathbf{A}(t_f-t)})^*\mathbf{W}_c(t_f)^{-1}\mathbf{x}(t_f). \tag{8.40}$$

The total energy expended by this control law is given by

$$\int_0^{t_f} \|\mathbf{u}(\tau)\|^2\,\mathrm{d}\tau = \mathbf{x}^*\mathbf{W}_c(t_f)^{-1}\mathbf{x}. \tag{8.41}$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

It can now be seen that if the controllability matrix is nearly singular, then there are directions that require extreme actuation energy to manipulate. Conversely, if the eigenvalues of $\mathbf{W}_c$ are all large, then the system is easily controlled.

It is generally impracticable to compute the Gramians directly using (8.38) and (8.39). Instead, the controllability Gramian is the solution to the following Lyapunov equation,

$$\mathbf{A}\mathbf{W}_c + \mathbf{W}_c\mathbf{A}^* + \mathbf{B}\mathbf{B}^* = \mathbf{0}, \tag{8.42}$$

while the observability Gramian is the solution to

$$\mathbf{A}^*\mathbf{W}_o + \mathbf{W}_o\mathbf{A} + \mathbf{C}^*\mathbf{C} = \mathbf{0}. \tag{8.43}$$

Obtaining Gramians by solving a Lyapunov equation is typically quite expensive for high-dimensional systems [76, 288, 310, 662, 669]. Instead, Gramians are often approximated empirically using snapshot data from the direct and adjoint systems, as will be discussed in Section 9.2.

Stabilizability and Detectability

In practice, full-state controllability and observability may be too much to expect in high-dimensional systems. For example, in a high-dimensional fluid system, it may be unrealistic to manipulate every minor fluid vortex; instead, control authority over the large, energy-containing coherent structures is often enough.

Stabilizability refers to the ability to control all unstable eigenvector directions of $\mathbf{A}$, so that they are in the span of $\mathcal{C}$. In practice, we might relax this definition to include lightly damped eigenvector modes, corresponding to eigenvalues with a small, negative real part. Similarly, if all unstable eigenvectors of $\mathbf{A}$ are in the span of $\mathcal{O}^*$, then the system is detectable.

There may also be states in the model description that are superfluous for control. As an example, consider the control system for a commercial passenger jet. The state of the system may include the passenger seat positions, although this will surely not be controllable by the pilot, nor should it be.

8.4 Optimal Full-State Control: Linear–Quadratic Regulator (LQR)

We have seen in the previous sections that if $(\mathbf{A}, \mathbf{B})$ is controllable, then it is possible to arbitrarily manipulate the eigenvalues of the closed-loop system $(\mathbf{A} - \mathbf{B}\mathbf{K})$ through choice of a full-state feedback control law $\mathbf{u} = -\mathbf{K}\mathbf{x}$. This implicitly assumes that full-state measurements are available (i.e., $\mathbf{C} = \mathbf{I}$ and $\mathbf{D} = \mathbf{0}$, so that $\mathbf{y} = \mathbf{x}$). Although full-state measurements are not always available, especially for high-dimensional systems, we will show in the next section

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

that, if the system is observable, it is possible to build a full-state estimate from the sensor measurements.

Given a controllable system, and either measurements of the full state or of an observable system with a full-state estimate, there are many choices of stabilizing control laws $\mathbf{u} = -\mathbf{K}\mathbf{x}$. It is possible to make the eigenvalues of the closed-loop system $(\mathbf{A} - \mathbf{B}\mathbf{K})$ arbitrarily stable, placing them as far as desired in the left half of the complex plane. However, overly stable eigenvalues may require exceedingly expensive control expenditure and might also result in actuation signals that exceed maximum allowable values. Choosing very stable eigenvalues may also cause the control system to overreact to noise and disturbances, much as a new driver will overreact to vibrations in the steering wheel, causing the closed-loop system to jitter. Over-stabilization can counter-intuitively degrade robustness and may lead to instability if there are small time delays or unmodeled dynamics. Robustness will be discussed in Section 8.8.

Choosing the best gain matrix $\mathbf{K}$ to stabilize the system without expending too much control effort is an important goal in *optimal* control. A balance must be struck between the stability of the closed-loop system and the aggressiveness of control. It is important to take control expenditure into account (1) to prevent the controller from overreacting to high-frequency noise and disturbances, (2) so that actuation does not exceed maximum allowed amplitudes, and (3) so that control is not prohibitively expensive. In particular, the cost function

$$J(t) = \int_0^t \mathbf{x}(\tau)^* \mathbf{Q} \mathbf{x}(\tau) + \mathbf{u}(\tau)^* \mathbf{R} \mathbf{u}(\tau) \, \mathrm{d}\tau \tag{8.44}$$

balances the cost of effective regulation of the state with the cost of control. The matrices $\mathbf{Q}$ and $\mathbf{R}$ weight the cost of deviations of the state from zero and the cost of actuation, respectively. The matrix $\mathbf{Q}$ is positive semi-definite, and $\mathbf{R}$ is positive definite; these matrices are often diagonal, and the diagonal elements may be tuned to change the relative importance of the control objectives.

Adding such a cost function makes choosing the control law a well-posed optimization problem, for which there is a wealth of theoretical and numerical techniques [101]. The linear–quadratic regulator (LQR) control law $\mathbf{u} = -\mathbf{K}_r \mathbf{x}$ is designed to minimize $J = \lim_{t \to \infty} J(t)$. LQR is so-named because it is a linear control law, designed for a linear system, minimizing a quadratic cost function, that regulates the state of the system to $\lim_{t \to \infty} \mathbf{x}(t) = \mathbf{0}$. Because the cost function in (8.44) is quadratic, there is an analytical solution for the optimal controller gains $\mathbf{K}_r$, given by

$$\mathbf{K}_r = \mathbf{R}^{-1} \mathbf{B}^* \mathbf{X}, \tag{8.45}$$

where $\mathbf{X}$ is the solution to an algebraic Riccati equation:

$$\mathbf{A}^* \mathbf{X} + \mathbf{X} \mathbf{A} - \mathbf{X} \mathbf{B} \mathbf{R}^{-1} \mathbf{B}^* \mathbf{X} + \mathbf{Q} = \mathbf{0}. \tag{8.46}$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

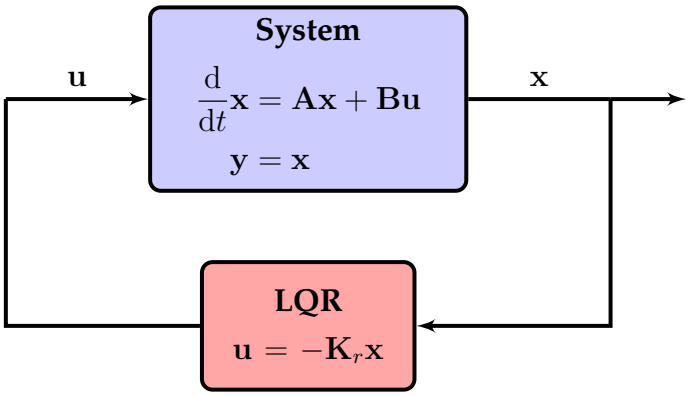


Figure 8.9: Schematic of the linear–quadratic regulator (LQR) for optimal full-state feedback. The optimal controller for a linear system given measurements of the full state, $\mathbf{y} = \mathbf{x}$, is given by proportional control $\mathbf{u} = -\mathbf{K}_r\mathbf{x}$, where $\mathbf{K}_r$ is a constant-gain matrix obtained by solving an algebraic Riccati equation.

Solving the above Riccati equation for $\mathbf{X}$, and hence for $\mathbf{K}_r$, is numerically robust and already implemented in many programming languages [76, 430]. The gain matrix $\mathbf{K}_r$ is obtained via

```
>> Kr = lqr(A,B,Q,R);
```

in MATLAB and via

```
>>> Kr = lqr(A,B,Q,R)[0]
```

in python-control. However, solving the Riccati equation scales as $\mathcal{O}(n^3)$ in the state dimension n, making it prohibitively expensive for large systems or for online computations for slowly changing state equations or linear parameter varying (LPV) control. This motivates the development of reduced-order models that capture the same dominant behavior with many fewer states. Control-oriented reduced-order models will be developed more in Chapter 9.

The LQR controller is shown schematically in Fig. 8.9. Out of all possible control laws $\mathbf{u} = \mathbf{K}(\mathbf{x})$, including nonlinear controllers, the LQR controller $\mathbf{u} = -\mathbf{K}_r\mathbf{x}$ is optimal, as we will show in Section 8.4. However, it may be the case that a linearized system is linearly uncontrollable while the full nonlinear system in (8.7) is controllable with a nonlinear control law $\mathbf{u} = \mathbf{K}(\mathbf{x})$.

Derivation of the Riccati Equation for Optimal Control

It is worth taking a theoretical detour here to derive the Riccati equation in (8.46) for the problem of optimal full-state regulation. This derivation will provide an example of how to solve convex optimization problems using the calculus of variations, and it will also provide a template for computing the optimal

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

control solution for *nonlinear* systems. Because of the similarity of optimal control to the formulation of Lagrangian and Hamiltonian classical mechanics in terms of the variational principal, we adopt similar language and notation.

First, we will add a terminal cost to our LQR cost function in (8.44), and also introduce a factor of $1/2$ to simplify computations:

$$J = \int_0^{t_f} \underbrace{\tfrac{1}{2}(\mathbf{x}^*\mathbf{Q}\mathbf{x} + \mathbf{u}^*\mathbf{R}\mathbf{u})}_{\text{Lagrangian, } \mathcal{L}} \,\mathrm{d}\tau + \underbrace{\tfrac{1}{2}\mathbf{x}(t_f)^*\mathbf{Q}_f\mathbf{x}(t_f)}_{\text{terminal cost}}. \tag{8.47}$$

The goal is to minimize the quadratic cost function J subject to the dynamical constraint

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}. \tag{8.48}$$

We may solve this using the calculus of variations by introducing the following augmented cost function:

$$J_{\text{aug}} = \int_0^{t_f} [\tfrac{1}{2}(\mathbf{x}^*\mathbf{Q}\mathbf{x} + \mathbf{u}^*\mathbf{R}\mathbf{u}) + \boldsymbol{\lambda}^*(\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} - \dot{\mathbf{x}})]\,\mathrm{d}\tau + \tfrac{1}{2}\mathbf{x}(t_f)^*\mathbf{Q}_f\mathbf{x}(t_f). \tag{8.49}$$

The variable $\boldsymbol{\lambda}$ is a Lagrange multiplier, called the *co-state*, that enforces the dynamic constraints; $\boldsymbol{\lambda}$ may take any value and $J_{\text{aug}} = J$ will hold.

Taking the total variation of J_{aug} in (8.49) yields

$$\delta J_{\text{aug}} = \int_0^{t_f} \left[\frac{\partial \mathcal{L}}{\partial \mathbf{x}}\delta\mathbf{x} + \frac{\partial \mathcal{L}}{\partial \mathbf{u}}\delta\mathbf{u} + \boldsymbol{\lambda}^*\mathbf{A}\delta\mathbf{x} + \boldsymbol{\lambda}^*\mathbf{B}\delta\mathbf{u} - \boldsymbol{\lambda}^*\delta\dot{\mathbf{x}}\right]\mathrm{d}\tau + \mathbf{Q}_f\mathbf{x}(t_f)\delta\mathbf{x}(t_f). \tag{8.50}$$

The partial derivatives[4] of the Lagrangian are $\partial\mathcal{L}/\partial\mathbf{x} = \mathbf{x}^*\mathbf{Q}$ and $\partial\mathcal{L}/\partial\mathbf{u} = \mathbf{u}^*\mathbf{R}$. The last term in the integral may be modified using integration by parts:

$$-\int_0^{t_f} \boldsymbol{\lambda}^*\delta\dot{\mathbf{x}}\,\mathrm{d}\tau = -\boldsymbol{\lambda}^*(t_f)\delta\mathbf{x}(t_f) + \boldsymbol{\lambda}^*(0)\delta\mathbf{x}(0) + \int_0^{t_f} \dot{\boldsymbol{\lambda}}^*\delta\mathbf{x}\,\mathrm{d}\tau.$$

The term $\boldsymbol{\lambda}^*(0)\delta\mathbf{x}(0)$ is equal to zero, or else the control system would be non-causal (i.e., then future control could change the initial condition of the system).

Finally, the total variation of the augmented cost function in (8.50) simplifies as follows:

$$\begin{aligned}\delta J_{\text{aug}} = &\int_0^{t_f} (\mathbf{x}^*\mathbf{Q} + \boldsymbol{\lambda}^*\mathbf{A} + \dot{\boldsymbol{\lambda}}^*)\delta\mathbf{x}\,\mathrm{d}\tau + \int_0^{t_f} (\mathbf{u}^*\mathbf{R} + \boldsymbol{\lambda}^*\mathbf{B})\delta\mathbf{u}\,\mathrm{d}\tau \\ &+ (\mathbf{x}(t_f)^*\mathbf{Q}_f - \boldsymbol{\lambda}^*(t_f))\delta\mathbf{x}(t_f).\end{aligned} \tag{8.51}$$

[4]The derivative of a matrix expression $\mathbf{A}\mathbf{x}$ with respect to $\mathbf{x}$ is $\mathbf{A}$, and the derivative of $\mathbf{x}^*\mathbf{A}$ with respect to $\mathbf{x}$ is $\mathbf{A}^*$.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Each variation term in (8.51) must equal zero for an optimal control solution that minimizes J. Thus, we may break this up into three equations:

$$\mathbf{x}^*\mathbf{Q} + \boldsymbol{\lambda}^*\mathbf{A} + \dot{\boldsymbol{\lambda}}^* = \mathbf{0}, \tag{8.52a}$$
$$\mathbf{u}^*\mathbf{R} + \boldsymbol{\lambda}^*\mathbf{B} = \mathbf{0}, \tag{8.52b}$$
$$\mathbf{x}(t_f)^*\mathbf{Q}_f - \boldsymbol{\lambda}^*(t_f) = \mathbf{0}. \tag{8.52c}$$

Note that the constraint in (8.52c) represents an initial condition for the reverse-time equation for $\boldsymbol{\lambda}$ starting at t_f. Thus, the dynamics in (8.48) with initial condition $\mathbf{x}(0) = \mathbf{x}_0$ and (8.52) with the final-time condition $\boldsymbol{\lambda}(t_f) = \mathbf{Q}_f\mathbf{x}(t_f)$ form a two-point boundary value problem. This may be integrated numerically to find the optimal control solution, even for nonlinear systems.

Because the dynamics are linear, it is possible to *posit* the form $\boldsymbol{\lambda} = \mathbf{P}\mathbf{x}$, and substitute into (8.52) above. The first equation becomes:

$$(\dot{\mathbf{P}}\mathbf{x} + \mathbf{P}\dot{\mathbf{x}})^* + \mathbf{x}^*\mathbf{Q} + \boldsymbol{\lambda}^*\mathbf{A} = \mathbf{0}.$$

Taking the transpose, and substituting (8.48) in for $\dot{\mathbf{x}}$, yields

$$\dot{\mathbf{P}}\mathbf{x} + \mathbf{P}(\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}) + \mathbf{Q}\mathbf{x} + \mathbf{A}^*\mathbf{P}\mathbf{x} = \mathbf{0}.$$

From (8.52b), we have

$$\mathbf{u} = -\mathbf{R}^{-1}\mathbf{B}^*\boldsymbol{\lambda} = -\mathbf{R}^{-1}\mathbf{B}^*\mathbf{P}\mathbf{x}.$$

Finally, combining yields

$$\dot{\mathbf{P}}\mathbf{x} + \mathbf{P}\mathbf{A}\mathbf{x} + \mathbf{A}^*\mathbf{P}\mathbf{x} - \mathbf{P}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^*\mathbf{P}\mathbf{x} + \mathbf{Q}\mathbf{x} = \mathbf{0}. \tag{8.53}$$

This equation must be true for all $\mathbf{x}$, and so it may also be written as a matrix equation. Dropping the terminal cost and letting time go to infinity, the $\dot{\mathbf{P}}$ term disappears, and we recover the algebraic Riccati equation:

$$\mathbf{P}\mathbf{A} + \mathbf{A}\mathbf{P}^* - \mathbf{P}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^*\mathbf{P} + \mathbf{Q} = \mathbf{0}.$$

Although this procedure is somewhat involved, each step is relatively straightforward. In addition, the dynamics in (8.48) may be replaced with nonlinear dynamics $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u})$, and a similar nonlinear two-point boundary value problem may be formulated with $\partial\mathbf{f}/\partial\mathbf{x}$ replacing $\mathbf{A}$ and $\partial\mathbf{f}/\partial\mathbf{u}$ replacing $\mathbf{B}$. This procedure is extremely general, and may be used to numerically obtain nonlinear optimal control trajectories.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Hamiltonian Formulation. Similar to the Lagrangian formulation above, it is also possible to solve the optimization problem by introducing the following Hamiltonian:

$$\mathcal{H} = \underbrace{\tfrac{1}{2}(\mathbf{x}^*\mathbf{Q}\mathbf{x} + \mathbf{u}^*\mathbf{R}\mathbf{u})}_{\mathcal{L}} + \boldsymbol{\lambda}^*(\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}). \tag{8.54}$$

Then Hamilton's equations become

$$\dot{\mathbf{x}} = \left(\frac{\partial \mathcal{H}}{\partial \boldsymbol{\lambda}}\right)^* = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}, \quad \mathbf{x}(0) = \mathbf{x}_0, \tag{8.55a}$$

$$-\dot{\boldsymbol{\lambda}} = \left(\frac{\partial \mathcal{H}}{\partial \mathbf{x}}\right)^* = \mathbf{Q}\mathbf{x} + \mathbf{A}^*\boldsymbol{\lambda}, \quad \boldsymbol{\lambda}(t_f) = \mathbf{Q}_f\mathbf{x}(t_f). \tag{8.55b}$$

Again, this is a two-point boundary value problem in $\mathbf{x}$ and $\boldsymbol{\lambda}$. Plugging in the same expression $\boldsymbol{\lambda} = \mathbf{P}\mathbf{x}$ will result in the same Riccati equation as above.

8.5 Optimal Full-State Estimation: the Kalman Filter

The optimal LQR controller from Section 8.4 relies on full-state measurements of the system. However, full-state measurements may be either prohibitively expensive or technologically infeasible to obtain, especially for high-dimensional systems. The computational burden of collecting and processing full-state measurements may also introduce unacceptable time delays that will limit robust performance.

Instead of measuring the full state $\mathbf{x}$, it may be possible to estimate the state from limited noisy measurements $\mathbf{y}$. In fact, full-state estimation is mathematically possible as long as the pair $(\mathbf{A}, \mathbf{C})$ are observable, although the effectiveness of estimation depends on the degree of observability as quantified by the observability Gramian. The Kalman filter [296, 370, 750] is the most commonly used full-state estimator, as it optimally balances the competing effects of measurement noise, disturbances, and model uncertainty. As will be shown in the next section, it is possible to use the full-state estimate from a Kalman filter in conjunction with the optimal full-state LQR feedback law.

When deriving the optimal full-state estimator, it is necessary to reintroduce disturbances to the state, $\mathbf{w}_d$, and sensor noise, $\mathbf{w}_n$:

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{x} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} + \mathbf{w}_d, \tag{8.56a}$$

$$\mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u} + \mathbf{w}_n. \tag{8.56b}$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

The Kalman filter assumes that both the disturbance and noise are zero-mean Gaussian processes with known covariances:

$$\mathbb{E}(\mathbf{w}_d(t)\mathbf{w}_d(\tau)^*) = \mathbf{V}_d\delta(t-\tau), \tag{8.57a}$$
$$\mathbb{E}(\mathbf{w}_n(t)\mathbf{w}_n(\tau)^*) = \mathbf{V}_n\delta(t-\tau). \tag{8.57b}$$

Here $\mathbb{E}$ is the expected value and $\delta(\cdot)$ is the Dirac delta function. The matrices $\mathbf{V}_d$ and $\mathbf{V}_n$ are positive semi-definite with entries containing the covariances of the disturbance and noise terms. Extensions to the Kalman filter exist for correlated, biased, and unknown noise and disturbance terms [489, 671].

It is possible to obtain an estimate $\hat{\mathbf{x}}$ of the full state $\mathbf{x}$ from measurements of the input $\mathbf{u}$ and output $\mathbf{y}$, via the following estimator dynamical system:

$$\frac{\mathrm{d}}{\mathrm{d}t}\hat{\mathbf{x}} = \mathbf{A}\hat{\mathbf{x}} + \mathbf{B}\mathbf{u} + \mathbf{K}_f(\mathbf{y}-\hat{\mathbf{y}}), \tag{8.58a}$$
$$\hat{\mathbf{y}} = \mathbf{C}\hat{\mathbf{x}} + \mathbf{D}\mathbf{u}. \tag{8.58b}$$

The matrices $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, and $\mathbf{D}$ are obtained from the system model, and the filter gain $\mathbf{K}_f$ is determined via a similar procedure as in LQR. Thus $\mathbf{K}_f$ is given by

$$\mathbf{K}_f = \mathbf{Y}\mathbf{C}^*\mathbf{V}_n^{-1}, \tag{8.59}$$

where $\mathbf{Y}$ is the solution to another algebraic Riccati equation:

$$\mathbf{Y}\mathbf{A}^* + \mathbf{A}\mathbf{Y} - \mathbf{Y}\mathbf{C}^*\mathbf{V}_n^{-1}\mathbf{C}\mathbf{Y} + \mathbf{V}_d = \mathbf{0}. \tag{8.60}$$

This solution is commonly referred to as the Kalman filter, and it is the optimal full-state estimator with respect to the following cost function:

$$J = \lim_{t\to\infty} \mathbb{E}((\mathbf{x}(t)-\hat{\mathbf{x}}(t))^*(\mathbf{x}(t)-\hat{\mathbf{x}}(t))). \tag{8.61}$$

This cost function implicitly includes the effects of disturbance and noise, which are required to determine the optimal balance between aggressive estimation and noise attenuation. Thus, the Kalman filter is referred to as *linear–quadratic estimation* (LQE), and has a dual formulation to the LQR optimization. The cost in (8.61) is computed as an ensemble average over many realizations.

The Kalman filter gain $\mathbf{K}_f$ may be determined via

```
>> Kf = lqe(A,I,C,Vd,Vn); % design Kalman filter gain
```

where $\mathbf{I}$ is the $n \times n$ identity matrix. Optimal control and estimation are mathematical dual problems, as are controllability and observability, so the Kalman filter may also be found using LQR:

```
>> Kf = (lqr(A',C',Vd,Vn))'; % LQR and LQE are dual problems
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

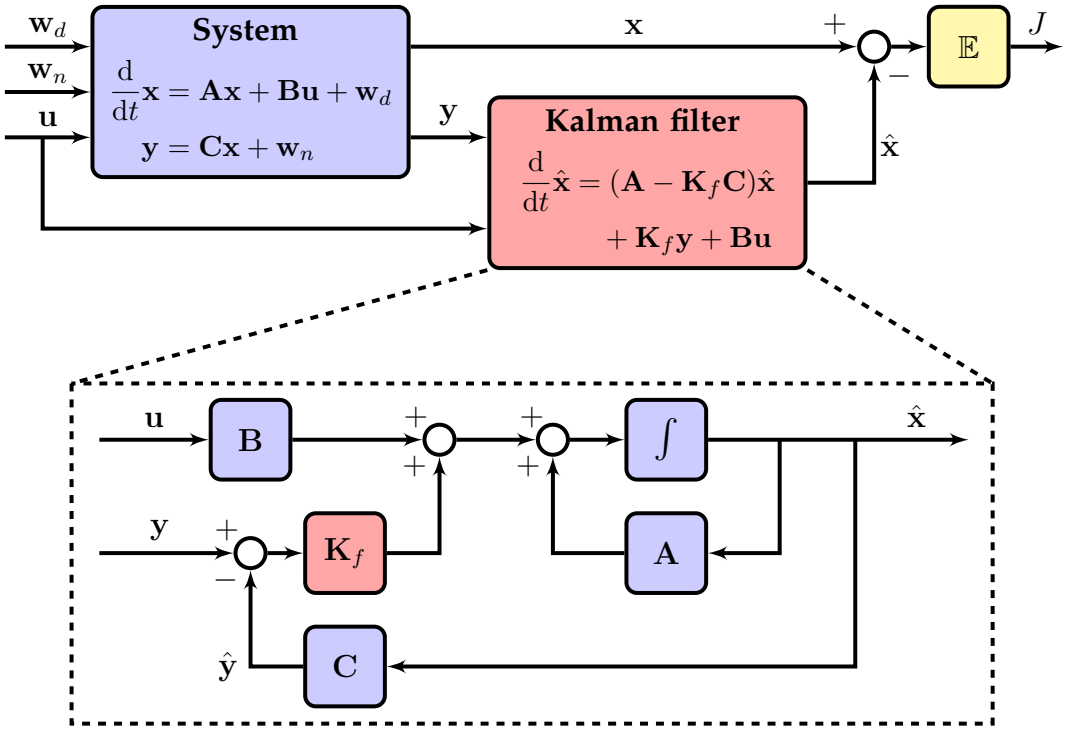


Figure 8.10: Schematic of the Kalman filter for full-state estimation from noisy measurements $\mathbf{y} = \mathbf{C}\mathbf{x}+\mathbf{w}_n$ with process noise (disturbances) $\mathbf{w}_d$. This diagram does not have a feedthrough term $\mathbf{D}$, although it may be included.

The Kalman filter is shown schematically in Fig. 8.10.

Substituting the output estimate $\hat{\mathbf{y}}$ from (8.58b) into (8.58a) yields

$$\frac{\mathrm{d}}{\mathrm{d}t}\hat{\mathbf{x}} = (\mathbf{A} - \mathbf{K}_f\mathbf{C})\hat{\mathbf{x}} + \mathbf{K}_f\mathbf{y} + (\mathbf{B} - \mathbf{K}_f\mathbf{D})\mathbf{u} \tag{8.62a}$$

$$= (\mathbf{A} - \mathbf{K}_f\mathbf{C})\hat{\mathbf{x}} + \begin{bmatrix}\mathbf{K}_f & (\mathbf{B} - \mathbf{K}_f\mathbf{D})\end{bmatrix}\begin{bmatrix}\mathbf{y}\\ \mathbf{u}\end{bmatrix}. \tag{8.62b}$$

The estimator dynamical system is expressed in terms of the estimate $\hat{\mathbf{x}}$ with inputs $\mathbf{y}$ and $\mathbf{u}$. If the system is observable, it is possible to place the eigenvalues of $\mathbf{A}-\mathbf{K}_f\mathbf{C}$ arbitrarily with choice of $\mathbf{K}_f$. When the eigenvalues of the estimator are stable, then the state estimate $\hat{\mathbf{x}}$ converges to the full state $\mathbf{x}$ asymptotically, as long as the model faithfully captures the true system dynamics. To see this

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

convergence, consider the dynamics of the estimation error $\boldsymbol{\epsilon} = \mathbf{x} - \hat{\mathbf{x}}$:

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{\epsilon} &= \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{x} - \frac{\mathrm{d}}{\mathrm{d}t}\hat{\mathbf{x}} \\
&= [\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} + \mathbf{w}_d] - [(\mathbf{A} - \mathbf{K}_f\mathbf{C})\hat{\mathbf{x}} + \mathbf{K}_f\mathbf{y} + (\mathbf{B} - \mathbf{K}_f\mathbf{D})\mathbf{u}] \\
&= \mathbf{A}\boldsymbol{\epsilon} + \mathbf{w}_d + \mathbf{K}_f\mathbf{C}\hat{\mathbf{x}} - \mathbf{K}_f\mathbf{y} + \mathbf{K}_f\mathbf{D}\mathbf{u} \\
&= \mathbf{A}\boldsymbol{\epsilon} + \mathbf{w}_d + \mathbf{K}_f\mathbf{C}\hat{\mathbf{x}} - \mathbf{K}_f \underbrace{[\mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u} + \mathbf{w}_n]}_{\mathbf{y}} + \mathbf{K}_f\mathbf{D}\mathbf{u} \\
&= (\mathbf{A} - \mathbf{K}_f\mathbf{C})\boldsymbol{\epsilon} + \mathbf{w}_d - \mathbf{K}_f\mathbf{w}_n.
\end{aligned}$$

Therefore, the estimate $\hat{\mathbf{x}}$ will converge to the true full state when $\mathbf{A} - \mathbf{K}_f\mathbf{C}$ has stable eigenvalues. As with LQR, there is a tradeoff between over-stabilization of these eigenvalues and the amplification of sensor noise. This is similar to the behavior of an inexperienced driver who may hold the steering wheel too tightly and will overreact to every minor bump and disturbance on the road.

There are many variants of the Kalman filter for nonlinear systems [360, 361, 729], including the extended and unscented Kalman filters. The ensemble Kalman filter [20] is an extension that works well for high-dimensional systems, such as in geophysical data assimilation [596]. All of these methods still assume Gaussian noise processes, and the particle filter provides a more general, although more computationally intensive, alternative that can handle arbitrary noise distributions [306, 602]. The unscented Kalman filter balances the efficiency of the Kalman filter and accuracy of the particle filter.

8.6 Optimal Sensor-Based Control: Linear–Quadratic Gaussian (LQG)

The full-state estimate from the Kalman filter is generally used in conjunction with the full-state feedback control law from LQR, resulting in optimal sensor-based feedback. Remarkably, the LQR gain $\mathbf{K}_r$ and the Kalman filter gain $\mathbf{K}_f$ may be designed separately, and the resulting sensor-based feedback will remain optimal and retain the closed-loop eigenvalues when combined.

Combining the LQR full-state feedback with the Kalman filter full-state estimator results in the linear–quadratic Gaussian (LQG) controller. The LQG controller is a dynamical system with input $\mathbf{y}$, output $\mathbf{u}$, and internal state $\hat{\mathbf{x}}$:

$$\frac{\mathrm{d}}{\mathrm{d}t}\hat{\mathbf{x}} = (\mathbf{A} - \mathbf{K}_f\mathbf{C} - \mathbf{B}\mathbf{K}_r)\hat{\mathbf{x}} + \mathbf{K}_f\mathbf{y}, \tag{8.63a}$$

$$\mathbf{u} = -\mathbf{K}_r\hat{\mathbf{x}}. \tag{8.63b}$$

The LQG controller is optimal with respect to the following ensemble-averaged

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

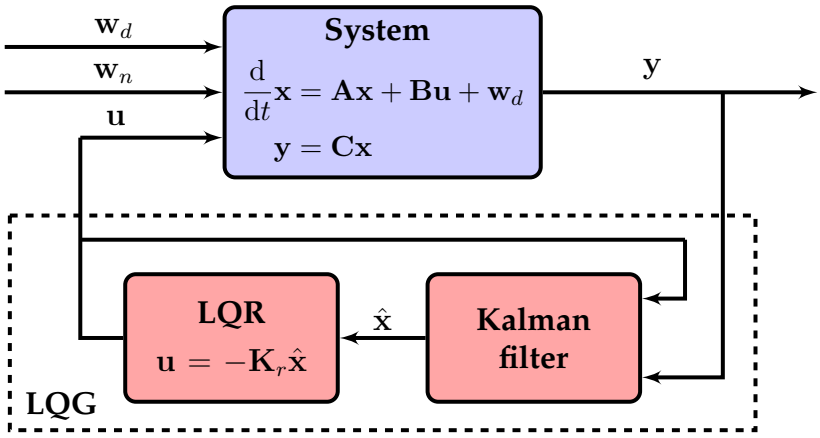


Figure 8.11: Schematic illustrating the linear–quadratic Gaussian (LQG) controller for optimal closed-loop feedback based on noisy measurements $\mathbf{y}$. The optimal LQR and Kalman filter gain matrices $\mathbf{K}_r$ and $\mathbf{K}_f$ may be designed independently, based on two different algebraic Riccati equations. When combined, the resulting sensor-based feedback remains optimal.

version of the cost function from (8.44):

$$J(t) = \left\langle \int_0^t \left[\mathbf{x}(\tau)^* \mathbf{Q}\mathbf{x}(\tau) + \mathbf{u}(\tau)^* \mathbf{R}\mathbf{u}(\tau)\right] \mathrm{d}\tau \right\rangle. \tag{8.64}$$

The controller $\mathbf{u} = -\mathbf{K}_r\hat{\mathbf{x}}$ is in terms of the state estimate, and so this cost function must be averaged over many realizations of the disturbance and noise. Applying LQR to $\hat{\mathbf{x}}$ results in the following state dynamics:

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{x} &= \mathbf{A}\mathbf{x} - \mathbf{B}\mathbf{K}_r\hat{\mathbf{x}} + \mathbf{w}_d & \text{(8.65a)} \\ &= \mathbf{A}\mathbf{x} - \mathbf{B}\mathbf{K}_r\mathbf{x} + \mathbf{B}\mathbf{K}_r(\mathbf{x} - \hat{\mathbf{x}}) + \mathbf{w}_d & \text{(8.65b)} \\ &= \mathbf{A}\mathbf{x} - \mathbf{B}\mathbf{K}_r\mathbf{x} + \mathbf{B}\mathbf{K}_r\boldsymbol{\epsilon} + \mathbf{w}_d. & \text{(8.65c)} \end{aligned}$$

Again $\boldsymbol{\epsilon} = \mathbf{x} - \hat{\mathbf{x}}$ as before. Finally, the closed-loop system may be written as

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix}\mathbf{x}\\ \boldsymbol{\epsilon}\end{bmatrix} = \begin{bmatrix}\mathbf{A}-\mathbf{B}\mathbf{K}_r & \mathbf{B}\mathbf{K}_r\\ \mathbf{0} & \mathbf{A}-\mathbf{K}_f\mathbf{C}\end{bmatrix}\begin{bmatrix}\mathbf{x}\\ \boldsymbol{\epsilon}\end{bmatrix} + \begin{bmatrix}\mathbf{I} & \mathbf{0}\\ \mathbf{I} & -\mathbf{K}_f\end{bmatrix}\begin{bmatrix}\mathbf{w}_d\\ \mathbf{w}_n\end{bmatrix}. \tag{8.66}$$

Thus, the closed-loop eigenvalues of the LQG-regulated system are given by the eigenvalues of $\mathbf{A}-\mathbf{B}\mathbf{K}_r$ and $\mathbf{A}-\mathbf{K}_f\mathbf{C}$, which were optimally chosen by the LQR and Kalman filter gain matrices, respectively.

The LQG framework, shown in Fig. 8.11, relies on an accurate model of the system and knowledge of the magnitudes of the disturbances and measurement noise, which are assumed to be Gaussian processes. In real-world systems, each of these assumptions may be invalid, and even small time delays

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

and model uncertainty may destroy the robustness of LQG and result in instability [209]. The lack of robustness of LQG regulators to model uncertainty motivates the introduction of robust control in Section 8.8. For example, it is possible to *robustify* LQG regulators through a process known as loop-transfer recovery. However, despite robustness issues, LQG control is extremely effective for many systems, and is among the most common control paradigms.

In contrast to classical control approaches, such as proportional–integral–derivative (PID) control and designing faster inner-loop control and slow outer-loop control assuming a separation of timescales, LQG is able to handle multiple-input, multiple-output (MIMO) systems with overlapping timescales and multi-objective cost functions with no additional complexity in the algorithm or implementation.

8.7 Case Study: Inverted Pendulum on a Cart

To consolidate the concepts of optimal control, we will implement a stabilizing controller for an inverted pendulum on a cart, shown in Fig. 8.12. The full nonlinear dynamics are given by

$$\dot{x} = v, \tag{8.67a}$$

$$\dot{v} = \frac{-m^2L^2g\cos(\theta)\sin(\theta) + mL^2(mL\omega^2\sin(\theta) - \delta v) + mL^2u}{mL^2(M + m(1-\cos(\theta)^2))}, \tag{8.67b}$$

$$\dot{\theta} = \omega, \tag{8.67c}$$

$$\dot{\omega} = \frac{(m+M)mgL\sin(\theta) - mL\cos(\theta)(mL\omega^2\sin(\theta) - \delta v) - mL\cos(\theta)u}{mL^2(M + m(1-\cos(\theta)^2))}, \tag{8.67d}$$

where x is the cart position, v is the velocity, θ is the pendulum angle, ω is the angular velocity, m is the pendulum mass, M is the cart mass, L is the pendulum arm, g is the gravitational acceleration, δ is a friction damping on the cart, and u is a control force applied to the cart.

The function **pendcart**, defined in Code 8.2, may be used to simulate the full nonlinear system in (8.67).

Code 8.2: [MATLAB] Right-hand side function for inverted pendulum on cart.

```
function dx = pendcart(x,m,M,L,g,d,u)

Sx = sin(x(3));
Cx = cos(x(3));
D = m*L*L*(M+m*(1-Cx^2));

dx(1,1) = x(2);
dx(2,1) = (1/D)*(-m^2*L^2*g*Cx*Sx + m*L^2*(m*L*x(4)^2*Sx - d
    *x(2))) + m*L*L*(1/D)*u;
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

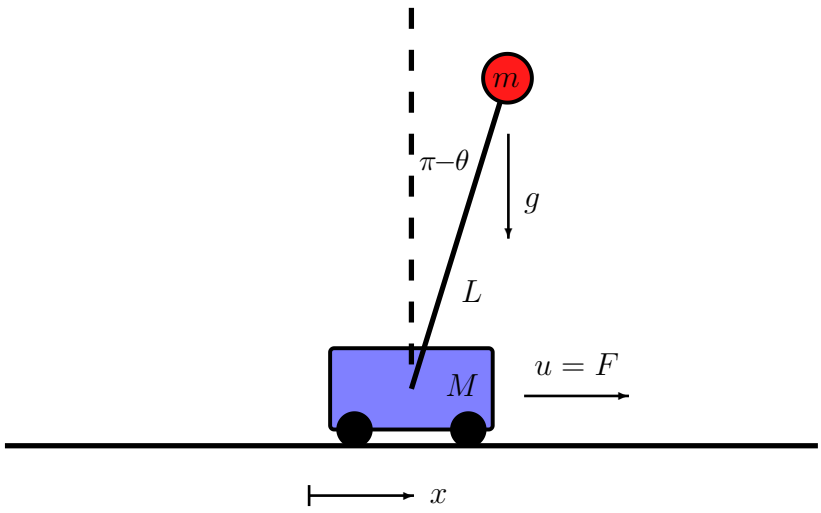


Figure 8.12: Schematic of inverted pendulum on a cart. The control forcing acts to accelerate or decelerate the cart. For this example, we assume the following parameter values: pendulum mass ($m = 1$), cart mass ($M = 5$), pendulum length ($L = 2$), gravitational acceleration ($g = -10$), and cart damping ($\delta = 1$).

```
dx(3,1) = x(4);
dx(4,1) = (1/D)*((m+M)*m*g*L*Sx - m*L*Cx*(m*L*x(4)^2*Sx - d*
    x(2))) - m*L*Cx*(1/D)*u;
```

Code 8.2: [Python] Right-hand side function for inverted pendulum on cart.

```
def pendcart(x,t,m,M,L,g,d,uf):
    u = uf(x) # evaluate anonymous function at x
    Sx = np.sin(x[2])
    Cx = np.cos(x[2])
    D = m*L*L*(M+m*(1-Cx**2))

    dx = np.zeros(4)
    dx[0] = x[1]
    dx[1] = (1/D)*(-(m**2)*(L**2)*g*Cx*Sx + m*(L**2)*(m*L*(x
        [3]**2)*Sx - d*x[1])) + m*L*L*(1/D)*u
    dx[2] = x[3]
    dx[3] = (1/D)*((m+M)*m*g*L*Sx - m*L*Cx*(m*L*(x[3]**2)*Sx
        - d*x[1])) - m*L*Cx*(1/D)*u;

    return dx
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

There are two fixed points, corresponding to either the pendulum-down ($\theta = 0$) or pendulum-up ($\theta = \pi$) configuration; in both cases, $v = \omega = 0$ for the fixed point, and the cart position x is a free variable, as the equations do not depend explicitly on x. It is possible to linearize the equations in (8.67) about either the up or down solutions, yielding the following linearized dynamics:

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -\delta/M & bmg/M & 0 \\ 0 & 0 & 0 & 1 \\ 0 & -b\delta/ML & -b(m+M)g/ML & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} 0 \\ 1/M \\ 0 \\ b/ML \end{bmatrix} u \tag{8.68}$$

for

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} x \\ v \\ \theta \\ \omega \end{bmatrix},$$

where $b = 1$ for the pendulum-up fixed point, and $b = -1$ for the pendulum-down fixed point. The system matrices $\mathbf{A}$ and $\mathbf{B}$ are initialized in Code 8.3 using the values for the constants given in Fig. 8.12.

Code 8.3: [MATLAB] Construct system matrices for inverted pendulum on a cart.

```
m = 1;  M = 5;  L = 2;  g = -10;  d = 1;

b = 1; % Pendulum up (b=1)

A = [0 1 0 0;
     0 -d/M b*m*g/M 0;
     0 0 0 1;
     0 -b*d/(M*L) -b*(m+M)*g/(M*L) 0];
B = [0; 1/M; 0; b*1/(M*L)];
```

Code 8.3: [Python] Construct system matrices for inverted pendulum on a cart.

```
m = 1;  M = 5;  L = 2;  g = -10;  d = 1;

b = 1 # pendulum up (b=1)

A = np.array([[0,1,0,0], [0,-d/M,b*m*g/M,0], [0,0,0,1], [0,-
    b*d/(M*L),-b*(m+M)*g/(M*L),0]])

B = np.array([0,1/M,0,b/(M*L)]).reshape((4,1))
```

We may also confirm that the open-loop system is unstable by checking the eigenvalues of $\mathbf{A}$ using `eig(A)` in MATLAB or `np.linalg.eig(A)` in Python, which returns

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

```
array([ 0.        , -2.431123  , -0.23363938,  2.46476238])
```

In the following, we will test for controllability and observability, develop full-state feedback (LQR), full-state estimation (Kalman filter), and sensor-based feedback (LQG) solutions.

Full-State Feedback Control of the Cart–Pendulum

In this section, we will design an LQR controller to stabilize the inverted pendulum configuration ($\theta = \pi$) assuming full-state measurements, $\mathbf{y} = \mathbf{x}$. Before any control design, we must confirm that the system is linearly controllable with the given $\mathbf{A}$ and $\mathbf{B}$ matrices. This is accomplished by computing the rank of the controllability matrix using either `rank(crtr(A,B))` in MATLAB or `numpy.linalg.matrix_rank(ctrb(A,B))` in Python, which returns a rank of 4. Thus, the pair $(\mathbf{A}, \mathbf{B})$ is controllable, since the controllability matrix has full rank. It is then possible to specify given $\mathbf{Q}$ and $\mathbf{R}$ matrices for the cost function and design the LQR controller gain matrix $\mathbf{K}$, as in Code 8.4.

Code 8.4: [MATLAB] Design LQR controller to stabilize inverted pendulum on a cart.

```
Q = eye(4);   % state cost, 4x4 identity matrix
R = .0001;    % control cost

K = lqr(A,B,Q,R);
```

Code 8.4: [Python] Design LQR controller to stabilize inverted pendulum on a cart.

```
Q = np.eye(4) # state cost, 4x4 identity matrix
R = 0.0001    # control cost

K = lqr(A,B,Q,R)[0]
```

We may then simulate the closed-loop system response of the full nonlinear system. We will initialize our simulation slightly off equilibrium, at $\mathbf{x}_0 = \begin{bmatrix} -1 & 0 & \pi + 0.1 & 0 \end{bmatrix}^T$, and we also impose a desired step change in the reference position of the cart, from $x = -1$ to $x = 1$.

Code 8.5: [MATLAB] Simulate closed-loop inverted pendulum on a cart system.

```
tspan = 0:.001:10;
x0 = [-1; 0; pi+.1; 0];   % initial condition
wr = [1; 0; pi; 0];       % reference position
u=@(x)-K*(x - wr);        % control law
[t,x] = ode45(@(t,x)pendcart(x,m,M,L,g,d,u(x)),tspan,x0);
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

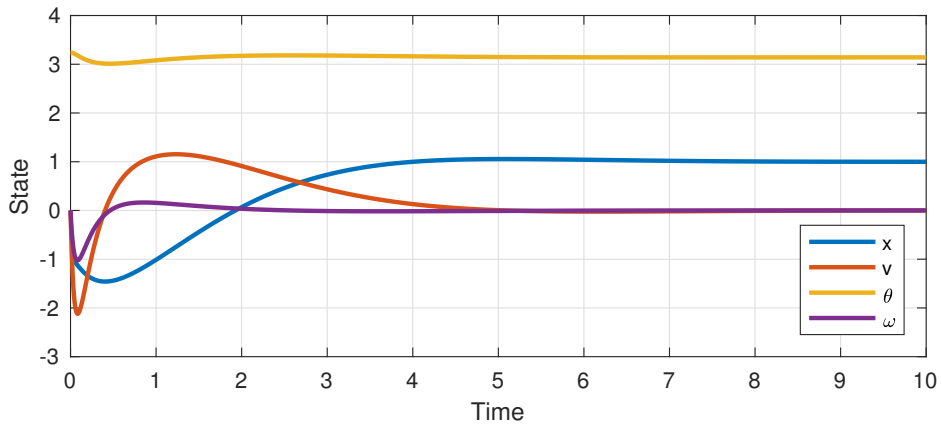


Figure 8.13: Closed-loop system response of inverted pendulum on a cart stabilized with an LQR controller.

Code 8.5: [Python] Simulate closed-loop inverted pendulum on a cart system.

```
tspan = np.arange(0,10,0.001)
x0 = np.array([-1,0,np.pi+0.1,0]) # Initial condition
wr = np.array([1,0,np.pi,0])      # Reference position
u = lambda x: -K@(x-wr)           # Control law
x = integrate.odeint(pendcart,x0,tspan,args=(m,M,L,g,d,u))
```

In this code, the actuation is set to

$$u = -\mathbf{K}(\mathbf{x} - \mathbf{w}_r), \tag{8.69}$$

where $\mathbf{w}_r = \begin{bmatrix} 1 & 0 & \pi & 0 \end{bmatrix}^T$ is the reference position. The closed-loop response is shown in Fig. 8.13.

In the above procedure, specifying the system dynamics and simulating the closed-loop system response is considerably more involved than actually designing the controller, which amounts to a single function call in MATLAB and Python. It is also helpful to compare the LQR response to the response obtained by non-optimal eigenvalue placement. In particular, Fig. 8.14 shows the system response and cost function for 100 randomly generated sets of stable eigenvalues, chosen in the interval $[-3.5, -0.5]$. The LQR controller has the lowest overall cost, as it is chosen to minimize J. The code to plot the pendulum–cart system is provided online.

Non-Minimum-Phase Systems. It can be seen from the response that, in order to move from $x = -1$ to $x = 1$, the system initially moves in the wrong direction. This behavior indicates that the system is a *non-minimum-phase* system, which introduces challenges for robust control, as we will soon see. There

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

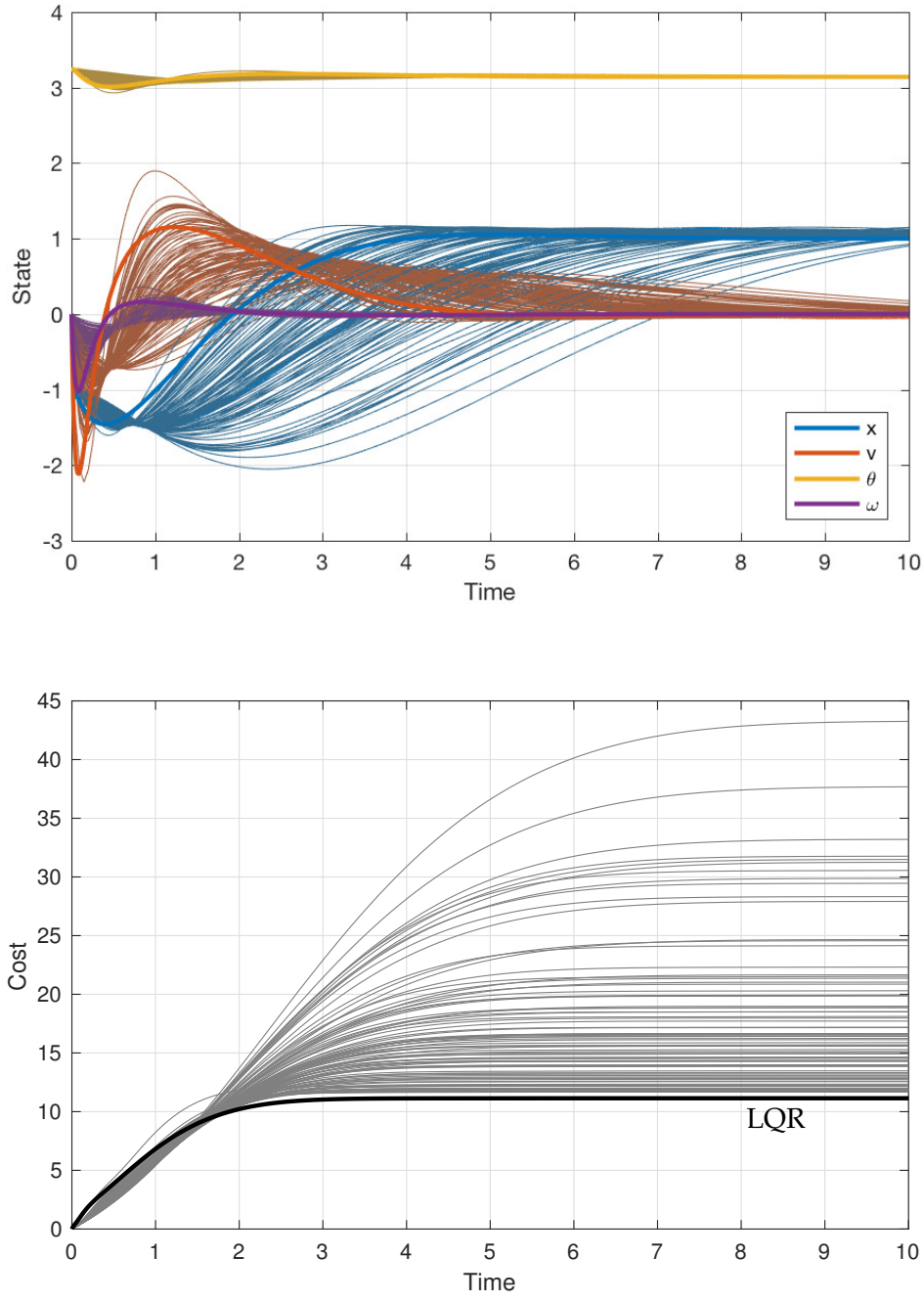


Figure 8.14: Comparison of LQR controller response and cost function with other pole placement locations. Bold lines represent the LQR solutions.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

are many examples of non-minimum-phase systems in control. For instance, parallel parking an automobile first involves moving the center of mass of the car away from the curb before it then moves closer. Other examples include increasing altitude in an aircraft, where the elevators must first move the center of mass down to increase the angle of attack on the main wings before lift increases the altitude. Adding cold fuel to a turbine may also initially drop the temperature before it eventually increases.

Full-State Estimation of the Cart–Pendulum

Now we turn to the full-state estimation problem based on limited noisy measurements $\mathbf{y}$. For this example, we will develop the Kalman filter for the pendulum-down condition ($\theta = 0$), since without feedback the system in the pendulum-up condition will quickly leave the fixed point where the linear model is valid. When we combine the Kalman filter with LQR in the next example, it will be possible to control the unstable inverted pendulum configuration. Switching to the pendulum-down configuration is simple in the code by setting $b = -1$.

Before designing a Kalman filter, we must choose a sensor and test for observability. If we measure the cart position, $y = x_1$, which corresponds to a matrix $\mathbf{C} = \begin{bmatrix}1 & 0 & 0 & 0\end{bmatrix}$, then the observability matrix has a full rank of 4. This may be confirmed using the command `rank(obsv(A,C))` in MATLAB or `numpy.linalg.matrix_rank(obsv(A,C))` in Python.

Because the cart position x_1 does not appear explicitly in the dynamics, the system is not fully observable for any measurement that does not include x_1. Thus, it is impossible to estimate the cart position with a measurement of the pendulum angle. However, if the cart position is not important for the cost function (i.e., if we only want to stabilize the pendulum, and do not care where the cart is located), then other choices of sensor will be admissible.

Now we design the Kalman filter, specifying disturbance and noise covariances, in Code 8.6.

Code 8.6: [MATLAB] Code to specify disturbance and noise magnitudes and develop Kalman filter gain.

```
Vd = eye(4);   % disturbance covariance
Vn = 1;        % noise covariance

%  Build Kalman filter
[Kf,P,E] = lqe(A,eye(4),C,Vd,Vn);  % design Kalman filter
% alternatively, possible to design using "LQR" code
Kf = (lqr(A',C',Vd,Vn))';
```

Code 8.6: [Python] Code to specify disturbance and noise magnitudes and de-

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

velop Kalman filter gain.

```
Vd = np.eye(4)  # disturbance covariance
Vn = 1          # noise covariance

# Build Kalman filter
Kf, P, E = lqe(A,np.eye(4),C,Vd,Vn)
```

The Kalman filter gain matrix is given by

$$\mathbf{K}_f = \begin{bmatrix} 1.9222 & 1.3474 & -0.6182 & -1.8016 \end{bmatrix}^T.$$

To simulate the system and Kalman filter, we must augment the original system to include disturbance and noise inputs, as in Code 8.7.

Code 8.7: [MATLAB] Augment system inputs with disturbance and noise terms, and create Kalman filter system.

```
B_aug = [B eye(4) 0*B];  % [u I*wd 0*wn]
D_aug = [0 0 0 0 0 1];   % D matrix passes noise through

sysC = ss(A,B_aug,C,D_aug);  % single-measurement system

% "true" system w/ full-state output, disturbance, no noise
sysTruth = ss(A,B_aug,eye(4),zeros(4,size(B_aug,2)));

sysKF = ss(A-Kf*C,[B Kf],eye(4),0*[B Kf]);  % Kalman filter
```

Code 8.7: [Python] Augment system inputs with disturbance and noise terms, and create Kalman filter system.

```
Baug = np.concatenate((B, np.eye(4),np.zeros_like(B)),axis
    =1) # [u I*wd 0*wn]
Daug = np.array([0,0,0,0,0,1]) # D matrix passes noise
    through

sysC = ss(A,Baug,C,Daug) # Single-measurement system

# "True" system w/ full-state output, disturbance, no noise
sysTruth = ss(A,Baug,np.eye(4),np.zeros((4,Baug.shape[1])))

BKf = np.concatenate((B,np.atleast_2d(Kf).T),axis=1)
sysKF = ss(A-np.outer(Kf,C),BKf,np.eye(4),np.zeros_like(BKf)
    )
```

Code 8.8 simulates the system with a single output measurement, including additive disturbances and noise, and we use this as the input to a Kalman filter estimator. At times $t = 1$ and $t = 15$, we give the system a large positive and negative impulse in the actuation, respectively.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Code 8.8: [MATLAB] Simulate system and estimate full state.

```
%%  Estimate linearized system in "down" position
dt = .01;
t = dt:dt:50;

uDIST = sqrt(Vd)*randn(4,size(t,2)); % random disturbance
uNOISE = sqrt(Vn)*randn(size(t));    % random noise
u = 0*t;
u(1/dt) = 20/dt;    % positive impulse
u(15/dt) = -20/dt;  % negative impulse

u_aug = [u; uDIST; uNOISE]; % input w/ disturbance and noise

[y,t] = lsim(sysC,u_aug,t);          % noisy measurement
[xtrue,t] = lsim(sysTruth,u_aug,t); % true state
[xhat,t] = lsim(sysKF,[u; y'],t);   % state estimate
```

Code 8.8: [Python] Simulate system and estimate full state.

```
## Estimate linearized system in down position: Gantry crane
dt = 0.01
t = np.arange(0,50,dt)

uDIST = np.sqrt(Vd) @ np.random.randn(4,len(t)) # random
    disturbance
uNOISE = np.sqrt(Vn) * np.random.randn(len(t))     # random
    noise
u = np.zeros_like(t)
u[100] = 20/dt   # positive impulse
u[1500] = -20/dt # negative impulse

# input w/ disturbance and noise:
uAUG = np.concatenate((u.reshape((1,len(u))),uDIST,uNOISE.
    reshape((1,len(uNOISE))))).T

y,t,_ = lsim(sysC,uAUG,t)                          # noisy
    measurement
xtrue,t,_ = lsim(sysTruth,uAUG,t)                  # true state
xhat,t,_ = lsim(sysKF,np.row_stack((u,y)).T,t) # estimate
```

Figure 8.15 shows the noisy measurement signal used by the Kalman filter, and Fig. 8.16 shows the full noiseless state, with disturbances, along with the Kalman filter estimate.

To build intuition, it is recommended that the reader investigate the performance of the Kalman filter when the model is an imperfect representation of the simulated dynamics. When combined with full-state control in the next

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

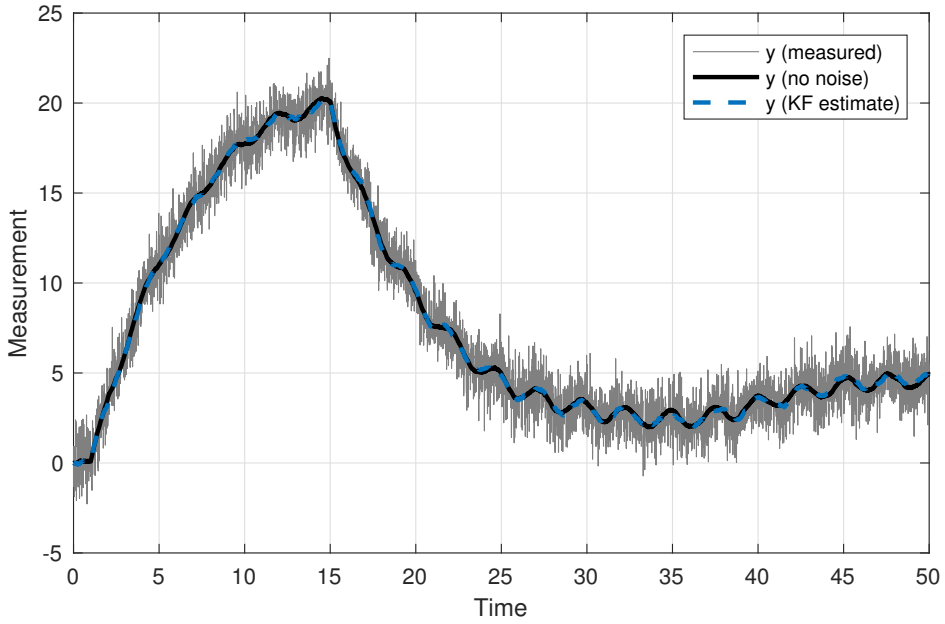


Figure 8.15: Noisy measurement that is used for the Kalman filter, along with the underlying noiseless signal and the Kalman filter estimate.

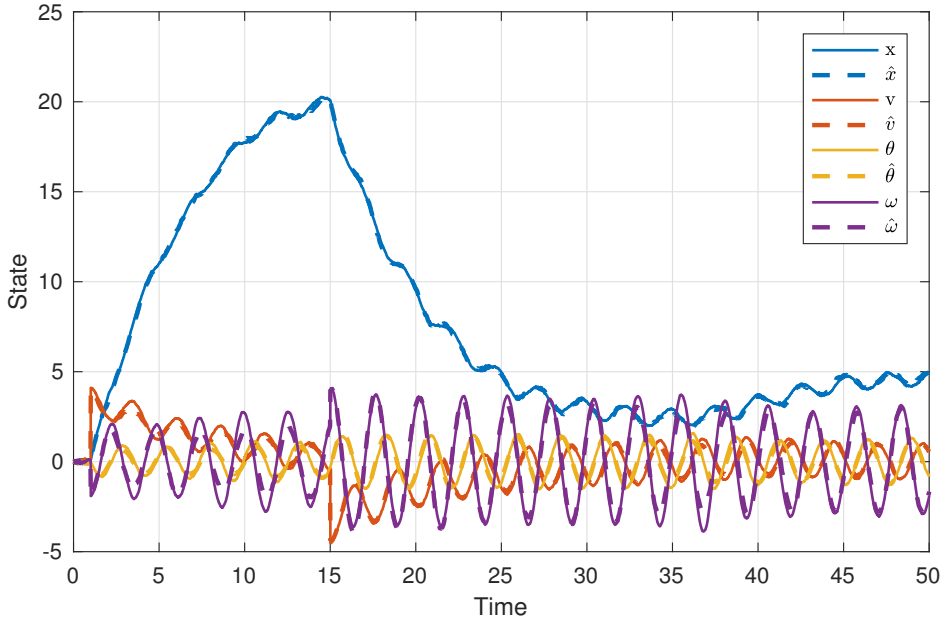


Figure 8.16: The true and Kalman filter estimated states for the pendulum on a cart system.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

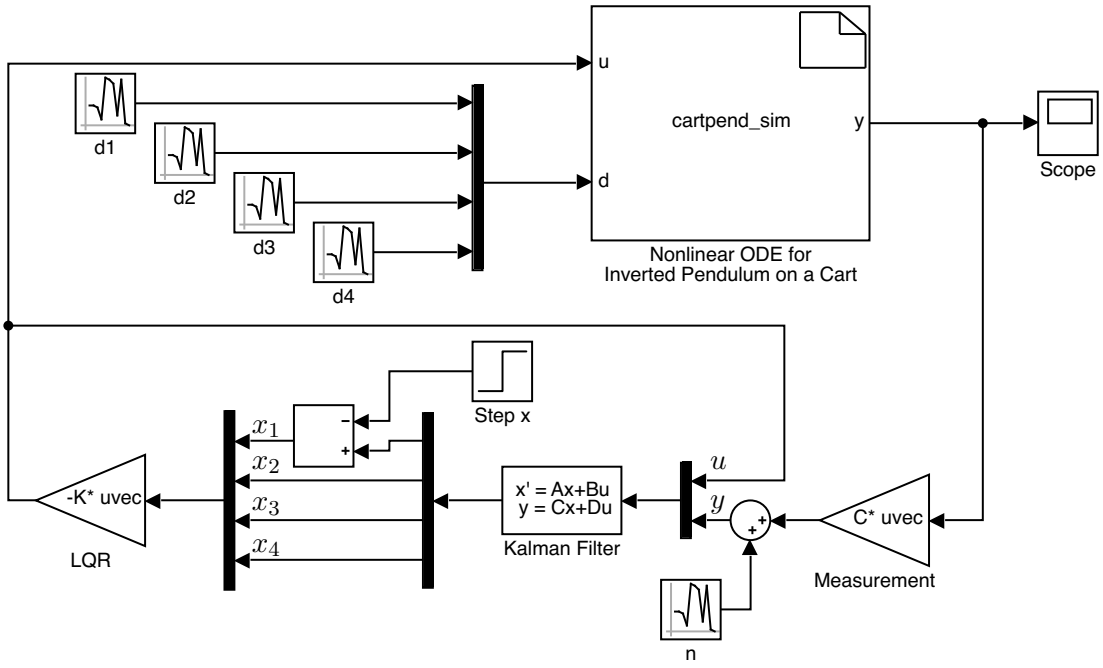


Figure 8.17: MATLAB Simulink model for sensor-based LQG feedback control.

section, small time delays and changes to the system model may cause fragility.

Sensor-Based Feedback Control of the Cart–Pendulum

To apply an LQG regulator to the inverted pendulum on a cart, we will simulate the full nonlinear system in Simulink, as shown in Fig. 8.17. The nonlinear dynamics are encapsulated in the block **cartpend_sim**, and the inputs consist of the actuation signal u and disturbance $\mathbf{w}_d$. We record the full state for performance analysis, although only noisy measurements $y = \mathbf{C}\mathbf{x} + w_n$ and the actuation signal u are passed to the Kalman filter. The full-state estimate is then passed to the LQR block, which commands the desired actuation signal. For this example, we use the following LQR and LQE weighting matrices: $\mathbf{Q} = \mathbf{I}_{4\times4}$, $R = 0.000001$, $\mathbf{V}_d = 0.04\,\mathbf{I}_{4\times4}$, and $V_n = 0.0002$.

The system starts near the vertical equilibrium, at $\mathbf{x}_0 = \begin{bmatrix}0 & 0 & 3.14 & 0\end{bmatrix}^T$, and we command a step in the cart position from $x = 0$ to $x = 1$ at $t = 10$. The resulting response is shown in Fig. 8.18. Despite noisy measurements (Fig. 8.19) and disturbances (Fig. 8.20), the controller is able to effectively track the reference cart position while stabilizing the inverted pendulum.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

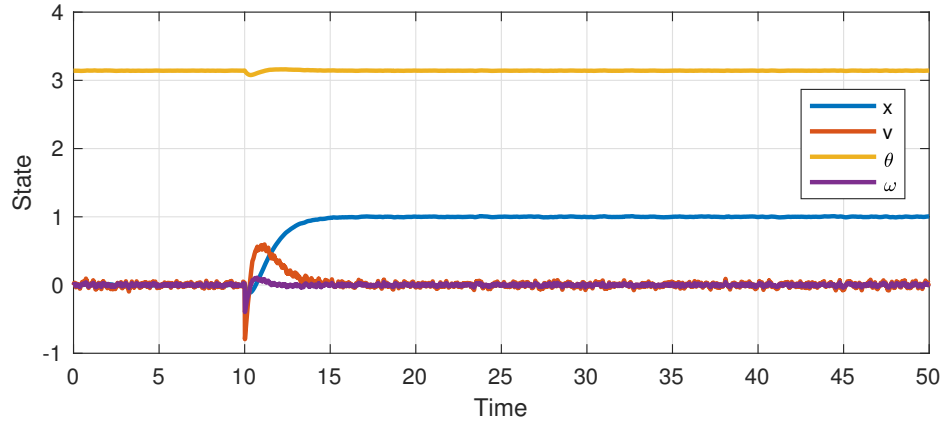


Figure 8.18: Output response using LQG feedback control.

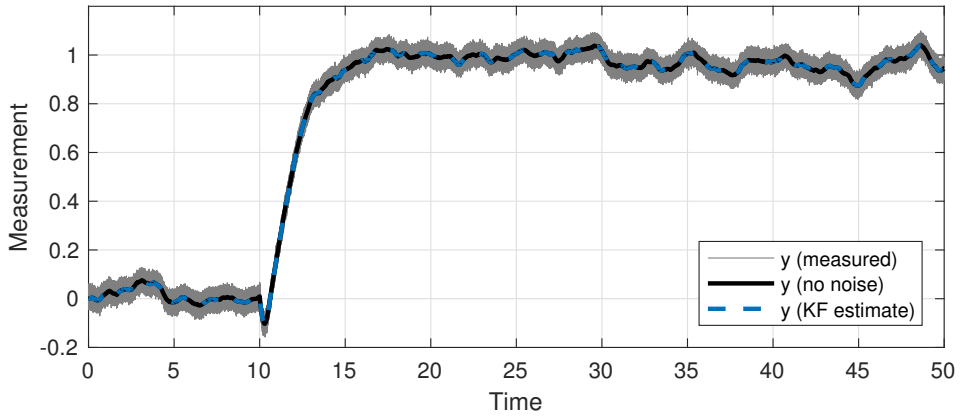


Figure 8.19: Noisy measurement used for the Kalman filter, along with the underlying noiseless signal and the Kalman filter estimate.

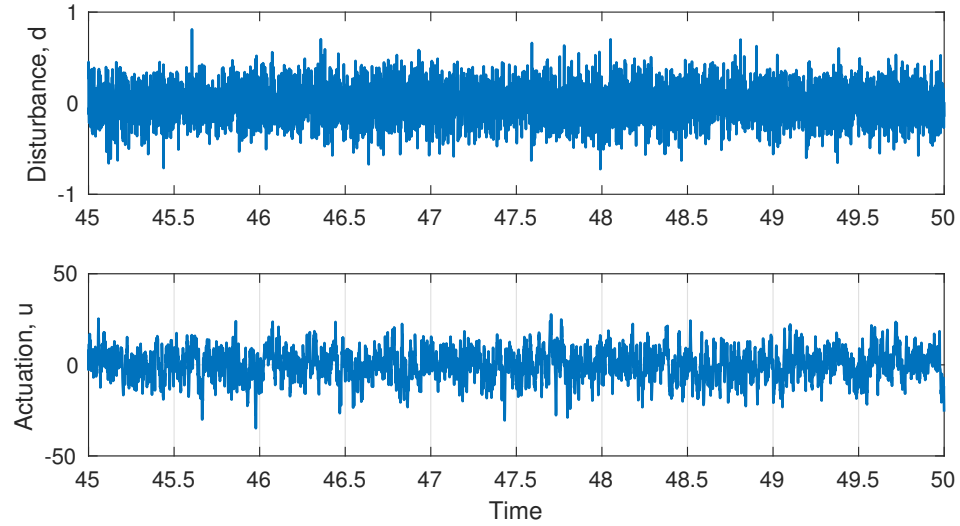


Figure 8.20: Disturbance and actuation signals.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

8.8 Robust Control and Frequency-Domain Techniques

Until now, we have described control systems in terms of state-space systems of ordinary differential equations. This perspective readily lends itself to stability analysis and design via placement of closed-loop eigenvalues. However, in a seminal paper by John Doyle in 1978 [209],[5] it was shown that LQG regulators can have arbitrarily small stability margins, making them *fragile* to model uncertainties, time delays, and other model imperfections.

Fortunately, a short time after Doyle's famous 1978 paper, a rigorous mathematical theory was developed to design controllers that promote robustness. Indeed, this new theory of robust control generalizes the optimal control framework used to develop LQR/LQG, by incorporating a different cost function that penalizes *worse-case scenario* performance.

To understand and design controllers for robust performance, it will be helpful to look at frequency-domain *transfer functions* of various signals. In particular, we will consider the sensitivity, complementary sensitivity, and loop transfer functions. These enable quantitative and visual approaches to assess robust performance, and they enable intuitive and compact representations of control systems.

Robust control is a natural perspective when considering uncertain models obtained from noisy or incomplete data. Moreover, it may be possible to manage system nonlinearity as a form of structured model uncertainty. Finally, we will discuss known factors that limit robust performance, including time delays and non-minimum-phase behavior.

Frequency-Domain Techniques

To understand and manage the tradeoffs between robustness and performance in a control system, it is helpful to design and analyze controllers using frequency-domain techniques.

The Laplace transform allows us to go between the time domain (state space) and frequency domain:

$$\mathcal{L}\{f(t)\} = f(s) = \int_{0^-}^{\infty} f(t)e^{-st}\,\mathrm{d}t. \tag{8.70}$$

Here, s is the complex-valued Laplace variable. The Laplace transform may be thought of as a one-sided generalized Fourier transform that is valid for functions that do not converge to zero as $t \to \infty$. The Laplace transform is particularly useful because it transforms differential equations into algebraic equations, and convolution integrals in the time domain become simple products in the frequency domain. To see how time derivatives pass through the

[5] Title: *Guaranteed margins for LQG regulators*. "Abstract: There are none."

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Laplace transform, we use integration by parts:

$$\begin{aligned}\mathcal{L}\left\{\frac{\mathrm{d}}{\mathrm{d}t}f(t)\right\} &= \int_{0^-}^{\infty} \underbrace{\frac{\mathrm{d}}{\mathrm{d}t}f(t)}_{\mathrm{d}v}\underbrace{e^{-st}}_{u}\,\mathrm{d}t \\ &= \Big[f(t)e^{-st}\Big]_{t=0^-}^{t=\infty} - \int_{0^-}^{\infty} f(t)(-se^{-st})\,\mathrm{d}t \\ &= f(0^-) + s\mathcal{L}\{f(t)\}.\end{aligned}$$

Thus, for zero initial conditions, $\mathcal{L}\{\mathrm{d}f/\mathrm{d}t\} = sf(s)$.

Taking the Laplace transform of the control system in (8.10) yields

$$s\mathbf{x}(s) = \mathbf{A}\mathbf{x}(s) + \mathbf{B}\mathbf{u}(s), \tag{8.71a}$$
$$\mathbf{y}(s) = \mathbf{C}\mathbf{x}(s) + \mathbf{D}\mathbf{u}(s). \tag{8.71b}$$

It is possible to solve for $\mathbf{x}(s)$ in the first equation, as

$$(s\mathbf{I} - \mathbf{A})\mathbf{x}(s) = \mathbf{B}\mathbf{u}(s) \quad \Longrightarrow \quad \mathbf{x}(s) = (s\mathbf{I} - \mathbf{A})^{-1}\mathbf{B}\mathbf{u}(s). \tag{8.72}$$

Substituting this into the second equation, we arrive at a mapping from inputs $\mathbf{u}$ to outputs $\mathbf{y}$:

$$\mathbf{y}(s) = [\mathbf{C}(s\mathbf{I} - \mathbf{A})^{-1}\mathbf{C} + \mathbf{D}]\mathbf{u}(s). \tag{8.73}$$

We define this mapping as the *transfer function*:

$$\mathbf{G}(s) = \frac{\mathbf{y}(s)}{\mathbf{u}(s)} = \mathbf{C}(s\mathbf{I} - \mathbf{A})^{-1}\mathbf{B} + \mathbf{D}. \tag{8.74}$$

For linear systems, there are three equivalent representations: (1) time domain, in terms of the impulse response; (2) frequency domain, in terms of the transfer function; and (3) state space, in terms of a system of differential equations. These representations are shown schematically in Fig. 8.21. As we will see, there are many benefits to analyzing control systems in the frequency domain.

Frequency Response

The transfer function in (8.74) is particularly useful because it gives rise to the frequency response, which is a graphical representation of the control system in terms of measurable data. To illustrate this, we will consider a single-input, single-output (SISO) system. It is a property of linear systems with zero initial conditions that a sinusoidal input will give rise to a sinusoidal output with the same frequency, perhaps with a different magnitude A and phase ϕ:

$$u(t) = \sin(\omega t) \quad \Longrightarrow \quad y(t) = A\sin(\omega t + \phi). \tag{8.75}$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

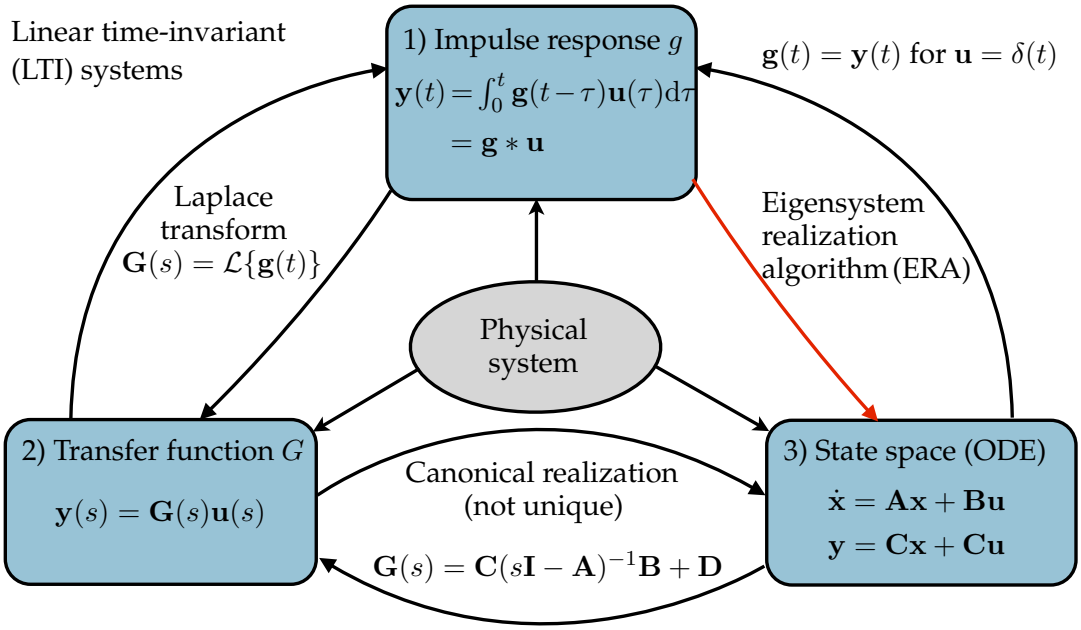


Figure 8.21: Three equivalent representations of linear time-invariant systems.

This is true for long times, after initial transients die out. The amplitude A and phase ϕ of the output sinusoid depend on the input frequency ω. These functions $A(\omega)$ and $\phi(\omega)$ may be mapped out by running a number of experiments with sinusoidal input at different frequencies ω. Alternatively, this information is obtained from the complex-valued transfer function $G(s)$:

$$A(\omega) = |G(i\omega)|, \quad \phi(\omega) = \angle G(i\omega). \tag{8.76}$$

Thus, the amplitude and phase angle for input $\sin(\omega t)$ may be obtained by evaluating the transfer function at $s = i\omega$ (i.e., along the imaginary axis in the complex plane). These quantities may then be plotted, resulting in the *frequency response* or *Bode plot*.

For a concrete example, consider the spring–mass–damper system, shown in Fig. 8.22. The equations of motion are given by

$$m\ddot{x} = -\delta\dot{x} - kx + u. \tag{8.77}$$

Choosing values $m = 1$, $\delta = 1$, $k = 2$, and taking the Laplace transform yields

$$G(s) = \frac{1}{s^2 + s + 2}. \tag{8.78}$$

Here we are assuming that the output y is a measurement of the position x of the mass. Note that the denominator of the transfer function $G(s)$ is the characteristic equation of (8.77), written in state-space form. Thus, the poles of the complex function $G(s)$ are eigenvalues of the state-space system.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

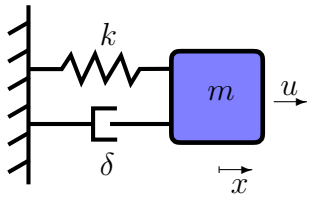


Figure 8.22: Spring–mass–damper system.

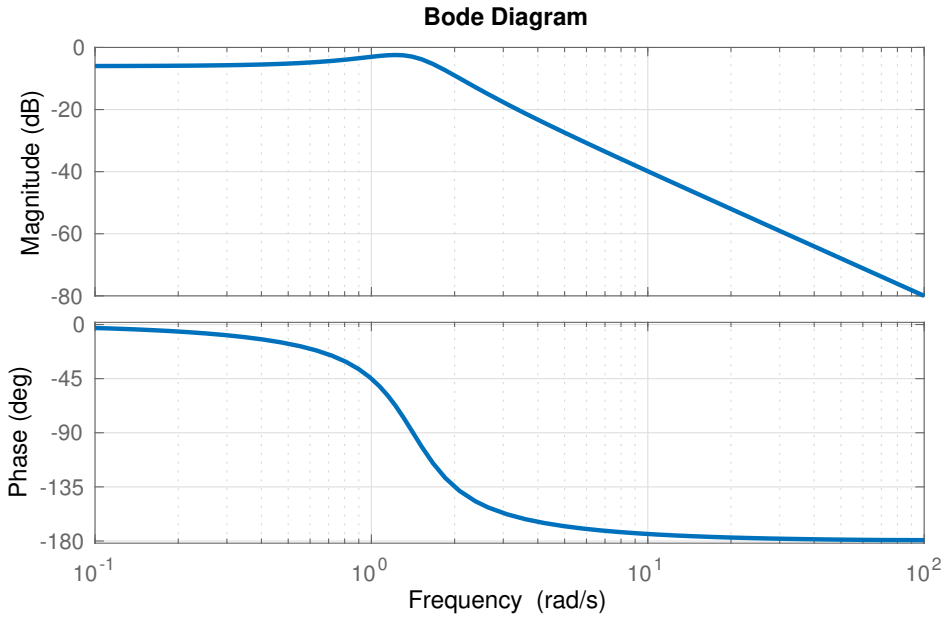


Figure 8.23: Frequency response of spring–mass–damper system. The magnitude is plotted on a logarithmic scale, in units of decibel (dB), and the frequency is likewise on a log scale.

It is now possible to create this system and plot the frequency response (i.e., the Bode plot), as shown in Fig. 8.23 and computed in Code 8.9. Note that the frequency response is readily interpretable and provides physical intuition. For example, the zero slope of the magnitude at low frequencies indicates that slow forcing translates directly into motion of the mass, while the roll-off of the magnitude at high frequencies indicates that fast forcing is attenuated and does not significantly affect the motion of the mass. Moreover, the resonance frequency is seen as a peak in the magnitude, indicating an amplification of forcing at this frequency. Code 8.9 also shows how to manipulate state-space realizations into

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

frequency-domain representations, and vice versa.

Code 8.9: [MATLAB] Create transfer function and plot frequency response (Bode) plot. Convert between state-space and frequency-domain representations.

```
s = tf('s');           % Laplace variable
G = 1/(s^2 + s + 2);   % Transfer function
bode(G);               % Frequency response

% State space realization
A = [0 1; -2 -1];
B = [0; 1];
C = [1 0];
D = 0;

% Convert to frequency domain
[num,den] = ss2tf(A,B,C,D);
G = tf(num,den)

% Convert back to state space
[A,B,C,D] = tf2ss(G.num{1},G.den{1})
```

Code 8.9: [Python] Create transfer function and plot frequency response (Bode) plot. Convert between state-space and frequency-domain representations.

```
s = tf(np.array([1,0]),np.array([0,1]))
G = 1/(s**2 + s + 2)
w, mag, phase = bode(G)

# State space realization
A = np.array([[0,1],[-2,-1]])
B = np.array([0,1]).reshape((2,1))
C = np.array([1,0])
D = 0

# Convert to frequency domain
G = ss2tf(A,B,C,D) # returns transfer function

# Convert back to state space
sysSS= tf2ss(G)    # returns state space system
```

Note that the state-space representation is not unique, and going from a transfer function to state space may switch the order of the state variables.

The frequency domain is also useful because impulsive or step inputs are particularly simple to represent with the Laplace transform. The impulse response (Fig. 8.24) and step response (Fig. 8.25) are computed in MATLAB by

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

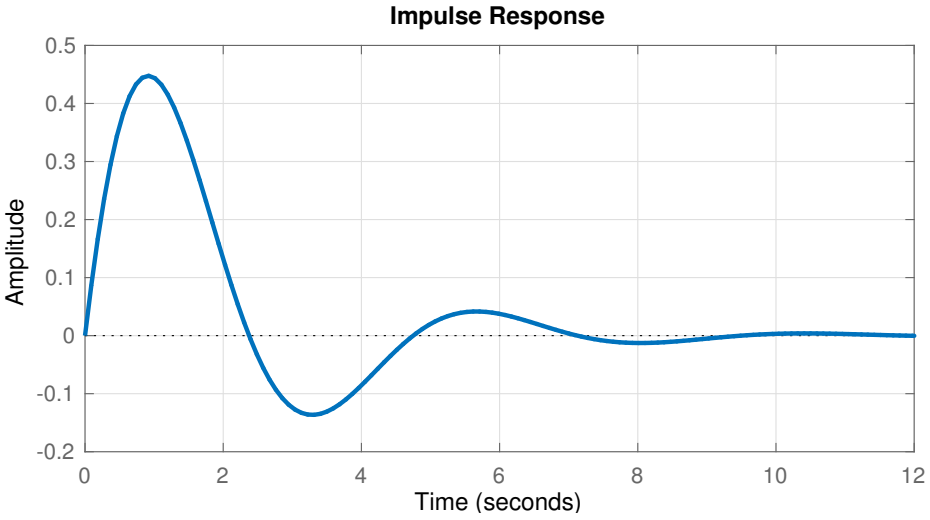


Figure 8.24: Impulse response of spring–mass–damper system.

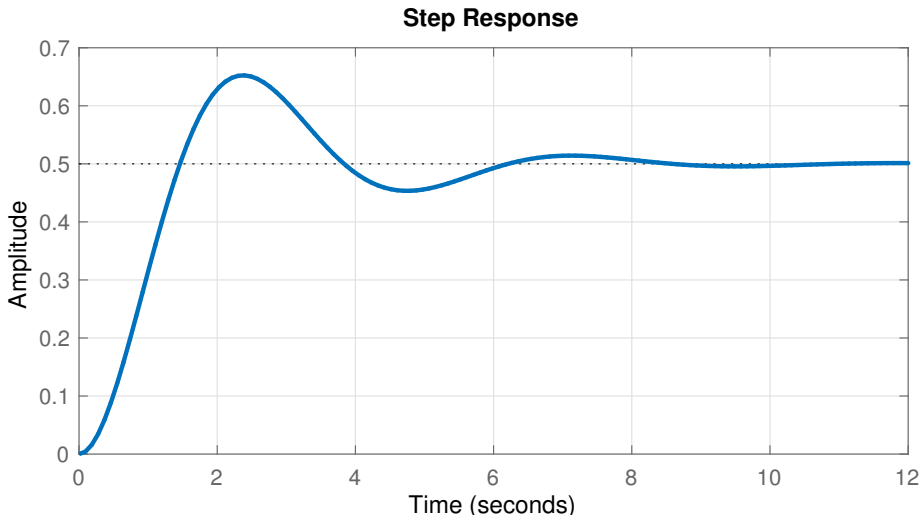


Figure 8.25: Step response of spring–mass–damper system.

```
>> impulse(G);    % Impulse response
>> step(G);       % Step response
```

and in python-control by

```
>>> ia,it = impulse(G) # Impulse response
>>> plt.plot(it,ia)    # Need to plot output of impulse
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

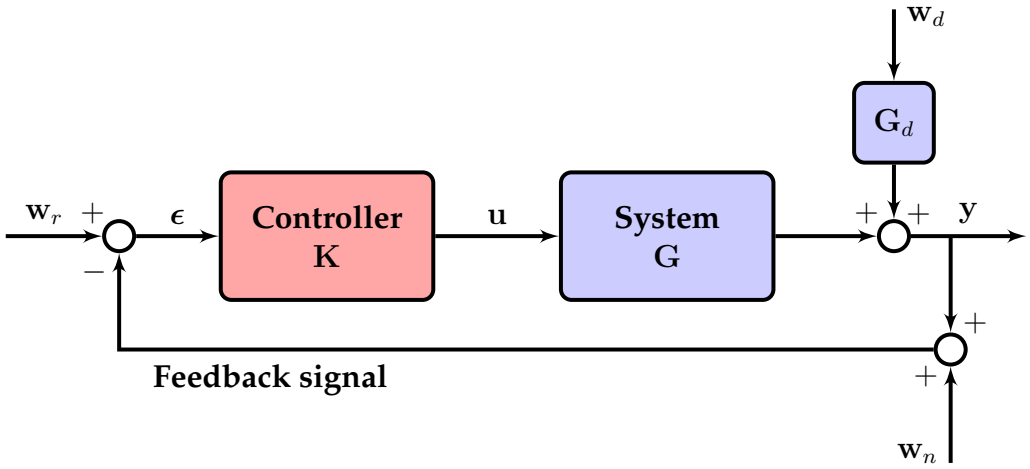


Figure 8.26: Closed-loop feedback control diagram with reference input, noise, and disturbance. We will consider the various transfer functions from exogenous inputs to the error ϵ, thus deriving the loop transfer function, as well as the sensitivity and complementary sensitivity functions.

```
>>> ia,it = step(G)     # Step response
>>> plt.plot(it,ia)     # Need to plot output of step
```

Performance and the Loop Transfer Function: Sensitivity and Complementary Sensitivity

Consider a slightly modified version of Fig. 8.4, where the disturbance has a model, $\mathbf{P}_d$. This new diagram, shown in Fig. 8.26, will be used to derive the important transfer functions relevant for assessing robust performance:

$$\mathbf{y} = \mathbf{GK}(\mathbf{w}_r - \mathbf{y} - \mathbf{w}_n) + \mathbf{G}_d\mathbf{w}_d \tag{8.79a}$$

$$\implies (\mathbf{I} + \mathbf{GK})\mathbf{y} = \mathbf{GK}\mathbf{w}_r - \mathbf{GK}\mathbf{w}_n + \mathbf{G}_d\mathbf{w}_d \tag{8.79b}$$

$$\implies \mathbf{y} = \underbrace{(\mathbf{I} + \mathbf{GK})^{-1}\mathbf{GK}}_{\mathbf{T}}\mathbf{w}_r - \underbrace{(\mathbf{I} + \mathbf{GK})^{-1}\mathbf{GK}}_{\mathbf{T}}\mathbf{w}_n + \underbrace{(\mathbf{I} + \mathbf{GK})^{-1}}_{\mathbf{S}}\mathbf{G}_d\mathbf{w}_d. \tag{8.79c}$$

Here, $\mathbf{S}$ is the *sensitivity function* and $\mathbf{T}$ is the *complementary sensitivity function*. We may denote $\mathbf{L} = \mathbf{GK}$, the *loop transfer function*, shown in Fig. 8.27, which is the open-loop transfer function in the absence of feedback. Both $\mathbf{S}$ and $\mathbf{T}$ may

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

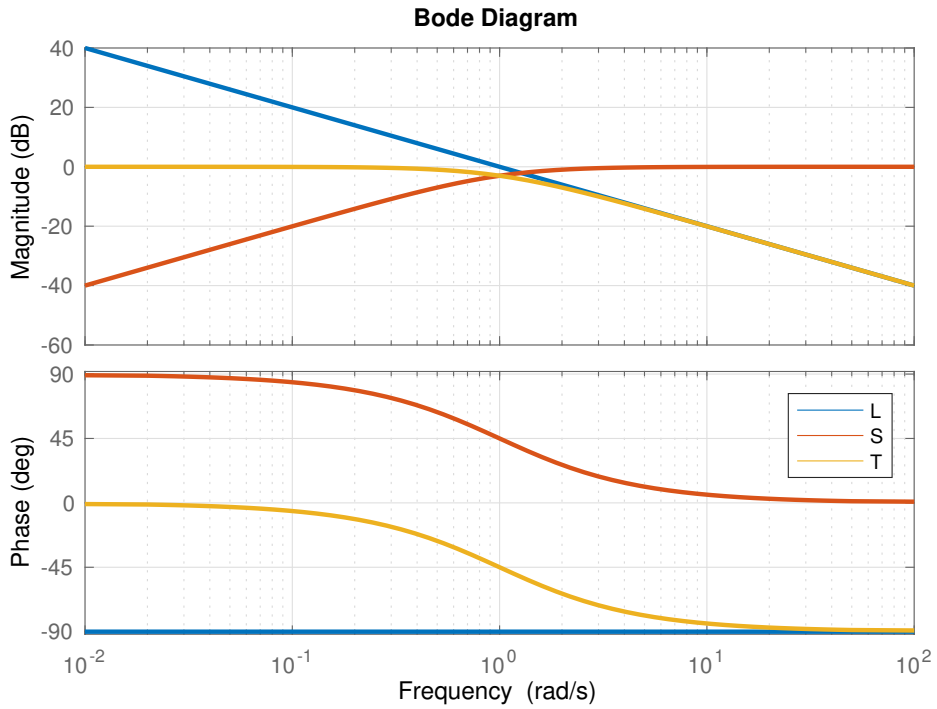


Figure 8.27: Loop transfer function $\mathbf{L}$ along with sensitivity $\mathbf{S}$ and complementary sensitivity $\mathbf{T}$ functions.

be simplified in terms of $\mathbf{L}$:

$$\mathbf{S} = (\mathbf{I} + \mathbf{L})^{-1}, \tag{8.80a}$$

$$\mathbf{T} = (\mathbf{I} + \mathbf{L})^{-1}\mathbf{L}. \tag{8.80b}$$

Conveniently, the sensitivity and complementary sensitivity functions must add up to the identity: $\mathbf{S} + \mathbf{T} = \mathbf{I}$.

In practice, the transfer function from the exogenous inputs to the noiseless error $\boldsymbol{\epsilon}$ is more useful for design:

$$\boldsymbol{\epsilon} = \mathbf{w}_r - \mathbf{y} = \mathbf{S}\mathbf{w}_r + \mathbf{T}\mathbf{w}_n - \mathbf{S}\mathbf{G}_d\mathbf{w}_d. \tag{8.81}$$

Thus, we see that the sensitivity and complementary sensitivity functions provide the maps from reference, disturbance, and noise inputs to the tracking error. Since we desire small tracking error, we may then specify $\mathbf{S}$ and $\mathbf{T}$ to have desirable properties, and ideally we will be able to achieve these specifications by designing the loop transfer function $\mathbf{L}$. In practice, we will choose the controller $\mathbf{K}$ with knowledge of the model $\mathbf{G}$ so that the loop transfer function has

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

beneficial properties in the frequency domain. For example, small gain at high frequencies will attenuate sensor noise, since this will result in $\mathbf{T}$ being small. Similarly, high gain at low frequencies will provide good reference tracking performance, as $\mathbf{S}$ will be small at low frequencies. However, $\mathbf{S}$ and $\mathbf{T}$ cannot both be small everywhere, since $\mathbf{S} + \mathbf{T} = \mathbf{I}$, from (8.80), and so these design objectives may compete.

For performance and robustness, we want the maximum peak of $\mathbf{S}$, $M_{\mathbf{S}} = \|\mathbf{S}\|_\infty$, to be as small as possible. From (8.81), it is clear that, in the absence of noise, feedback control improves performance (i.e., reduces error) for all frequencies where $|\mathbf{S}| < 1$; thus control is effective when $\mathbf{T} \approx \mathbf{1}$. As explained in [665, p. 37], all real systems will have a range of frequencies where $|\mathbf{S}| > 1$, in which case performance is degraded. Minimizing the peak $M_{\mathbf{S}}$ mitigates the amount of degradation experienced with feedback at these frequencies, improving performance. In addition, the minimum distance of the loop transfer function $\mathbf{L}$ to the point -1 in the complex plane is given by $M_{\mathbf{S}}^{-1}$. By the Nyquist stability theorem, the larger this distance, the greater the stability margin of the closed-loop system, improving robustness. These are the two major reasons to minimize $M_{\mathbf{S}}$.

The controller *bandwidth* ω_B is the frequency below which feedback control is effective. This is a subjective definition. Often, ω_B is the frequency where $|\mathbf{S}(j\omega)|$ first crosses $-3\,\text{dB}$ from below. We would ideally like the controller bandwidth to be as large as possible without amplifying sensor noise, which typically has a high frequency. However, there are fundamental bandwidth limitations that are imposed for systems that have time delays or right half-plane zeros [665].

Inverting the Dynamics

With a model of the form in (8.10) or (8.73), it may be possible to design an open-loop control law to achieve some desired specification without the use of measurement-based feedback or feedforward control. For instance, if perfect tracking of the reference input $\mathbf{w}_r$ is desired in Fig. 8.3, under certain circumstances it may be possible to design a controller by inverting the system dynamics $\mathbf{G}$, i.e., $\mathbf{K}(s) = \mathbf{G}^{-1}(s)$. In this case, the transfer function from reference $\mathbf{w}_r$ to output $\mathbf{s}$ is given by $\mathbf{G}\mathbf{G}^{-1} = \mathbf{1}$, so that the output perfectly matches the reference. However, perfect control is never possible in real-world systems, and this strategy should be used with caution, since it generally relies on a number of significant assumptions on the system $\mathbf{G}$. First, effective control based on inversion requires extremely precise knowledge of $\mathbf{G}$ and well-characterized, predictable disturbances; there is little room for model errors or uncertainties, as there are no sensor measurements to determine if performance is as expected and no corrective feedback mechanisms to modify the actuation strategy to

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

compensate.

For open-loop control using system inversion, $\mathbf{G}$ must also be stable. It is impossible to fundamentally change the dynamics of a linear system through open-loop control, and thus an unstable system cannot be stabilized without feedback. Attempting to stabilize an unstable system by inverting the dynamics will typically have disastrous consequences. For instance, consider the following unstable system with a pole at $s = 5$ and a zero at $s = -10$:

$$G(s) = (s + 10)/(s - 5).$$

Inverting the dynamics would result in a controller

$$K = (s - 5)/(s + 10).$$

However, if there is even the slightest uncertainty in the model, so that the true pole is at $5 - \epsilon$, then the open-loop system will be

$$G_{\text{true}}(s)K(s) = (s - 5)/(s - 5 + \epsilon).$$

This system is still unstable, despite the attempted pole cancelation. Moreover, the unstable mode is now nearly unobservable.

In addition to stability, $\mathbf{G}$ must not have any time delays or zeros in the right half-plane, and it must have the same number of poles as zeros. If $\mathbf{G}$ has any zeros in the right half-plane, then the inverted controller $\mathbf{K}$ will be unstable, since it will have right half-plane poles. These systems are called *non-minimum-phase*, and there have been generalizations to dynamic inversion that provide bounded inverses to these systems [203]. Similarly, time delays are not invertible, and if $\mathbf{G}$ has more poles than zeros, then the resulting controller will not be realizable and may have extremely large actuation signals $\mathbf{b}$. There are also generalizations that provide *regularized* model inversion, where optimization schemes are applied with penalty terms added to keep the resulting actuation signal $\mathbf{b}$ bounded. These regularized open-loop controllers are often significantly more effective, with improved robustness.

Combined, these restrictions on $\mathbf{G}$ imply that model-based open-loop control should only be used when the system is well behaved, accurately characterized by a model, when disturbances are characterized, and when the additional feedback control hardware is unnecessarily expensive. Otherwise, performance goals must be modest. Open-loop model inversion is often used in manufacturing and robotics, where systems are well characterized and constrained in a standard operating environment.

Robust Control

As discussed above, LQG controllers are known to have arbitrarily poor robustness margins. This is a serious problem in systems such as turbulence control,

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

neuromechanical systems, and epidemiology, where the dynamics are wrought with uncertainty and time delays.

Figure 8.2 shows the most general schematic for closed-loop feedback control, encompassing both optimal and robust control strategies. In the generalized theory of modern control, the goal is to minimize the transfer function from exogenous inputs $\mathbf{w}$ (reference, disturbances, noise, etc.) to a multi-objective cost function $\mathbf{J}$ (accuracy, actuation cost, time-domain performance, etc.). Optimal control (e.g., LQR, LQE, LQG) is optimal with respect to the $\mathcal{H}_2$-norm, a bounded 2-norm on a Hardy space, consisting of stable and strictly proper transfer functions (meaning gain rolls off at high frequency). Robust control is similarly optimal with respect to the $\mathcal{H}_\infty$ bounded infinity-norm, consisting of stable and proper transfer functions (gain does not grow infinite at high frequencies). The infinity-norm is defined as

$$\|\mathbf{G}\|_\infty \triangleq \max_\omega \sigma_1(\mathbf{G}(i\omega)). \tag{8.82}$$

Here, σ_1 denotes the maximum singular value. Since the $\|\cdot\|_\infty$-norm is the maximum value of the transfer function at any frequency, it is often called a *worst-case scenario norm*; therefore, minimizing the infinity-norm provides robustness to worst-case exogenous inputs. $\mathcal{H}_\infty$ robust controllers are used when robustness is important. There are many connections between $\mathcal{H}_2$ and $\mathcal{H}_\infty$ control, as they exist within the same framework and simply optimize different norms. We refer the reader to the excellent reference books expanding on this theory [222, 665].

If we let $\mathbf{G}_{\mathbf{w}\to\mathbf{J}}$ denote the transfer function from $\mathbf{w}$ to $\mathbf{J}$, then the goal of $\mathcal{H}_\infty$ control is to construct a controller to minimize the infinity-norm: $\min \|\mathbf{G}_{\mathbf{w}\to\mathbf{J}}\|_\infty$. This is typically difficult, and no analytic closed-form solution exists for the optimal controller in general. However, there are relatively efficient iterative methods to find a controller such that $\|\mathbf{G}_{\mathbf{w}\to\mathbf{J}}\|_\infty < \gamma$, as described in [211]. There are numerous conditions and caveats that describe when this method can be used. In addition, there are computationally efficient algorithms implemented in both MATLAB and Python, and these methods require relatively low overhead from the user.

Selecting the cost function $\mathbf{J}$ to meet design specifications is a critically important part of robust control design. Considerations such as disturbance rejection, noise attenuation, controller bandwidth, and actuation cost may be accounted for by a weighted sum of the transfer functions $\mathbf{S}$, $\mathbf{T}$, and $\mathbf{KS}$. In the *mixed sensitivity* control problem, various weighting transfer functions are used to balance the relative importance of these considerations at various frequency ranges. For instance, we may weight $\mathbf{S}$ by a low-pass filter and $\mathbf{KS}$ by a high-pass filter, so that disturbance rejection at low frequency is promoted and control response at high frequency is discouraged. A general cost function may

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

consist of three weighting filters $\mathbf{F}_k$ multiplying $\mathbf{S}$, $\mathbf{T}$, and $\mathbf{KS}$:

$$\left\| \begin{bmatrix} \mathbf{F}_1\mathbf{S} \\ \mathbf{F}_2\mathbf{T} \\ \mathbf{F}_3\mathbf{KS} \end{bmatrix} \right\|_\infty .$$

Another possible robust control design is called $\mathcal{H}_\infty$ loop shaping. This procedure may be more straightforward than mixed sensitivity synthesis for many problems. The loop shaping method consists of two major steps. First, a desired open-loop transfer function is specified based on performance goals and classical control design. Second, the shaped loop is made robust with respect to a large class of model uncertainty. Indeed, the procedure of $\mathcal{H}_\infty$ loop shaping allows the user to design an ideal controller to meet performance specifications, such as rise-time, bandwidth, settling-time, etc. Typically, a loop shape should have large gain at low frequency to guarantee accurate reference tracking and slow disturbance rejection, low gain at high frequencies to attenuate sensor noise, and a crossover frequency that ensures desirable bandwidth. The loop transfer function is then robustified so that there are improved gain and phase margins.

$\mathcal{H}_2$ optimal control (e.g., LQR, LQE, LQG) has been an extremely popular control paradigm because of its simple mathematical formulation and its tunability by user input. However, the advantages of $\mathcal{H}_\infty$ control are being increasingly realized. Additionally, there are numerous consumer software solutions that make implementation relatively straightforward. In MATLAB, mixed sensitivity is accomplished using the **mixsyn** command in the robust control toolbox. Similarly, loop shaping is accomplished using the **loopsyn** command in the robust control toolbox.

Fundamental Limitations on Robust Performance

As discussed above, we want to minimize the peaks of $\mathbf{S}$ and $\mathbf{T}$ to improve robustness. Some peakedness is inevitable, and there are certain system characteristics that significantly limit performance and robustness. Most notably, time delays and right half-plane zeros of the open-loop system will limit the effective control bandwidth and will increase the attainable lower bound for peaks of $\mathbf{S}$ and $\mathbf{T}$. This contributes to both degrading performance and decreasing robustness.

Similarly, a system will suffer from robust performance limitations if the number of poles exceeds the number of zeros by more than two. These fundamental limitations are quantified in the *waterbed* integrals, which are so named because if you push a waterbed down in one location, it must rise in another. Thus, there are limits to how much one can push down peaks in $\mathbf{S}$ without causing other peaks to pop up.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Time delays are relatively easy to understand, since a time delay τ will introduce an additional phase lag of $\tau\omega$ at the frequency ω, limiting how fast the controller can respond effectively (i.e., bandwidth). Thus, the bandwidth for a controller with acceptable phase margins is typically $\omega_B < 1/\tau$.

Following the discussion in [665], these fundamental limitations may be understood in relation to the limitations of open-loop control based on model inversion. If we consider high-gain feedback $\mathbf{u} = \mathbf{K}(\mathbf{w}_r - \mathbf{y})$ for a system as in Fig. 8.26 and (8.81), but without disturbances or noise, we have

$$\mathbf{u} = \mathbf{K}\boldsymbol{\epsilon} = \mathbf{K}\mathbf{S}\mathbf{w}_r. \tag{8.83}$$

We may write this in terms of the complementary sensitivity $\mathbf{T}$, by noting that since $\mathbf{T} = \mathbf{I} - \mathbf{S}$, we have $\mathbf{T} = \mathbf{L}(\mathbf{I} + \mathbf{L})^{-1} = \mathbf{G}\mathbf{K}\mathbf{S}$:

$$\mathbf{u} = \mathbf{G}^{-1}\mathbf{T}\mathbf{w}_r. \tag{8.84}$$

Thus, at frequencies where $\mathbf{T}$ is nearly the identity $\mathbf{I}$ and control is effective, the actuation is effectively inverting $\mathbf{G}$. Even with sensor-based feedback, perfect control is unattainable. For example, if $\mathbf{G}$ has right half-plane zeros, then the actuation signal will become unbounded if the gain $\mathbf{K}$ is too aggressive. Similarly, limitations arise with time delays and when the number of poles of $\mathbf{G}$ exceeds the number of zeros, as in the case of open-loop model-based inversion.

As a final illustration of the limitation of right half-plane zeros, we consider the case of proportional control $u = Ky$ in a SISO system with $G(s) = N(s)/D(s)$. Here, roots of the numerator $N(s)$ are zeros and roots of the denominator $D(s)$ are poles. The closed-loop transfer function from reference w_r to sensors s is given by

$$\frac{y(s)}{w_r(s)} = \frac{GK}{1 + GK} = \frac{NK/D}{1 + NK/D} = \frac{NK}{D + NK}. \tag{8.85}$$

For small control gain K, the term NK in the denominator is small, and the poles of the closed-loop system are near the poles of G, given by roots of D. As K is increased, the NK term in the denominator begins to dominate, and closed-loop poles are attracted to the roots of N, which are the open-loop zeros of G. Thus, if there are right half-plane zeros of the open-loop system G, then high-gain proportional control will drive the system unstable. These effects are often observed in the root locus plot from classical control theory. In this way, we see that right half-plane zeros will directly impose limitations on the gain margin of the controller.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Suggested Reading

Texts

(1) Feedback systems: An introduction for scientists and engineers, by K. J. Aström and R. M. Murray, 2010 [30].

(2) Feedback control theory, by J. C. Doyle, B. A. Francis, and A. R. Tannenbaum, 2013 [210].

(3) Multivariable feedback control: Analysis and design, by S. Skogestad and I. Postlethwaite, 2005 [665].

(4) A course in robust control theory: A convex approach, by G. E. Dullerud and F. Paganini, 2000 [222].

(5) Optimal control and estimation, by R. F. Stengel, 2012 [675].

Papers and reviews

(1) Guaranteed margins for LQG regulators, by J. C. Doyle, *IEEE Transactions on Automatic Control*, 1978 [209].

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Homework

Exercise 8-1. Give an example of a control system in your daily life. Describe the inputs and outputs. What are the system dynamics? What are the control objectives?

Exercise 8-2. This example will explore the optimal control workflow on a rotary inverted pendulum.

(a) Derive the equations of motion for a rotary pendulum, where the base of the pendulum is mounted to a rotating arm. The control input is a torque input to the rotor arm.

(b) Identify the fixed points of the system and linearize about each fixed point. What is the stability of each fixed point? Determine the linear controllability of each fixed point.

(c) Design an LQR controller for the pendulum-up configuration assuming full-state measurements.

(d) Determine the observability of the pendulum-up configuration if we cannot measure the full state, but instead measure the pendulum angle and the rotor angle. Similarly, determine the observability if we only measure the pendulum angular rate and the rotor angle. Which sensor configuration is *more* observable? Pick at least one different sensor set and assess if this configuration is observable.

(e) Assuming a measurement of the pendulum angle and rotor angle, design a Kalman filter to estimate the full state. Design for disturbance magnitude 1×10^{-3} and sensor noise 1×10^{-2}. Simulate the noisy system and compare the Kalman filter estimate with the true state without added noise.

(f) Design an LQG controller for the pendulum-up configuration and demonstrate this controller in simulation. How does the controller perform when you introduce a small time delay? At what point does the time delay cause the system to go unstable?

Exercise 8-3. Derive the equations of motion for a double pendulum on a cart.

Repeat each step above for the double pendulum.

Exercise 8-4. This exercise will design a controller to move a cart with a pendulum in the down position from one point on a track to another. The goal will be to move quickly from point $x = 0$ to point $x = 1$ while minimizing the

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

amount the pendulum swings, which corresponds to the problem faced when designing a controller for a gantry crane. What will make this example more interesting is that it is a *non-minimum-phase* system.

First, try using a simple proportional feedback controller, with gain K, where the control input is proportional to the error between the state and the goal. Plot the poles and zeros of the closed-loop system for a range of K, and explain the results.

Design a full-state LQR controller to track a reference position of the cart, while minimizing the pendulum swinging. Try different LQR gain matrices, and compare the step response when setting the reference state from $x = 0$ to $x = 1$.

Exercise 8-5. Generate two different state-space realizations for the following transfer function:

$$G(s) = \frac{1}{s^2 + 3s + 2}.$$

Find a coordinate transformation between the states of the two systems.

Exercise 8-6. This example will explore how an ill-conditioned controllability Gramian can affect state-feedback control.

(a) Create a continuous-time, single-input state-space system that has $n = 2$ states and full-state measurements (i.e., $\mathbf{C} = \mathbf{I}$). Design the system to be technically controllable, yet with one of the directions being much more controllable than the other (i.e., 10^6 times more controllable). Compute the controllability Gramian for this system and compute its eigendecomposition. Explain what you find.

(b) Design an LQR controller for this system using weight matrices $\mathbf{Q} = \mathbf{I}$ and $R = 1$. Simulate the response of the closed-loop system, and explain the results.

(c) Using this controller, initialize the system at thousands of points randomly sampled from a unit circle where $\|\mathrm{x}\| = 1$, and compute the LQR cost J for each of these trajectories. Plot the initial conditions on the sphere, color-coded by the cost J. Reconcile this plot with the controllability Gramian you computed earlier.

Exercise 8-7. For the inverted pendulum on a cart, we will analyze the sensitivity and robustness of LQG control. Plot the sensitivity and complementary sensitivity for this system. What are the limits of robustness?

Robustify the LQG controller using loop synthesis. Compare the robustness before and after.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

For the LQG controller, introduce a small time delay to the system and characterize what changes in the control response and performance. At what size time delay does the system go unstable? Determine the units for a realistic-sized pendulum.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Chapter 9

Balanced Models for Control

Many systems of interest are exceedingly high-dimensional, making them difficult to characterize. High dimensionality also limits controller robustness due to significant computational time delays. For example, for the governing equations of fluid dynamics, the resulting discretized equations may have millions or billions of degrees of freedom, making them expensive to simulate. Thus, significant effort has gone into obtaining reduced-order models that capture the most relevant mechanisms and are suitable for feedback control.

Unlike reduced-order models based on proper orthogonal decomposition (see Chapters 12 and 13), which order modes based on energy content in the data, here we will discuss a class of *balanced* reduced-order models that employ a different inner product to order modes based on input–output energy. Thus, only modes that are both highly controllable and highly observable are selected, making balanced models ideal for control applications. In this chapter we also describe related procedures for model reduction and system identification, depending on whether or not the user starts with a high-fidelity model or simply has access to measurement data.

9.1 Model Reduction and System Identification

In many nonlinear systems, it is still possible to use linear control techniques. For example, in fluid dynamics there are numerous success stories of linear model-based flow control [39, 124, 240], for example to delay transition from laminar to turbulent flow in a spatially developing boundary layer, to reduce skin-friction drag in wall turbulence, and to stabilize the flow past an open cavity. However, many linear control approaches do not scale well to large state spaces, and they may be prohibitively expensive to enact for real-time control on short timescales. Thus, it is often necessary to develop low-dimensional approximations of the system for use in real-time feedback control.

There are two broad approaches to obtain reduced-order models (ROMs). First, it is possible to start with a high-dimensional system, such as the dis-

cretized Navier–Stokes equations, and project the dynamics onto a low-dimensional subspace identified, for example, using proper orthogonal decomposition (POD; Chapter 12) [79, 335] and Galerkin projection [75, 577]. There are numerous variations to this procedure, including the discrete empirical interpolation method (DEIM; Section 13.5) [171, 556], gappy POD (Section 13.1) [239], balanced proper orthogonal decomposition (BPOD; Section 9.2) [608, 755], and many more. The second approach is to collect data from a simulation or an experiment and identify a low-rank model using data-driven techniques. This approach is typically called system identification, and is often preferred for control design because of the relative ease of implementation. Examples include the dynamic mode decomposition (DMD; Section 7.2) [422, 611, 635, 727], the eigensystem realization algorithm (ERA; Section 9.3) [358, 468], the observer Kalman filter identification (OKID; Section 9.3) [357, 359, 563], NARMAX [86], and the sparse identification of nonlinear dynamics (SINDy; Section 7.3) [132].

After a linear model has been identified, either by model reduction or system identification, it may then be used for model-based control design. However, there are a number of issues that may arise in practice, as linear model-based control might not work for a large class of systems. First, the system being modeled may be strongly nonlinear, in which case the linear approximation might only capture a small portion of the dynamic effects. Next, the system may be stochastically driven, so that the linear model will average out the relevant fluctuations. Finally, when control is applied to the full system, the attractor dynamics may change, rendering the linearized model invalid. Exceptions include the stabilization of fixed points, where feedback control rejects nonlinear disturbances and keeps the system in a neighborhood of the fixed point where the linearized model is accurate. There are also methods for system identification and model reduction that are nonlinear, involve stochasticity, and change with the attractor. However, these methods are typically advanced and they also may limit the available machinery from control theory.

9.2 Balanced Model Reduction

The high dimensionality and short timescales associated with complex systems may render the model-based control strategies described in Chapter 8 infeasible for real-time applications. Moreover, obtaining $\mathcal{H}_2$ and $\mathcal{H}_\infty$ optimal controllers may be computationally intractable, as they involve either solving a high-dimensional Riccati equation, or an expensive iterative optimization. As has been demonstrated throughout this book, even if the ambient dimension is large, there may still be a few dominant coherent structures that characterize the system. Reduced-order models provide efficient, low-dimensional representations of these most relevant mechanisms. Low-order models may then

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

be used to design efficient controllers that can be applied in real time, even for high-dimensional systems. An alternative is to develop controllers based on the full-dimensional model and then apply model reduction techniques directly to the full controller [172, 261, 283, 537].

Model reduction is essentially data reduction that respects the fact that the data is generated by a dynamic process. If the dynamical system is a linear time-invariant (LTI) input–output system, then there is a wealth of machinery available for model reduction, and performance bounds may be quantified. The techniques explored here are based on the singular value decomposition (SVD; Chapter 1) [144, 285, 286], and the minimal realization theory of Ho and Kalman [329, 509]. The general idea is to determine a hierarchical modal decomposition of the system state that may be truncated at some model order, only keeping the coherent structures that are most important for control.

The Goal of Model Reduction

Consider a high-dimensional system, depicted schematically in Fig. 9.1,

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{x} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}, \tag{9.1a}$$

$$\mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u}, \tag{9.1b}$$

for example, from a spatially discretized simulation of a partial differential equation (PDE). The primary goal of model reduction is to find a coordinate transformation $\mathbf{x} = \mathbf{\Psi}\tilde{\mathbf{x}}$ giving rise to a related system $(\tilde{\mathbf{A}}, \tilde{\mathbf{B}}, \tilde{\mathbf{C}}, \tilde{\mathbf{D}})$ with similar input–output characteristics,

$$\frac{\mathrm{d}}{\mathrm{d}t}\tilde{\mathbf{x}} = \tilde{\mathbf{A}}\tilde{\mathbf{x}} + \tilde{\mathbf{B}}\mathbf{u}, \tag{9.2a}$$

$$\mathbf{y} = \tilde{\mathbf{C}}\tilde{\mathbf{x}} + \tilde{\mathbf{D}}\mathbf{u}, \tag{9.2b}$$

in terms of a state $\tilde{\mathbf{x}} \in \mathbb{R}^r$ with reduced dimension, $r \ll n$. Note that $\mathbf{u}$ and $\mathbf{y}$ are the same in (9.1) and (9.2) even though the system states are different. Obtaining the projection operator $\mathbf{\Psi}$ will be the focus of this section.

As a motivating example, consider the following simplified model:

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} -2 & 0 \\ 0 & -1 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 1 \\ 10^{-10} \end{bmatrix} u, \tag{9.3a}$$

$$y = \begin{bmatrix} 1 & 10^{-10} \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}. \tag{9.3b}$$

In this case, the state x_2 is barely controllable and barely observable. Simply choosing $\tilde{x} = x_1$ will result in a reduced-order model that faithfully captures

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

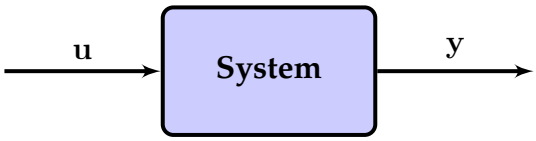


Figure 9.1: Input–output system. A control-oriented reduced-order model will capture the transfer function from $\mathbf{u}$ to $\mathbf{y}$.

the input–output dynamics. Although the choice $\tilde{x} = x_1$ seems intuitive in this extreme case, many model reduction techniques would erroneously favor the state $\tilde{x} = x_2$, since it is more lightly damped. Throughout this section, we will investigate how to accurately and efficiently find the transformation matrix $\mathbf{\Psi}$ that best captures the input–output dynamics.

The proper orthogonal decomposition [79, 335] from Chapter 12 provides a transform matrix $\mathbf{\Psi}$, the columns of which are modes that are ordered based on energy content.[1] POD has been widely used to generate ROMs of complex systems, many for control, and it is guaranteed to provide an optimal low-rank basis to capture the maximal energy or variance in a data set. However, it may be the case that the most energetic modes are nearly uncontrollable or unobservable, and therefore may not be relevant for control. Similarly, in many cases the most controllable and observable state directions may have very low energy; for example, acoustic modes typically have very low energy, yet they mediate the dominant input–output dynamics in many fluid systems. The rudder on a ship provides a good analogy: although it accounts for a small amount of the total energy, it is dynamically important for control.

Instead of ordering modes based on energy, it is possible to determine a hierarchy of modes that are most controllable and observable, therefore capturing the most input–output information. These modes give rise to *balanced* models, giving equal weighting to the controllability and observability of a state via a coordinate transformation that makes the controllability and observability Gramians equal and diagonal. These models have been extremely successful, although computing a balanced model using traditional methods is prohibitively expensive for high-dimensional systems. In this section, we describe the balancing procedure, as well as modern methods for efficient computation of balanced models. A computationally efficient suite of algorithms for model reduction and system identification may be found in [71].

A balanced reduced-order model should map inputs to outputs as faithfully as possible for a given model order r. It is therefore important to introduce an

[1] When the training data consists of velocity fields, for example from a high-dimensional discretized fluid system, then the singular values literally indicate the kinetic energy content of the associated mode. It is common to refer to POD modes as being ordered by *energy* content, even in other applications, although *variance* is more technically correct.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

operator norm to quantify how similarly (9.1) and (9.2) act on a given set of inputs. Typically, we take the infinity-norm of the difference between the transfer functions $\mathbf{G}(s)$ and $\mathbf{G}_r(s)$ obtained from the full system (9.1) and reduced system (9.2), respectively. This norm is given by

$$\|\mathbf{G}\|_\infty \triangleq \max_\omega \sigma_1(\mathbf{G}(i\omega)). \tag{9.4}$$

See Section 8.8 for a primer on transfer functions. To summarize, we seek a reduced-order model (9.2) of low order, $r \ll n$, so the operator norm $\|\mathbf{G}-\mathbf{G}_r\|_\infty$ is small.

Change of Variables in Control Systems

The balanced model reduction problem may be formulated in terms of first finding a coordinate transformation,

$$\mathbf{x} = \mathbf{T}\mathbf{z}, \tag{9.5}$$

that hierarchically orders the states in $\mathbf{z}$ in terms of their ability to capture the input–output characteristics of the system. We will begin by considering an invertible transformation $\mathbf{T} \in \mathbb{R}^{n\times n}$, and then provide a method to compute just the first r columns, which will comprise the transformation $\mathbf{\Psi}$ in (9.2). Thus, it will be possible to retain only the first r most controllable/observable states, while truncating the rest. This is similar to the change of variables into eigenvector coordinates in (8.18), except that we emphasize controllability and observability rather than characteristics of the dynamics.

Substituting $\mathbf{T}\mathbf{z}$ into (9.1) gives

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{T}\mathbf{z} = \mathbf{A}\mathbf{T}\mathbf{z} + \mathbf{B}\mathbf{u}, \tag{9.6a}$$

$$\mathbf{y} = \mathbf{C}\mathbf{T}\mathbf{z} + \mathbf{D}\mathbf{u}. \tag{9.6b}$$

Finally, multiplying (9.6a) by $\mathbf{T}^{-1}$ yields

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{z} = \mathbf{T}^{-1}\mathbf{A}\mathbf{T}\mathbf{z} + \mathbf{T}^{-1}\mathbf{B}\mathbf{u}, \tag{9.7a}$$

$$\mathbf{y} = \mathbf{C}\mathbf{T}\mathbf{z} + \mathbf{D}\mathbf{u}. \tag{9.7b}$$

This results in the following transformed equations:

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{z} = \hat{\mathbf{A}}\mathbf{z} + \hat{\mathbf{B}}\mathbf{u}, \tag{9.8a}$$

$$\mathbf{y} = \hat{\mathbf{C}}\mathbf{z} + \mathbf{D}\mathbf{u}, \tag{9.8b}$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

where $\hat{\mathbf{A}} = \mathbf{T}^{-1}\mathbf{A}\mathbf{T}$, $\hat{\mathbf{B}} = \mathbf{T}^{-1}\mathbf{B}$, and $\hat{\mathbf{C}} = \mathbf{C}\mathbf{T}$. Note that when the columns of $\mathbf{T}$ are orthonormal, the change of coordinates becomes

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{z} = \mathbf{T}^*\mathbf{A}\mathbf{T}\mathbf{z} + \mathbf{T}^*\mathbf{B}\mathbf{u}, \tag{9.9a}$$

$$\mathbf{y} = \mathbf{C}\mathbf{T}\mathbf{z} + \mathbf{D}\mathbf{u}. \tag{9.9b}$$

Gramians and Coordinate Transformations

The controllability and observability Gramians each establish an inner product on state space in terms of how controllable or observable a given state is, respectively. As such, Gramians depend on the particular choice of coordinate system and will transform under a change of coordinates. In the coordinate system $\mathbf{z}$ given by (9.5), the controllability Gramian becomes

$$\hat{\mathbf{W}}_c = \int_0^\infty e^{\hat{\mathbf{A}}\tau}\hat{\mathbf{B}}\hat{\mathbf{B}}^* e^{\hat{\mathbf{A}}^*\tau}\,\mathrm{d}\tau \tag{9.10a}$$

$$= \int_0^\infty e^{\mathbf{T}^{-1}\mathbf{A}\mathbf{T}\tau}\mathbf{T}^{-1}\mathbf{B}\mathbf{B}^*\mathbf{T}^{-*}e^{\mathbf{T}^*\mathbf{A}^*\mathbf{T}^{-*}\tau}\,\mathrm{d}\tau \tag{9.10b}$$

$$= \int_0^\infty \mathbf{T}^{-1}e^{\mathbf{A}\tau}\mathbf{T}\mathbf{T}^{-1}\mathbf{B}\mathbf{B}^*\mathbf{T}^{-*}\mathbf{T}^* e^{\mathbf{A}^*\tau}\mathbf{T}^{-*}\,\mathrm{d}\tau \tag{9.10c}$$

$$= \mathbf{T}^{-1}\left(\int_0^\infty e^{\mathbf{A}\tau}\mathbf{B}\mathbf{B}^* e^{\mathbf{A}^\tau}\,\mathrm{d}\tau\right)\mathbf{T}^{-*} \tag{9.10d}$$

$$= \mathbf{T}^{-1}\mathbf{W}_c\mathbf{T}^{-*}. \tag{9.10e}$$

Note that here we introduce $\mathbf{T}^{-*} := (\mathbf{T}^{-1})^* = (\mathbf{T}^*)^{-1}$. The observability Gramian transforms similarly:

$$\hat{\mathbf{W}}_o = \mathbf{T}^*\mathbf{W}_o\mathbf{T}, \tag{9.11}$$

which is an exercise for the reader. Both Gramians transform as tensors (i.e., in terms of the transform matrix $\mathbf{T}$ and its transpose, rather than $\mathbf{T}$ and its inverse), which is consistent with them inducing an inner product on state space.

Simple Rescaling

This example, modified from Moore [509], demonstrates the ability to balance a system through a change of coordinates. Consider the system

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & -10 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 10^{-3} \\ 10^3 \end{bmatrix} u, \tag{9.12a}$$

$$y = \begin{bmatrix} 10^3 & 10^{-3} \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}. \tag{9.12b}$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

In this example, the first state x_1 is barely controllable, while the second state is barely observable. However, under the change of coordinates $z_1 = 10^3 x_1$ and $z_2 = 10^{-3} x_2$, the system becomes balanced:

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & -10 \end{bmatrix}\begin{bmatrix} z_1 \\ z_2 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} u, \tag{9.13a}$$

$$y = \begin{bmatrix} 1 & 1 \end{bmatrix}\begin{bmatrix} z_1 \\ z_2 \end{bmatrix}. \tag{9.13b}$$

In this example, the coordinate change simply rescales the state $\mathbf{x}$. For instance, it may be that the first state had units of millimeters while the second state had units of kilometers. Writing both states in meters balances the dynamics; i.e., the controllability and observability Gramians are equal and diagonal.

Balancing Transformations

Now we are ready to derive the balancing coordinate transformation $\mathbf{T}$ that makes the controllability and observability Gramians equal and diagonal:

$$\hat{\mathbf{W}}_c = \hat{\mathbf{W}}_o = \mathbf{\Sigma}. \tag{9.14}$$

First, consider the product of the Gramians from (9.10) and (9.11):

$$\hat{\mathbf{W}}_c \hat{\mathbf{W}}_o = \mathbf{T}^{-1}\mathbf{W}_c\mathbf{W}_o\mathbf{T}, \tag{9.15}$$

Plugging in the desired $\hat{\mathbf{W}}_c = \hat{\mathbf{W}}_o = \mathbf{\Sigma}$ yields

$$\mathbf{T}^{-1}\mathbf{W}_c\mathbf{W}_o\mathbf{T} = \mathbf{\Sigma}^2 \quad \Longrightarrow \quad \mathbf{W}_c\mathbf{W}_o\mathbf{T} = \mathbf{T}\mathbf{\Sigma}^2. \tag{9.16}$$

The latter expression in (9.16) is the equation for the eigendecomposition of $\mathbf{W}_c\mathbf{W}_o$, the product of the Gramians in the original coordinates. Thus, the balancing transformation $\mathbf{T}$ is related to the eigendecomposition of $\mathbf{W}_c\mathbf{W}_o$. The expression above is valid for any scaling of the eigenvectors, and the correct rescaling must be chosen to exactly balance the Gramians. In other words, there are many such transformations $\mathbf{T}$ that make the product $\hat{\mathbf{W}}_c\hat{\mathbf{W}}_o = \mathbf{\Sigma}^2$, but where the individual Gramians are not equal (for example, diagonal Gramians $\hat{\mathbf{W}}_c = \mathbf{\Sigma}_c$ and $\hat{\mathbf{W}}_o = \mathbf{\Sigma}_o$ will satisfy (9.16) if $\mathbf{\Sigma}_c\mathbf{\Sigma}_o = \mathbf{\Sigma}^2$).

Below, we will introduce the matrix $\mathbf{S} = \mathbf{T}^{-1}$ to simplify notation.

Scaling Eigenvectors for the Balancing Transformation

To find the correct scaling of eigenvectors to make $\hat{\mathbf{W}}_c = \hat{\mathbf{W}}_o = \mathbf{\Sigma}$, first consider the simplified case of balancing the first diagonal element of $\mathbf{\Sigma}$. Let $\boldsymbol{\xi}_u$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

denote the unscaled first column of $\mathbf{T}$, and let $\boldsymbol{\eta}_u$ denote the unscaled first row of $\mathbf{S} = \mathbf{T}^{-1}$. Then

$$\boldsymbol{\eta}_u \mathbf{W}_c \boldsymbol{\eta}_u^* = \sigma_c, \tag{9.17a}$$
$$\boldsymbol{\xi}_u^* \mathbf{W}_o \boldsymbol{\xi}_u = \sigma_o. \tag{9.17b}$$

The first element of the diagonalized controllability Gramian is thus σ_c, while the first element of the diagonalized observability Gramian is σ_o. If we scale the eigenvector $\boldsymbol{\xi}_u$ by σ_s, then the inverse eigenvector $\boldsymbol{\eta}_u$ is scaled by σ_s^{-1}. Transforming via the new scaled eigenvectors $\boldsymbol{\xi}_s = \sigma_s \boldsymbol{\xi}_u$ and $\boldsymbol{\eta}_s = \sigma_s^{-1} \boldsymbol{\eta}_u$ yields

$$\boldsymbol{\eta}_s \mathbf{W}_c \boldsymbol{\eta}_s^* = \sigma_s^{-2} \sigma_c, \tag{9.18a}$$
$$\boldsymbol{\xi}_s^* \mathbf{W}_o \boldsymbol{\xi}_s = \sigma_s^2 \sigma_o. \tag{9.18b}$$

Thus, for the two Gramians to be equal,

$$\sigma_s^{-2} \sigma_c = \sigma_s^2 \sigma_o \quad \Longrightarrow \quad \sigma_s = \left(\frac{\sigma_c}{\sigma_o} \right)^{1/4}. \tag{9.19}$$

To balance every diagonal entry of the controllability and observability Gramians, we first consider the unscaled eigenvector transformation $\mathbf{T}_u$ from (9.16); the subscript u simply denotes *unscaled*. As an example, we use the standard scaling in most computational software so that the columns of $\mathbf{T}_u$ have unit norm. Then both Gramians are diagonalized, but are not necessarily equal:

$$\mathbf{T}_u^{-1} \mathbf{W}_c \mathbf{T}_u^{-*} = \boldsymbol{\Sigma}_c, \tag{9.20a}$$
$$\mathbf{T}_u^* \mathbf{W}_o \mathbf{T}_u = \boldsymbol{\Sigma}_o. \tag{9.20b}$$

The scaling that exactly balances these Gramians is then given by $\boldsymbol{\Sigma}_s = \boldsymbol{\Sigma}_c^{1/4} \boldsymbol{\Sigma}_o^{-1/4}$. Thus, the exact balancing transformation is given by

$$\mathbf{T} = \mathbf{T}_u \boldsymbol{\Sigma}_s. \tag{9.21}$$

It is possible to directly confirm that this transformation balances the Gramians:

$$(\mathbf{T}_u \boldsymbol{\Sigma}_s)^{-1} \mathbf{W}_c (\mathbf{T}_u \boldsymbol{\Sigma}_s)^{-*} = \boldsymbol{\Sigma}_s^{-1} \mathbf{T}_u^{-1} \mathbf{W}_c \mathbf{T}_u^{-*} \boldsymbol{\Sigma}_s^{-1} = \boldsymbol{\Sigma}_s^{-1} \boldsymbol{\Sigma}_c \boldsymbol{\Sigma}_s^{-1} = \boldsymbol{\Sigma}_c^{1/2} \boldsymbol{\Sigma}_o^{1/2}, \tag{9.22a}$$
$$(\mathbf{T}_u \boldsymbol{\Sigma}_s)^* \mathbf{W}_o (\mathbf{T}_u \boldsymbol{\Sigma}_s) = \boldsymbol{\Sigma}_s \mathbf{T}_u^* \mathbf{W}_o \mathbf{T}_u \boldsymbol{\Sigma}_s = \boldsymbol{\Sigma}_s \boldsymbol{\Sigma}_o \boldsymbol{\Sigma}_s = \boldsymbol{\Sigma}_c^{1/2} \boldsymbol{\Sigma}_o^{1/2}. \tag{9.22b}$$

The manipulations above rely on the fact that diagonal matrices commute, so that $\boldsymbol{\Sigma}_c \boldsymbol{\Sigma}_o = \boldsymbol{\Sigma}_o \boldsymbol{\Sigma}_c$, etc.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Example of the Balancing Transform and Gramians

Before confronting the practical challenges associated with accurately and efficiently computing the balancing transformation, it is helpful to consider an illustrative example.

In MATLAB, computing the balanced system and the balancing transformation is a simple one-line command:

```
[sysb,g,Ti,T] = balreal(sys); % Balance the system
```

In this code, **T** is the transformation, **Ti** is the inverse transformation, **sysb** is the balanced system, and **g** is a vector containing the diagonal elements of the balanced Gramians.

In Python, computing the balanced system is also simple using the Python Control Systems Library (`python-control`):[2]

```
sysb = balreal(sys,len(B)); # Balance the system
```

The following example illustrates the balanced realization for a two-dimensional system. First, we generate a system and compute its balanced realization, along with the Gramians for each system. Next, we visualize the Gramians of the unbalanced and balanced systems in Fig. 9.2.

Code 9.1: [MATLAB] Obtaining a balanced realization.

```
A = [-.75 1; -.3 -.75];
B = [2; 1];
C = [1 2];
D = 0;

sys = ss(A,B,C,D);

Wc = gram(sys,'c'); % Controllability Gramian
Wo = gram(sys,'o'); % Observability Gramian

[sysb,g,Ti,T] = balreal(sys); % Balance the system

BWc = gram(sysb,'c') % Balanced Gramians
BWo = gram(sysb,'o')
```

Code 9.1: [Python] Obtaining a balanced realization.

```
from control.matlab import *    # Code will resemble Matlab
import slycot

A = np.array([[-0.75,1],[-0.3,-0.75]])
```

[2]The Python control toolbox is available at `https://python-control.readthedocs.io/`.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

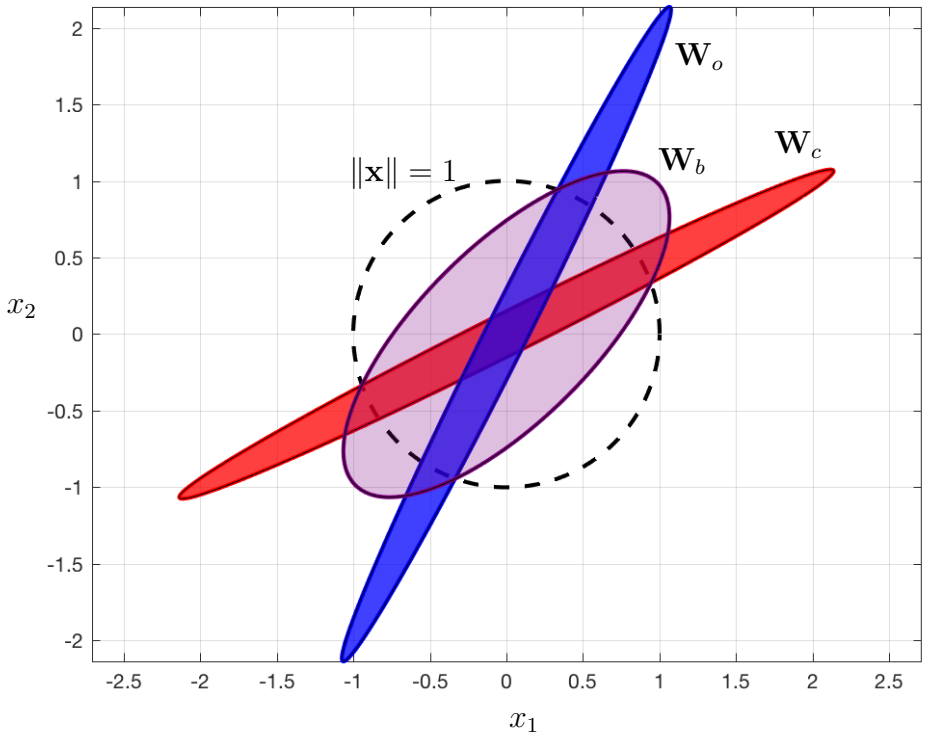


Figure 9.2: Illustration of balancing transformation on Gramians. The reachable set with unit control input is shown in red, given by $\mathbf{W}_c^{1/2}\mathbf{x}$ for $\|\mathbf{x}\| = 1$. The corresponding observable set is shown in blue. Under the balancing transformation $\mathbf{T}$, the Gramians are equal, shown in purple.

```
B = np.array([2,1]).reshape((2,1))
C = np.array([1,2])
D = 0

sys = ss(A,B,C,D)

Wc = gram(sys,'c') # Controllability Gramian
Wo = gram(sys,'o') # Observability Gramian

sysb = balred(sys,len(B)) # Balance the system

BWc = gram(sysb,'c') # Balanced Gramians
BWo = gram(sysb,'o')
```

The resulting balanced Gramians are equal, diagonal, and ordered from most controllable/observable mode to least:

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

```
>>BWc =
    1.9439   -0.0000
   -0.0000    0.3207

>>BWo =
    1.9439    0.0000
    0.0000    0.3207
```

To visualize the Gramians in Fig. 9.2, we first recall that the distance the system can go in a direction $\mathbf{x}$ with a unit actuation input is given by $\mathbf{x}^*\mathbf{W}_c\mathbf{x}$. Thus, the controllability Gramian may be visualized by plotting $\mathbf{W}_c^{1/2}\mathbf{x}$ for $\mathbf{x}$ on a sphere with $\|\mathbf{x}\| = 1$. The observability Gramian may be similarly visualized.

In this example, we see that the most controllable and observable directions may not be well aligned. However, by a change of coordinates, it is possible to find a new direction that is the most jointly controllable and observable. It is then possible to represent the system in this one-dimensional subspace, while still capturing a significant portion of the input–output energy. If the red and blue Gramians were exactly perpendicular, so that the most controllable direction was the least observable direction, and vice versa, then the balanced Gramian would be a circle. In this case, there is no preferred state direction, and both directions are equally important for the input–output behavior.

Instead of using the **balreal** command, it is possible to manually construct the balancing transformation from the eigendecomposition of $\mathbf{W}_c\mathbf{W}_o$, as described above and provided in code available online.

Balanced Truncation

We have now shown that it is possible to define a change of coordinates so that the controllability and observability Gramians are equal and diagonal. Moreover, these new coordinates may be ranked hierarchically in terms of their joint controllability and observability. It may be possible to truncate these coordinates and keep only the most controllable/observable directions, resulting in a reduced-order model that faithfully captures input–output dynamics.

Given the new coordinates $\mathbf{z} = \mathbf{T}^{-1}\mathbf{x} \in \mathbb{R}^n$, it is possible to define a reduced-order state $\tilde{\mathbf{x}} \in \mathbb{R}^r$ as

$$\mathbf{z} = \begin{bmatrix} z_1 \\ \vdots \\ z_r \\ z_{r+1} \\ \vdots \\ z_n \end{bmatrix} \left.\begin{matrix} \\ \\ \\ \\ \\ \\ \end{matrix}\right\} \tilde{\mathbf{x}} \tag{9.23}$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

in terms of the first r most controllable and observable directions. If we partition the balancing transformation $\mathbf{T}$ and inverse transformation $\mathbf{S} = \mathbf{T}^{-1}$ into the first r modes to be retained and the last $n - r$ modes to be truncated,

$$\mathbf{T} = \begin{bmatrix} \boldsymbol{\Psi} & \mathbf{T}_t \end{bmatrix}, \quad \mathbf{S} = \begin{bmatrix} \boldsymbol{\Phi}^* \\ \mathbf{S}_t \end{bmatrix}, \tag{9.24}$$

then it is possible to rewrite the transformed dynamics in (9.7) as

$$\frac{\mathrm{d}}{\mathrm{d}t}\left[\begin{array}{c} \tilde{\mathbf{x}} \\ \hline \mathbf{z}_t \end{array}\right] = \left[\begin{array}{c|c} \boldsymbol{\Phi}^*\mathbf{A}\boldsymbol{\Psi} & \boldsymbol{\Phi}^*\mathbf{A}\mathbf{T}_t \\ \hline \mathbf{S}_t\mathbf{A}\boldsymbol{\Psi} & \mathbf{S}_t\mathbf{A}\mathbf{T}_t \end{array}\right] \left[\begin{array}{c} \tilde{\mathbf{x}} \\ \hline \mathbf{z}_t \end{array}\right] + \left[\begin{array}{c} \boldsymbol{\Phi}^*\mathbf{B} \\ \hline \mathbf{S}_t\mathbf{B} \end{array}\right] \mathbf{u}, \tag{9.25a}$$

$$\mathbf{y} = \left[\begin{array}{c|c} \mathbf{C}\boldsymbol{\Psi} & \mathbf{C}\mathbf{T}_t \end{array}\right] \left[\begin{array}{c} \tilde{\mathbf{x}} \\ \hline \mathbf{z}_t \end{array}\right] + \mathbf{D}\mathbf{u}. \tag{9.25b}$$

In balanced truncation, the state $\mathbf{z}_t$ is simply truncated (i.e., discarded and set equal to zero), and only the $\tilde{\mathbf{x}}$ equations remain:

$$\frac{\mathrm{d}}{\mathrm{d}t}\tilde{\mathbf{x}} = \boldsymbol{\Phi}^*\mathbf{A}\boldsymbol{\Psi}\tilde{\mathbf{x}} + \boldsymbol{\Phi}^*\mathbf{B}\mathbf{u}, \tag{9.26a}$$

$$\mathbf{y} = \mathbf{C}\boldsymbol{\Psi}\tilde{\mathbf{x}} + \mathbf{D}\mathbf{u}. \tag{9.26b}$$

Only the first r columns of $\mathbf{T}$ and of $\mathbf{S}^* = \mathbf{T}^{-*}$ are required to construct $\boldsymbol{\Psi}$ and $\boldsymbol{\Phi}$, and thus computing the entire balancing transformation $\mathbf{T}$ is unnecessary. Note that the matrix $\boldsymbol{\Phi}$ here is different than the matrix of DMD modes in Section 7.2. The computation of $\boldsymbol{\Psi}$ and $\boldsymbol{\Phi}$ without $\mathbf{T}$ will be discussed in the following sections. A key benefit of balanced truncation is the existence of upper and lower bounds on the error of a given order truncation:

$$\text{upper bound} \quad \|\mathbf{G} - \mathbf{G}_r\|_\infty \leq 2\sum_{j=r+1}^{n} \sigma_j, \tag{9.27a}$$

$$\text{lower bound} \quad \|\mathbf{G} - \mathbf{G}_r\|_\infty > \sigma_{r+1}, \tag{9.27b}$$

where σ_j is the jth diagonal entry of the balanced Gramians. The diagonal entries of $\boldsymbol{\Sigma}$ are also known as *Hankel singular values*.

Computing Balanced Realizations

In the previous section we demonstrated the feasibility of obtaining a coordinate transformation that balances the controllability and observability Gramians. However, the computation of this balancing transformation is non-trivial, and significant work has gone into obtaining accurate and efficient methods, starting with Moore in 1981 [509], and continuing with Lall, Marsden, and Glavaški in 2002 [426], Willcox and Peraire in 2002 [755], and Rowley in 2005

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

[608]. For an excellent and complete treatment of balanced realizations and model reduction, see Antoulas [24].

In practice, computing the Gramians $\mathbf{W}_c$ and $\mathbf{W}_o$ and the eigendecomposition of the product $\mathbf{W}_c\mathbf{W}_o$ in (9.16) may be prohibitively expensive for high-dimensional systems. Instead, the balancing transformation may be approximated from impulse-response data, utilizing the singular value decomposition for efficient extraction of the most relevant subspaces.

We will first show that Gramians may be approximated via a snapshot matrix from impulse-response experiments/simulations. Then, we will show how the balancing transformation may be obtained from this data.

Empirical Gramians

In practice, computing Gramians via the Lyapunov equation is computationally expensive, with computational complexity of $\mathcal{O}(n^3)$. Instead, the Gramians may be approximated by full-state measurements of the discrete-time direct and adjoint systems:

$$\text{direct} \quad \mathbf{x}_{k+1} = \mathbf{A}_d\mathbf{x}_k + \mathbf{B}_d\mathbf{u}_k, \tag{9.28a}$$

$$\text{adjoint} \quad \mathbf{x}_{k+1} = \mathbf{A}_d^*\mathbf{x}_k + \mathbf{C}_d^*\mathbf{y}_k. \tag{9.28b}$$

Equation (9.28a) is the discrete-time dynamic update equation from (8.21), and (9.28b) is the adjoint equation. The matrices $\mathbf{A}_d$, $\mathbf{B}_d$, and $\mathbf{C}_d$ are the discrete-time system matrices from (8.22). Note that the adjoint equation is generally non-physical, and must be simulated; thus the methods here apply to analytical equations and simulations, but not to experimental data. An alternative formulation that does not rely on adjoint data, and therefore generalizes to experiments, will be provided in Section 9.3.

Computing the impulse response of the direct and adjoint systems yields the following discrete-time snapshot matrices:

$$\mathcal{C}_d = \begin{bmatrix}\mathbf{B}_d & \mathbf{A}_d\mathbf{B}_d & \cdots & \mathbf{A}_d^{m_c-1}\mathbf{B}_d\end{bmatrix}, \quad \mathcal{O}_d = \begin{bmatrix}\mathbf{C}_d \\ \mathbf{C}_d\mathbf{A}_d \\ \vdots \\ \mathbf{C}_d\mathbf{A}_d^{m_o-1}\end{bmatrix}. \tag{9.29}$$

Note that when $m_c = n$, $\mathcal{C}_d$ is the discrete-time controllability matrix, and when $m_o = n$, $\mathcal{O}_d$ is the discrete-time observability matrix; however, we generally consider $m_c, m_o \ll n$. These matrices may also be obtained by sampling the continuous-time direct and adjoint systems at a regular interval Δt.

It is now possible to compute *empirical* Gramians that approximate the true Gramians without solving the Lyapunov equations in (8.42) and (8.43):

$$\mathbf{W}_c \approx \mathbf{W}_c^e = \mathcal{C}_d\mathcal{C}_d^*, \tag{9.30a}$$

$$\mathbf{W}_o \approx \mathbf{W}_o^e = \mathcal{O}_d^*\mathcal{O}_d. \tag{9.30b}$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

The empirical Gramians essentially comprise a Riemann sum approximation of the integral in the continuous-time Gramians, which becomes exact as the time-step of the discrete-time system becomes arbitrarily small and the duration of the impulse response becomes arbitrarily large. In practice, the impulse-response snapshots should be collected until the lightly damped transients die out. The method of empirical Gramians is quite efficient, and is widely used [425, 426, 509, 608, 755]. Note that p adjoint impulse responses are required, where p is the number of outputs. This becomes intractable when there are a large number of outputs (e.g., full-state measurements), motivating the output projection below.

Balanced POD

Instead of computing the eigendecomposition of $\mathbf{W}_c\mathbf{W}_o$, which is an $n \times n$ matrix, it is possible to compute the balancing transformation via the singular value decomposition of the product of the snapshot matrices,

$$\boldsymbol{\mathcal{O}_d}\boldsymbol{\mathcal{C}_d}, \tag{9.31}$$

reminiscent of the method of snapshots from Section 1.3 [663]. This is the approach taken by Rowley [608].

First, define the generalized Hankel matrix as the product of the adjoint ($\boldsymbol{\mathcal{O}}_d$) and direct ($\boldsymbol{\mathcal{C}}_d$) snapshot matrices from (9.29), for the discrete-time system:

$$\mathbf{H} = \boldsymbol{\mathcal{O}}_d\boldsymbol{\mathcal{C}}_d = \begin{bmatrix} \mathbf{C}_d \\ \mathbf{C}_d\mathbf{A}_d \\ \vdots \\ \mathbf{C}_d\mathbf{A}_d^{m_o-1} \end{bmatrix} \begin{bmatrix} \mathbf{B}_d & \mathbf{A}_d\mathbf{B}_d & \cdots & \mathbf{A}_d^{m_c-1}\mathbf{B}_d \end{bmatrix} \tag{9.32a}$$

$$= \begin{bmatrix} \mathbf{C}_d\mathbf{B}_d & \mathbf{C}_d\mathbf{A}_d\mathbf{B}_d & \cdots & \mathbf{C}_d\mathbf{A}_d^{m_c-1}\mathbf{B}_d \\ \mathbf{C}_d\mathbf{A}_d\mathbf{B}_d & \mathbf{C}_d\mathbf{A}_d^2\mathbf{B}_d & \cdots & \mathbf{C}_d\mathbf{A}_d^{m_c}\mathbf{B}_d \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{C}_d\mathbf{A}_d^{m_o-1}\mathbf{B}_d & \mathbf{C}_d\mathbf{A}_d^{m_o}\mathbf{B}_d & \cdots & \mathbf{C}_d\mathbf{A}_d^{m_c+m_o-2}\mathbf{B}_d \end{bmatrix}. \tag{9.32b}$$

Next, we factor $\mathbf{H}$ using the SVD:

$$\mathbf{H} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^* = \begin{bmatrix} \tilde{\mathbf{U}} & \mathbf{U}_t \end{bmatrix} \begin{bmatrix} \tilde{\boldsymbol{\Sigma}} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\Sigma}_t \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{V}}^* \\ \mathbf{V}_t^* \end{bmatrix} \approx \tilde{\mathbf{U}}\tilde{\boldsymbol{\Sigma}}\tilde{\mathbf{V}}^*. \tag{9.33}$$

For a given desired model order $r \ll n$, only the first r columns of $\mathbf{U}$ and $\mathbf{V}$ are retained, along with the first $r \times r$ block of $\boldsymbol{\Sigma}$; the remaining contribution from $\mathbf{U}_t\boldsymbol{\Sigma}_t\mathbf{V}_t^*$ may be truncated. This yields a bi-orthogonal set of modes given by:

$$\text{direct modes} \quad \boldsymbol{\Psi} = \boldsymbol{\mathcal{C}}_d\tilde{\mathbf{V}}\tilde{\boldsymbol{\Sigma}}^{-1/2}, \tag{9.34a}$$

$$\text{adjoint modes} \quad \boldsymbol{\Phi} = \boldsymbol{\mathcal{O}}_d^*\tilde{\mathbf{U}}\tilde{\boldsymbol{\Sigma}}^{-1/2}. \tag{9.34b}$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

The direct modes $\boldsymbol{\Psi} \in \mathbb{R}^{n\times r}$ and adjoint modes $\boldsymbol{\Phi} \in \mathbb{R}^{n\times r}$ are bi-orthogonal, $\boldsymbol{\Phi}^*\boldsymbol{\Psi} = \mathbf{I}_{r\times r}$, and Rowley [608] showed that they establish the change of coordinates that balance the truncated empirical Gramians. Thus, $\boldsymbol{\Psi}$ approximates the first r columns of the full $n \times n$ balancing transformation, $\mathbf{T}$, and $\boldsymbol{\Phi}^*$ approximates the first r rows of the $n \times n$ inverse balancing transformation, $\mathbf{S} = \mathbf{T}^{-1}$.

Now, it is possible to project the original system onto these modes, yielding a balanced reduced-order model of order r:

$$\tilde{\mathbf{A}} = \boldsymbol{\Phi}^*\mathbf{A}_d\boldsymbol{\Psi}, \tag{9.35a}$$

$$\tilde{\mathbf{B}} = \boldsymbol{\Phi}^*\mathbf{B}_d, \tag{9.35b}$$

$$\tilde{\mathbf{C}} = \mathbf{C}_d\boldsymbol{\Psi}. \tag{9.35c}$$

It is possible to compute the reduced system dynamics in (9.35a) without having direct access to $\mathbf{A}_d$. In some cases, $\mathbf{A}_d$ may be exceedingly large and unwieldy, and instead it is only possible to evaluate the action of this matrix on an input vector. For example, in many modern fluid dynamics codes, the matrix $\mathbf{A}_d$ is not actually represented, but because it is sparse, it is possible to implement efficient routines to multiply this matrix by a vector.

It is important to note that the reduced-order model in (9.35) is formulated in discrete time, as it is based on discrete-time empirical snapshot matrices. However, it is simple to obtain the corresponding continuous-time system:

```
>>sysD = ss(Atilde,Btilde,Ctilde,D,dt);   % Discrete-time
>>sysC = d2c(sysD);                       % Continuous-time
```

In this example, $\mathbf{D}$ is the same in continuous time and discrete time, and in the full-order and reduced-order models.

Note that a BPOD model may not exactly satisfy the upper bound from balanced truncation (see (9.27)) due to errors in the empirical Gramians.

Output Projection

Often, in high-dimensional simulations, we assume full-state measurements, so that $p = n$ is exceedingly large. To avoid computing $p = n$ adjoint simulations, it is possible instead to solve an output-projected adjoint equation [608]:

$$\mathbf{x}_{k+1} = \mathbf{A}_d^*\mathbf{x}_k + \mathbf{C}_d^*\tilde{\mathbf{U}}\mathbf{y}, \tag{9.36}$$

where $\tilde{\mathbf{U}}$ is a matrix containing the first r singular vectors of $\boldsymbol{\mathcal{C}_d}$. Thus, we first identify a low-dimensional POD subspace $\tilde{\mathbf{U}}$ from a direct impulse response, and then only perform adjoint impulse-response simulations by exciting these few *POD coefficient* measurements. More generally, if $\mathbf{y}$ is high-dimensional but does not measure the full state, it is possible to use a POD subspace trained on the measurements, given by the first r singular vectors $\tilde{\mathbf{U}}$ of $\mathbf{C}_d\boldsymbol{\mathcal{C}_d}$. Adjoint impulse responses may then be performed in these output POD directions.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Data Collection and Stacking

The powers m_c and m_o in (9.32) signify that data must be collected until the matrices $\mathcal{C}_d$ and $\mathcal{O}_d^*$ are full rank, after which the controllable/observable subspaces have been sampled. Unless we collect data until transients decay, the true Gramians are only approximately balanced. Instead, it is possible to collect data until the Hankel matrix is full rank, balance the resulting model, and then truncate. This more efficient approach is developed in [724] and [462].

The snapshot matrices in (9.29) are generated from impulse-response simulations of the direct (9.28a) and adjoint (9.36) systems. These time-series snapshots are then interleaved to form the snapshot matrices.

Historical Note

The balanced POD method described above originated with the seminal work of Moore in 1981 [509], which provided a data-driven generalization of the minimal realization theory of Ho and Kalman [329]. Until then, minimal realizations were defined in terms of idealized controllable and observable subspaces, which neglected the subtlety of degrees of controllability and observability.

Moore's paper introduced a number of critical concepts that bridged the gap from theory to reality. First, he established a connection between principal component analysis (PCA) and Gramians, showing that information about degrees of controllability and observability may be mined from data via the SVD. Next, Moore showed that a balancing transformation exists that makes the Gramians equal, diagonal, and hierarchically ordered by balanced controllability and observability; moreover, he provides an algorithm to compute this transformation. This set the stage for principled model reduction, whereby states may be truncated based on their joint controllability and observability. Moore further introduced the notion of an empirical Gramian, although he did not use this terminology. He also realized that computing $\mathbf{W}_c$ and $\mathbf{W}_o$ directly is less accurate than computing the SVD of the empirical snapshot matrices from the direct and adjoint systems, and he avoided directly computing the eigendecomposition of $\mathbf{W}_c\mathbf{W}_o$ by using these SVD transformations. In 2002, Lall, Marsden, and Glavaški [426] generalized this theory to nonlinear systems.

One drawback of Moore's approach is that he computed the entire $n \times n$ balancing transformation, which is not suitable for exceedingly high-dimensional systems. In 2002, Willcox and Peraire [755] generalized the method to high-dimensional systems, introducing a variant based on the rank-r decompositions of $\mathbf{W}_c$ and $\mathbf{W}_o$ obtained from the direct and adjoint snapshot matrices. It is then possible to compute the eigendecomposition of $\mathbf{W}_c\mathbf{W}_o$ using efficient eigenvalue solvers without ever actually writing down the full $n \times n$ matrices. However, this approach has the drawback of requiring as many adjoint impulse-response simulations as the number of output equations, which may

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

be exceedingly large for full-state measurements. In 2005, Rowley [608] addressed this issue by introducing the output projection, discussed above, which limits the number of adjoint simulations to the number of relevant POD modes in the data. He also showed that it is possible to use the eigendecomposition of the product $\mathcal{O}_d\mathcal{C}_d$. The product $\mathcal{O}_d\mathcal{C}_d$ is often smaller, and these computations may be more accurate.

It is interesting to note that a nearly equivalent formulation was developed 20 years earlier in the field of system identification. The so-called eigensystem realization algorithm (ERA) [358], introduced in 1985 by Juang and Pappa, obtains equivalent balanced models without the need for adjoint data, making it useful for system identification in experiments. This connection between ERA and BPOD was established by Ma et al. in 2011 [468].

Balanced Model Reduction Example

In this example we will demonstrate the computation of balanced truncation and balanced POD models on a random state-space system with $n = 100$ states, $q = 2$ inputs, and $p = 2$ outputs. First, we generate a system:

```
q = 2;    % Number of inputs
p = 2;    % Number of outputs
n = 100;  % State dimension
sysFull = drss(n,p,q); % Discrete random system
```

Next, we compute the Hankel singular values, which are plotted in Fig. 9.3. We see that $r = 10$ modes capture over 90% of the input–output energy.

```
hsvs = hsvd(sysFull); % Hankel singular values
```

Now we construct an exact balanced truncation model with order $r = 10$:

```
%% Exact balanced truncation
sysBT = balred(sysFull,r);   % Balanced truncation
```

The full-order system, and the balanced truncation and balanced POD models are compared in Fig. 9.4. The code used to generate a BPOD model is available in MATLAB and Python on the book's GitHub. It can be seen that the balanced model accurately captures the dominant input–output dynamics, even when only 10% of the modes are kept.

9.3 System Identification

In contrast to model reduction, where the system model $(\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D})$ was known, system identification is purely data-driven. System identification may be thought of as a form of machine learning, where an input–output map of a system is learned from training data in a representation that generalizes to data that was

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

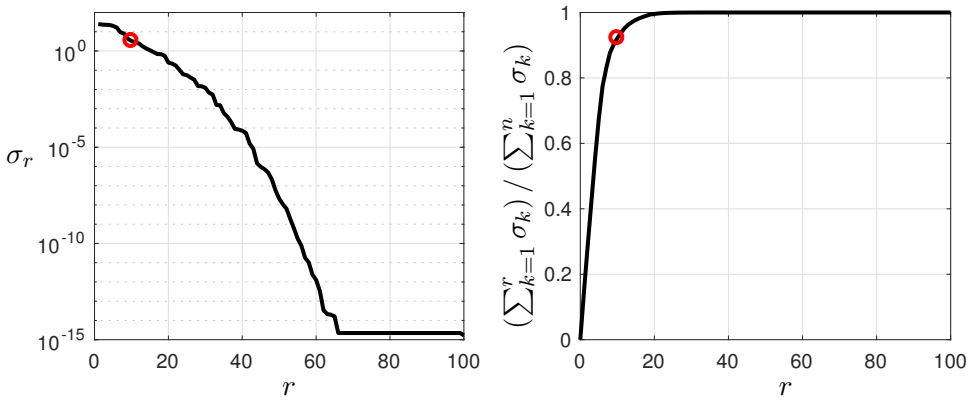


Figure 9.3: Hankel singular values (left) and cumulative sum (right) for random state-space system with $n = 100$ and $p = q = 2$. The first $r = 10$ Hankel singular values contain 92.9% of the cumulative sum.

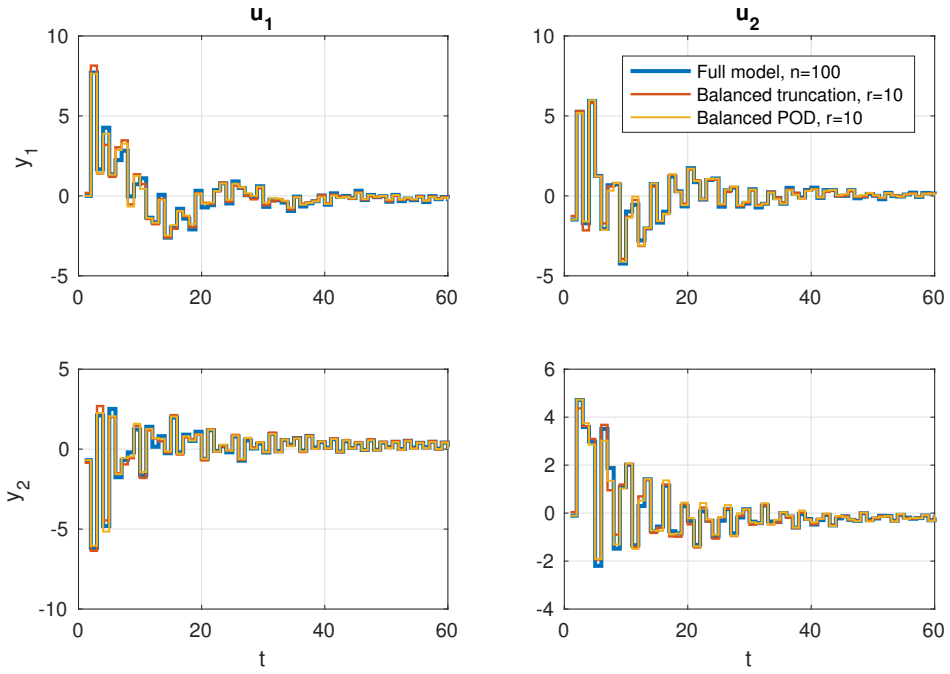


Figure 9.4: Impulse response of full-state model with $n = 100$ and $p = q = 2$, along with balanced truncation and balanced POD models with $r = 10$.

not in the training set. There is a vast literature on methods for system identification [357, 451], and many of the leading methods are based on a form of dynamic regression that fits models based on data, such as the DMD from Section 7.2. For this section, we consider the eigensystem realization algorithm (ERA) and observer Kalman filter identification (OKID) methods because of

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

their connection to balanced model reduction [468, 509, 608, 727] and their successful application in high-dimensional systems such as vibration control of aerospace structures and closed-loop flow control [38, 39, 345]. The ERA/OKID procedure is also applicable to multiple-input, multiple-output (MIMO) systems. Other methods include the autoregressive moving average (ARMA) and autoregressive moving average with exogenous inputs (ARMAX) models [100, 751], the nonlinear autoregressive moving average with exogenous inputs (NARMAX) [86] model, and the SINDy method from Section 7.3.

Eigensystem Realization Algorithm

The eigensystem realization algorithm (ERA) produces low-dimensional linear input–output models from sensor measurements of an impulse-response experiment, based on the "minimal realization" theory of Ho and Kalman [329]. The modern theory was developed to identify structural models for various spacecraft [358], and it has been shown by Ma et al. [468] that ERA models are equivalent to BPOD models.[3] However, ERA is based entirely on impulse-response measurements and does not require prior knowledge of a model.

We consider a discrete-time system, as described in Section 8.2:

$$\mathbf{x}_{k+1} = \mathbf{A}_d\mathbf{x}_k + \mathbf{B}_d\mathbf{u}_k, \tag{9.37a}$$

$$\mathbf{y}_k = \mathbf{C}_d\mathbf{x}_k + \mathbf{D}_d\mathbf{u}_k. \tag{9.37b}$$

A discrete-time delta function input in the actuation $\mathbf{u}$,

$$\mathbf{u}_k^\delta \triangleq \mathbf{u}^\delta(k\Delta t) = \begin{cases} \mathbf{I}, & k = 0, \\ \mathbf{0}, & k = 1, 2, 3, \ldots, \end{cases} \tag{9.38}$$

gives rise to a discrete-time impulse response in the sensors $\mathbf{y}$:

$$\mathbf{y}_k^\delta \triangleq \mathbf{y}^\delta(k\Delta t) = \begin{cases} \mathbf{D}_d, & k = 0, \\ \mathbf{C}_d\mathbf{A}_d^{k-1}\mathbf{B}_d, & k = 1, 2, 3, \ldots. \end{cases} \tag{9.39}$$

In an experiment or simulation, typically q impulse responses are performed, one for each of the q separate input channels. The output responses are collected for each impulsive input, and, at a given time-step k, the output vector in response to the jth impulsive input will form the jth column of $\mathbf{y}_k^\delta$. Thus, each of the $\mathbf{y}_k^\delta$ is a $p \times q$ matrix $\mathbf{C}\mathbf{A}^{k-1}\mathbf{B}$. Note that the system matrices $(\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D})$ do not actually need to exist, as the method below is purely data-driven.

The Hankel matrix $\mathbf{H}$ from (9.32) is formed by stacking shifted time series of impulse-response measurements into a matrix, as in the HAVOK method from

[3] BPOD and ERA models both balance the empirical Gramians and approximate balanced truncation [509] for high-dimensional systems, given a sufficient volume of data.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Section 7.5:

$$\mathbf{H} = \begin{bmatrix} \mathbf{y}_1^\delta & \mathbf{y}_2^\delta & \cdots & \mathbf{y}_{m_c}^\delta \\ \mathbf{y}_2^\delta & \mathbf{y}_3^\delta & \cdots & \mathbf{y}_{m_c+1}^\delta \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{y}_{m_o}^\delta & \mathbf{y}_{m_o+1}^\delta & \cdots & \mathbf{y}_{m_c+m_o-1}^\delta \end{bmatrix} \tag{9.40a}$$

$$= \begin{bmatrix} \mathbf{C}_d\mathbf{B}_d & \mathbf{C}_d\mathbf{A}_d\mathbf{B}_d & \cdots & \mathbf{C}_d\mathbf{A}_d^{m_c-1}\mathbf{B}_d \\ \mathbf{C}_d\mathbf{A}_d\mathbf{B}_d & \mathbf{C}_d\mathbf{A}_d^2\mathbf{B}_d & \cdots & \mathbf{C}_d\mathbf{A}_d^{m_c}\mathbf{B}_d \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{C}_d\mathbf{A}_d^{m_o-1}\mathbf{B}_d & \mathbf{C}_d\mathbf{A}_d^{m_o}\mathbf{B}_d & \cdots & \mathbf{C}_d\mathbf{A}_d^{m_c+m_o-2}\mathbf{B}_d \end{bmatrix}. \tag{9.40b}$$

The matrix $\mathbf{H}$ may be constructed purely from measurements $\mathbf{y}^\delta$, without separately constructing $\mathcal{O}_d$ and $\mathcal{C}_d$. Thus, we do not need access to adjoint equations.

Taking the SVD of the Hankel matrix yields the dominant temporal patterns in the time-series data:

$$\mathbf{H} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^* = \begin{bmatrix} \tilde{\mathbf{U}} & \mathbf{U}_t \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{\Sigma}} & \mathbf{0} \\ \mathbf{0} & \mathbf{\Sigma}_t \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{V}}^* \\ \mathbf{V}_t^* \end{bmatrix} \approx \tilde{\mathbf{U}}\tilde{\mathbf{\Sigma}}\tilde{\mathbf{V}}^*. \tag{9.41}$$

The small singular values in $\mathbf{\Sigma}_t$ are truncated, and only the first r singular values in $\tilde{\mathbf{\Sigma}}$ are retained. The columns of $\tilde{\mathbf{U}}$ and $\tilde{\mathbf{V}}$ are *eigen*-time-delay coordinates.

Until this point, the ERA algorithm closely resembles the BPOD procedure from Section 9.2. However, we do not require direct access to $\mathcal{O}_d$ and $\mathcal{C}_d$ or the system $(\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D})$ to construct the direct and adjoint balancing transformations. Instead, with sensor measurements from an impulse-response experiment, it is also possible to create a second, shifted Hankel matrix $\mathbf{H}'$:

$$\mathbf{H}' = \begin{bmatrix} \mathbf{y}_2 & \mathbf{y}_3^\delta & \cdots & \mathbf{y}_{m_c+1}^\delta \\ \mathbf{y}_3^\delta & \mathbf{y}_4^\delta & \cdots & \mathbf{y}_{m_c+2}^\delta \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{y}_{m_o+1}^\delta & \mathbf{y}_{m_o+2}^\delta & \cdots & \mathbf{y}_{m_c+m_o}^\delta \end{bmatrix} \tag{9.42a}$$

$$= \begin{bmatrix} \mathbf{C}_d\mathbf{A}_d\mathbf{B}_d & \mathbf{C}_d\mathbf{A}_d^2\mathbf{B}_d & \cdots & \mathbf{C}_d\mathbf{A}_d^{m_c}\mathbf{B}_d \\ \mathbf{C}_d\mathbf{A}_d^2\mathbf{B}_d & \mathbf{C}_d\mathbf{A}_d^3\mathbf{B}_d & \cdots & \mathbf{C}_d\mathbf{A}_d^{m_c+1}\mathbf{B}_d \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{C}_d\mathbf{A}_d^{m_o}\mathbf{B}_d & \mathbf{C}_d\mathbf{A}_d^{m_o+1}\mathbf{B}_d & \cdots & \mathbf{C}_d\mathbf{A}_d^{m_c+m_o-1}\mathbf{B}_d \end{bmatrix} = \mathcal{O}_d\mathbf{A}\mathcal{C}_d. \tag{9.42b}$$

Based on the matrices $\mathbf{H}$ and $\mathbf{H}'$, we are able to construct a reduced-order

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

model as follows:

$$\tilde{\mathbf{A}} = \tilde{\mathbf{\Sigma}}^{-1/2}\tilde{\mathbf{U}}^*\mathbf{H}'\tilde{\mathbf{V}}\tilde{\mathbf{\Sigma}}^{-1/2}, \tag{9.43a}$$

$$\tilde{\mathbf{B}} = \tilde{\mathbf{\Sigma}}^{1/2}\tilde{\mathbf{V}}^* \begin{bmatrix} \mathbf{I}_q \\ \mathbf{0} \end{bmatrix}, \tag{9.43b}$$

$$\tilde{\mathbf{C}} = \begin{bmatrix} \mathbf{I}_p & \mathbf{0} \end{bmatrix} \tilde{\mathbf{U}}\tilde{\mathbf{\Sigma}}^{1/2}. \tag{9.43c}$$

Here $\mathbf{I}_q$ is the $q \times q$ identity matrix, which extracts the first q columns, and $\mathbf{I}_p$ is the $p \times p$ identity matrix, which extracts the first p rows. Alternatively, $\tilde{\mathbf{B}}$ and $\tilde{\mathbf{C}}$ may be computed using the fact that $\mathbf{H} \approx \tilde{\mathbf{U}}\tilde{\mathbf{\Sigma}}\tilde{\mathbf{V}}^*$ as

$$\tilde{\mathbf{B}} = \tilde{\mathbf{\Sigma}}^{-1/2}\tilde{\mathbf{U}}^*\mathbf{H} \begin{bmatrix} \mathbf{I}_q \\ \mathbf{0} \end{bmatrix}, \tag{9.44a}$$

$$\tilde{\mathbf{C}} = \begin{bmatrix} \mathbf{I}_p & \mathbf{0} \end{bmatrix} \mathbf{H}\tilde{\mathbf{V}}\tilde{\mathbf{\Sigma}}^{-1/2}. \tag{9.44b}$$

Thus, we express the input–output dynamics in terms of a reduced system with a low-dimensional state $\tilde{\mathbf{x}} \in \mathbb{R}^r$:

$$\tilde{\mathbf{x}}_{k+1} = \tilde{\mathbf{A}}\tilde{\mathbf{x}}_k + \tilde{\mathbf{B}}\mathbf{u}, \tag{9.45a}$$

$$\mathbf{y} = \tilde{\mathbf{C}}\tilde{\mathbf{x}}_k. \tag{9.45b}$$

The Hankel matrices $\mathbf{H}$ and $\mathbf{H}'$ are constructed from impulse-response simulations/experiments, without the need for storing direct or adjoint snapshots, as in other balanced model reduction techniques. However, if full-state snapshots are available, for example, by collecting velocity fields in simulations or particle image velocimetry (PIV) experiments, it is then possible to construct direct modes. These full-state snapshots form $\boldsymbol{\mathcal{C}}_d$, and modes can be constructed by

$$\mathbf{\Psi} = \boldsymbol{\mathcal{C}}_d\tilde{\mathbf{V}}\tilde{\mathbf{\Sigma}}^{-1/2}. \tag{9.46}$$

These modes may then be used to approximate the full state of the high-dimensional system from the low-dimensional model in (9.45) by

$$\mathbf{x} \approx \mathbf{\Psi}\tilde{\mathbf{x}}. \tag{9.47}$$

If enough data is collected when constructing the Hankel matrix $\mathbf{H}$, then ERA balances the empirical controllability and observability Gramians, $\boldsymbol{\mathcal{O}}_d\boldsymbol{\mathcal{O}}_d^*$ and $\boldsymbol{\mathcal{C}}_d^*\boldsymbol{\mathcal{C}}_d$. However, if less data is collected, so that lightly damped transients do not have time to decay, then ERA will only approximately balance the system. It is instead possible to collect just enough data so that the Hankel matrix $\mathbf{H}$ reaches numerical full rank (i.e., so that remaining singular values are below

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

a threshold tolerance), and compute an ERA model. The resulting ERA model will typically have a relatively low order, given by the numerical rank of the controllability and observability subspaces. It may then be possible to apply exact balanced truncation to this smaller model, as is advocated in [724] and [462].

The code to compute ERA is provided in Code 9.2 below. Large portions of the code that format the input data into a Hankel matrix are omitted, but can be found on the book's GitHub.

Code 9.2: [MATLAB] Eigensystem realization algorithm.

```
function [Ar,Br,Cr,Dr,HSVs] = ERA(YY,m,n,nin,nout,r)

% Code to format data into Hankel matrix H omitted

[U,S,V] = svd(H,'econ');
Sigma = S(1:r,1:r);
Ur = U(:,1:r);
Vr = V(:,1:r);
Ar = Sigma^(-.5)*Ur'*H2*Vr*Sigma^(-.5);
Br = Sigma^(-.5)*Ur'*H(:,1:nin);
Cr = H(1:nout,:)*Vr*Sigma^(-.5);
HSVs = diag(S);
```

Code 9.2: [Python] Eigensystem realization algorithm.

```
def ERA(YY,m,n,nin,nout,r):

    # Code to format data into Hankel matrix H omitted

    U,S,VT = np.linalg.svd(H,full_matrices=0)
    V = VT.T
    Sigma = np.diag(S[:r])
    Ur = U[:,:r]
    Vr = V[:,:r]
    Ar = fractional_matrix_power(Sigma,-0.5) @ Ur.T @ H2 @
       Vr @ fractional_matrix_power(Sigma,-0.5)
    Br = fractional_matrix_power(Sigma,-0.5) @ Ur.T @ H[:,:
       nin]
    Cr = H[:nout,:] @ Vr @ fractional_matrix_power(Sigma
       ,-0.5)
    HSVs = S
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

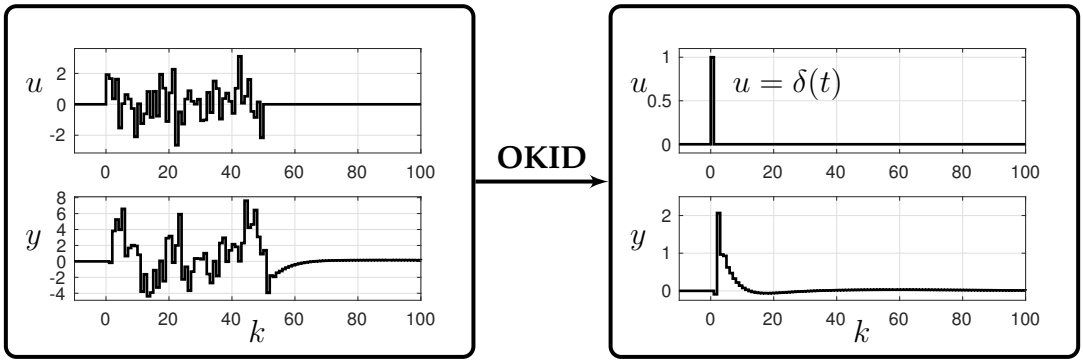


Figure 9.5: Schematic overview of OKID procedure. The output of OKID is an impulse response that can be used for system identification via ERA.

Observer Kalman Filter Identification

OKID, illustrated in Fig. 9.5, was developed to complement the ERA for lightly damped experimental systems with noise [359]. In practice, performing isolated impulse-response experiments is challenging, and the effect of measurement noise can contaminate results. Moreover, if there is a large separation of timescales, then a tremendous amount of data must be collected to use ERA. This section poses the general problem of approximating the impulse response from arbitrary input–output data. Typically, one would identify reduced-order models according to the following general procedure:

1. Collect the output in response to a pseudo-random input.
2. This information is passed through the OKID algorithm to obtain the denoised linear impulse response.
3. The impulse response is passed through the ERA to obtain a reduced-order state-space system.

The output $\mathbf{y}_k$ in response to a general input signal $\mathbf{u}_k$, for zero initial condition $\mathbf{x}_0 = \mathbf{0}$, is given by

$$\mathbf{y}_0 = \mathbf{D}_d\mathbf{u}_0, \tag{9.48a}$$
$$\mathbf{y}_1 = \mathbf{C}_d\mathbf{B}_d\mathbf{u}_0 + \mathbf{D}_d\mathbf{u}_1, \tag{9.48b}$$
$$\mathbf{y}_2 = \mathbf{C}_d\mathbf{A}_d\mathbf{B}_d\mathbf{u}_0 + \mathbf{C}_d\mathbf{B}_d\mathbf{u}_1 + \mathbf{D}_d\mathbf{u}_2, \tag{9.48c}$$
$$\vdots$$
$$\mathbf{y}_k = \mathbf{C}_d\mathbf{A}_d^{k-1}\mathbf{B}_d\mathbf{u}_0 + \mathbf{C}_d\mathbf{A}_d^{k-2}\mathbf{B}_d\mathbf{u}_1 + \cdots + \mathbf{C}_d\mathbf{B}_d\mathbf{u}_{k-1} + \mathbf{D}_d\mathbf{u}_k. \tag{9.48d}$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Note that there is no $\mathbf{C}$ term in the expression for $\mathbf{y}_0$ since there is zero initial condition $\mathbf{x}_0 = \mathbf{0}$. This progression of measurements $\mathbf{y}_k$ may be further simplified and expressed in terms of impulse-response measurements $\mathbf{y}_k^\delta$:

$$\underbrace{\begin{bmatrix}\mathbf{y}_0 & \mathbf{y}_1 & \cdots & \mathbf{y}_m\end{bmatrix}}_{\boldsymbol{\mathcal{S}}} = \underbrace{\begin{bmatrix}\mathbf{y}_0^\delta & \mathbf{y}_1^\delta & \cdots & \mathbf{y}_m^\delta\end{bmatrix}}_{\boldsymbol{\mathcal{S}}^\delta} \underbrace{\begin{bmatrix}\mathbf{u}_0 & \mathbf{u}_1 & \cdots & \mathbf{u}_m \\ \mathbf{0} & \mathbf{u}_0 & \cdots & \mathbf{u}_{m-1} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{u}_0\end{bmatrix}}_{\boldsymbol{\mathcal{B}}}. \tag{9.49}$$

It is often possible to invert the matrix of control inputs, $\boldsymbol{\mathcal{B}}$, to solve for the Markov parameters $\boldsymbol{\mathcal{S}}^\delta$. However, $\boldsymbol{\mathcal{B}}$ either may be un-invertible, or inversion may be ill conditioned. In addition, $\boldsymbol{\mathcal{B}}$ is large for lightly damped systems, making inversion computationally expensive. Finally, noise is not optimally filtered by simply inverting $\boldsymbol{\mathcal{B}}$ to solve for the Markov parameters.

The OKID method addresses each of these issues. Instead of the original discrete-time system, we now introduce an optimal observer system:

$$\hat{\mathbf{x}}_{k+1} = \mathbf{A}_d\hat{\mathbf{x}}_k + \mathbf{K}_f(\mathbf{y}_k - \hat{\mathbf{y}_\mathbf{k}}) + \mathbf{B}_d\mathbf{u}_k, \tag{9.50a}$$

$$\hat{\mathbf{y}}_k = \mathbf{C}_d\hat{\mathbf{x}}_k + \mathbf{D}_d\mathbf{u}_k, \tag{9.50b}$$

which may be rewritten as

$$\hat{\mathbf{x}}_{k+1} = \underbrace{(\mathbf{A}_d - \mathbf{K}_f\mathbf{C}_d)}_{\bar{\mathbf{A}}_d}\hat{\mathbf{x}}_k + \underbrace{\begin{bmatrix}\mathbf{B}_d - \mathbf{K}_f\mathbf{D}_d, & \mathbf{K}_f\end{bmatrix}}_{\bar{\mathbf{B}}_d}\begin{bmatrix}\mathbf{u}_k \\ \mathbf{y}_k\end{bmatrix}. \tag{9.51}$$

Recall from above that if, the system is observable, it is possible to place the poles of $\mathbf{A}_d - \mathbf{K}_f\mathbf{C}_d$ anywhere we like. However, depending on the amount of noise in the measurements, the magnitude of process noise, and the uncertainty in our model, there are *optimal* pole locations that are given by the *Kalman filter* (recall Section 8.5). We may now solve for the *observer Markov parameters* $\bar{\boldsymbol{\mathcal{S}}}^\delta$ of the system in (9.51) in terms of measured inputs and outputs according to the following algorithm from [359]:

1. Choose the number of observer Markov parameters to identify, l.

2. Construct the data matrices below:

$$\boldsymbol{\mathcal{S}} = \begin{bmatrix}\mathbf{y}_0 & \mathbf{y}_1 & \cdots & \mathbf{y}_l & \cdots & \mathbf{y}_m\end{bmatrix}, \tag{9.52}$$

$$\boldsymbol{\mathcal{V}} = \begin{bmatrix}\mathbf{u}_0 & \mathbf{u}_1 & \cdots & \mathbf{u}_l & \cdots & \mathbf{u}_m \\ \mathbf{0} & \mathbf{v}_0 & \cdots & \mathbf{v}_{l-1} & \cdots & \mathbf{v}_{m-1} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{v}_0 & \cdots & \mathbf{v}_{m-l}\end{bmatrix}, \tag{9.53}$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

where $\mathbf{v}_i = \begin{bmatrix}\mathbf{u}_i^T & \mathbf{y}_i^T\end{bmatrix}^T$.

The matrix $\boldsymbol{\mathcal{V}}$ resembles $\boldsymbol{\mathcal{B}}$, except that it has been augmented with the outputs $\mathbf{y}_i$. In this way, we are working with a system that is augmented to include a Kalman filter. We are now identifying the observer Markov parameters of the *augmented* system, $\bar{\boldsymbol{\mathcal{S}}}^\delta$, using the equation $\boldsymbol{\mathcal{S}} = \bar{\boldsymbol{\mathcal{S}}}^\delta \boldsymbol{\mathcal{V}}$. It will be possible to identify these observer Markov parameters from the data and then extract the impulse response (Markov parameters) of the original system.

3. Identify the matrix $\bar{\boldsymbol{\mathcal{S}}}^\delta$ of observer Markov parameters by solving $\boldsymbol{\mathcal{S}} = \bar{\boldsymbol{\mathcal{S}}}^\delta \boldsymbol{\mathcal{V}}$ for $\bar{\boldsymbol{\mathcal{S}}}^\delta$ using the right pseudo-inverse of $\boldsymbol{\mathcal{V}}$ (i.e., SVD).

4. Recover system Markov parameters, $\boldsymbol{\mathcal{S}}^\delta$, from the observer Markov parameters, $\bar{\boldsymbol{\mathcal{S}}}^\delta$:

   (a) Order the observer Markov parameters $\bar{\boldsymbol{\mathcal{S}}}^\delta$ as

   $$\bar{\boldsymbol{\mathcal{S}}}_0^\delta = \mathbf{D}, \tag{9.54}$$

   $$\bar{\boldsymbol{\mathcal{S}}}_k^\delta = \begin{bmatrix}(\bar{\boldsymbol{\mathcal{S}}}^\delta)_k^{(1)} & (\bar{\boldsymbol{\mathcal{S}}}^\delta)_k^{(2)}\end{bmatrix} \quad \text{for } k \geq 1, \tag{9.55}$$

   where $(\bar{\boldsymbol{\mathcal{S}}}^\delta)_k^{(1)} \in \mathbb{R}^{p\times q}$, $(\bar{\boldsymbol{\mathcal{S}}}^\delta)_k^{(2)} \in \mathbb{R}^{p\times p}$, and $\mathbf{y}_0^\delta = \bar{\boldsymbol{\mathcal{S}}}_0^\delta = \mathbf{D}$.

   (b) Reconstruct system Markov parameters as

   $$\mathbf{y}_k^\delta = (\bar{\boldsymbol{\mathcal{S}}}^\delta)_k^{(1)} + \sum_{i=1}^{k} (\bar{\boldsymbol{\mathcal{S}}}^\delta)_i^{(2)} \mathbf{y}_{k-i}^\delta \quad \text{for } k \geq 1. \tag{9.56}$$

Thus, the OKID method identifies the Markov parameters of a system augmented with an asymptotically stable Kalman filter. The system Markov parameters are extracted from the observer Markov parameters by (9.56). These system Markov parameters approximate the impulse response of the system, and may be used directly as inputs to the ERA algorithm. A code to compute OKID is provided in both MATLAB and Python on the book's GitHub.

ERA/OKID has been widely applied across a range of system identification tasks, including to identify models of aeroelastic structures and fluid dynamic systems. There are numerous extensions of the ERA/OKID methods. For example, there are generalizations for linear parameter varying (LPV) systems and systems linearized about a limit cycle.

Combining ERA and OKID

Here we demonstrate ERA and OKID on the same model system from Section 9.2. Because ERA yields the same balanced models as BPOD, the reduced system responses should be the same.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

First, Code 9.3 computes an impulse response of the full system, and uses this as an input to ERA.

Code 9.3: [MATLAB] Compute impulse response and use ERA to generate model.

```
%% Obtain impulse response of full system
[yFull,t] = impulse(sysFull,0:1:(r*5)+1);
YY = permute(yFull,[2 3 1]); % Reorder to be size p x q x m
                             % (default is m x p x q)

%% Compute ERA from impulse response
mco = floor((length(yFull)-1)/2);  % m_c = m_o = (m-1)/2
[Ar,Br,Cr,Dr,HSVs] = ERA(YY,mco,mco,numInputs,numOutputs,r);
sysERA = ss(Ar,Br,Cr,Dr,-1);
```

Code 9.3: [Python] Compute impulse response and use ERA to generate model.

```
for qi in range(q):
    yFull[:,:,qi],t = impulse(sysFull,T=tspan,input=qi)
YY = np.transpose(yFull,axes=(1,2,0)) # reorder to p x q x m

## Compute ERA from impulse response
mco = int(np.floor((yFull.shape[0]-1)/2)) # m_c=m_o=(m-1)/2
Ar,Br,Cr,Dr,HSVs = ERA(YY,mco,mco,q,p,r)
sysERA = ss(Ar,Br,Cr,Dr,1)
```

Next, if an impulse response is unavailable, it is possible to excite the system with a random input signal and use OKID to extract an impulse response. This impulse response is then used by ERA to extract the model, as in Code 9.4.

Code 9.4: [MATLAB] Approximate impulse response with OKID and use ERA to generate model.

```
%% Compute random input simulation for OKID
uRandom = randn(numInputs,200);  % Random forcing input
yRandom = lsim(sysFull,uRandom,1:200)'; % Output

%% Compute OKID and then ERA
H = OKID(yRandom,uRandom,r);
mco = floor((length(H)-1)/2);  % m_c = m_o
[Ar,Br,Cr,Dr,HSVs] = ERA(H,mco,mco,numInputs,numOutputs,r);
sysERAOKID = ss(Ar,Br,Cr,Dr,-1);
```

Code 9.4: [Python] Approximate impulse response with OKID and use ERA to generate model.

```
## Compute random input simulation for OKID
uRandom = np.random.randn(q,200) # Random forcing input
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

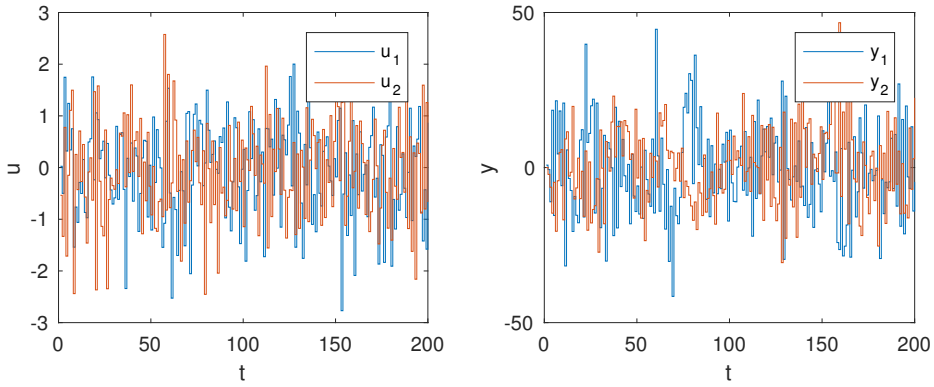


Figure 9.6: Input–output data used by OKID.

```
yRandom = lsim(sysFull,uRandom,range(200))[0].T # Output

## Compute OKID and then ERA
H = OKID(yRandom,uRandom,r)
mco = int(np.floor((H.shape[2]-1)/2)) # m_c = m_o
Ar,Br,Cr,Dr,HSVs = ERA(H,mco,mco,q,p,r)
sysERAOKID = ss(Ar,Br,Cr,Dr,1)
```

Figure 9.6 shows the input–output data used by OKID to approximate the impulse response. The impulse responses of the resulting systems are computed as in Code 9.5.

Code 9.5: [MATLAB] Plot impulse responses of various models.

```
[y1,t1] = impulse(sysFull,0:1:200);
[y2,t2] = impulse(sysERA,0:1:100);
[y3,t3] = impulse(sysERAOKID,0:1:100);
```

Code 9.5: [Python] Plot impulse responses of various models.

```
for qi in range(q):
    y1[:,:,qi],t1 = impulse(sysFull,np.arange(200),input=qi)
    y2[:,:,qi],t2 = impulse(sysERA,np.arange(100),input=qi)
    y3[:,:,qi],t3 = impulse(sysERAOKID,np.arange(100),input=
       qi)
```

Finally, the system responses can be seen in Fig. 9.7. The low-order ERA and ERA/OKID models closely match the full model and have similar performance to the BPOD models described above. Because ERA and BPOD are mathematically equivalent, this agreement is not surprising. However, the ability of ERA/OKID to extract a reduced-order model from the random input data in Fig. 9.6 is quite remarkable. Moreover, unlike BPOD, these methods are readily

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

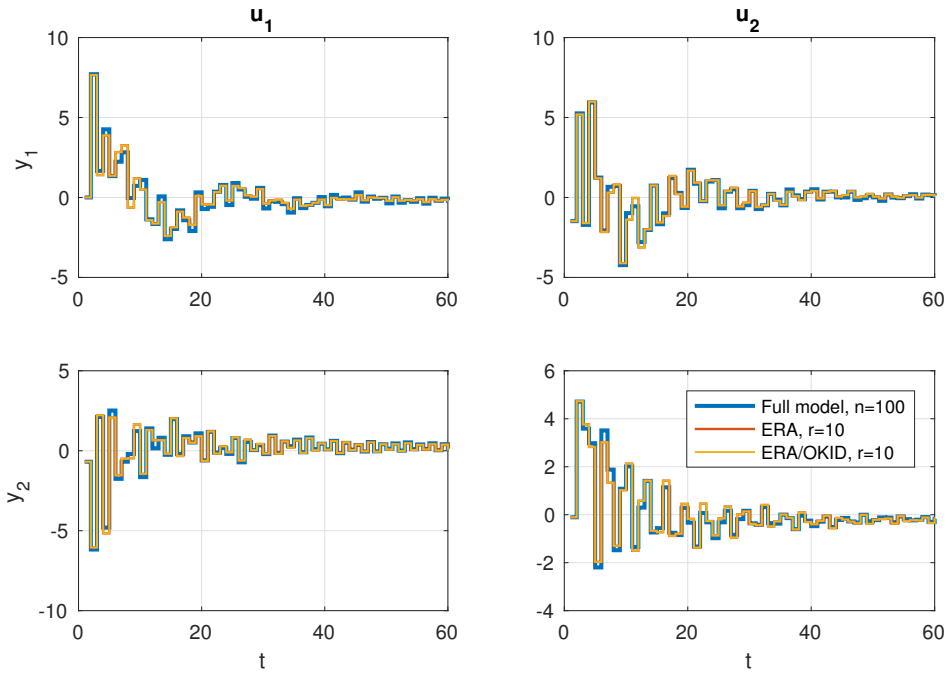


Figure 9.7: Impulse response of full-state model with $n = 100$ and $p = q = 2$, along with ERA and ERA/OKID models with $r = 10$.

applicable to experimental measurements, as they do not require non-physical adjoint equations.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Suggested Reading

Papers and reviews

(1) Principal component analysis in linear systems: Controllability, observability, and model reduction, by B. C. Moore, *IEEE Transactions on Automatic Control*, 1981 [509].

(2) Identification of linear parameter varying models, by B. Bamieh and L. Giarré, *International Journal of Robust and Nonlinear Control*, 2002 [47].

(3) Balanced model reduction via the proper orthogonal decomposition, by K. Willcox and J. Peraire, *AIAA Journal*, 2002 [755].

(4) Model reduction for fluids using balanced proper orthogonal decomposition, by C. W. Rowley, *International Journal of Bifurcations and Chaos*, 2005 [608].

(5) An eigensystem realization algorithm for modal parameter identification and model reduction, by J. N. Juang and R. S. Pappa, *Journal of Guidance, Control, and Dynamics*, 1985 [358].

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Homework

Exercise 9-1. Generate a random state-space system, as in the example used to generate Fig. 9.4. For various model orders r, compute the exact balanced truncation model at this order. Compute the relative error between the truncated model and the full model in at least two different system norms; there are several system norms, such as the 2-norm and the infinity-norm. Plot the various relative error norms versus the model order r.

Now compute the relative Frobenius norm error of the truncated model impulse response compared to the impulse response of the full model; to compute the Frobenius norm, simply vectorize the impulse-response data and compute the 2-norm of the resulting vector. Plot the relative Frobenius norm error versus the model order r, and compare the with the system norms above.

Exercise 9-2. For the exercise above, we will now compare balanced truncation with output truncation, input truncation, and truncation based on the eigenvalues of the $\mathbf{A}$ matrix.

For output truncation, you will compute the truncated projection basis by computing the SVD of the observability matrix and retaining the states corresponding to the first r singular vectors; alternatively, you may use the first r leading eigenvectors of the observability Gramian. For input truncation, you will do the same thing, but for the controllability matrix (respectively, controllability Gramian). For truncation based on the eigenvalues of $\mathbf{A}$, you will compute a truncated projection basis using the eigenvectors of $\mathbf{A}$, with the retained states corresponding to the r least-damped eigenvalues (i.e., the eigenvalues with the most positive or least negative real part for continuous-time dynamics, or the eigenvalues with the largest radius for discrete-time dynamics). In this case, you may want to manually order the eigenvalues and eigenvectors.

In all cases above, for a given model order r, you will use the computed basis to truncate the system. Reproduce Fig. 9.4 with these new forms of truncation and discuss the results. Also create a plot of the relative error between the truncated models and the full model versus model order r.

Exercise 9-3. This exercise will explore balanced *residualization*, which is an alternative to balanced truncation.

(a) First, create a single-input, single-output random state-space system with $n = 100$. Now, plot the frequency response (i.e., Bode plot) of the full model and the balanced truncation models for various model orders r. Comment on how they agree and disagree.

(b) In balanced residualization, instead of truncating the $\mathbf{z}_t$ state in (9.25) en-

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

tirely, the equation $\dot{\mathbf{z}}_t = \mathbf{0}$ is solved for $\mathbf{z}_t$, and this is substituted into the $\tilde{\mathbf{x}}$ and $\mathbf{y}$ equations.

(c) Write down the balanced residualization equations based on (9.25). Use this formulation to compute the balanced residualization model of order $r = 10$ and reproduce Fig. 9.4 with this new residualized model. You can also use the "`MatchDC`" option in MATLAB to check your results.

(d) Plot the frequency response of the full model and the balanced residualization models for various model orders r. Discuss how these differ from the balanced truncation models.

Exercise 9-4. This exercise will explore the ERA/OKID procedure for model identification, including how different input forcing signals affect the ability of ERA/OKID to identify a model.

(a) First, reproduce the ERA/OKID results in Fig. 9.7 for model order $r = 10$. Now, compute the relative Frobenius norm error between the full model and each of the ERA and ERA/OKID models for all model orders r between $r = 1$ and $r = 10$, as well as model orders $r = 20$, $r = 30$, $r = 40$, and $r = 50$. To compute the Frobenius norm of the multiple-input, multiple-output impulse response, simply vectorize the impulse-response data and compute the 2-norm of the vector. Plot the relative error versus model order.

(b) Now, add a small amount of Gaussian white noise to the randomly forced data that is input to the ERA/OKID procedure. How does the noise magnitude affect the fidelity of the models of various order?

(c) Repeat the ERA/OKID results in Fig. 9.7 for model order $r = 10$, but instead of Gaussian white noise forcing for $\mathbf{u}$, use a quadratic chirp signal as in (2.51). Compare the impulse response of the ERA/OKID system with that of the full model.

(d) Repeat part (c) above, but using a pseudo-random sequence of step functions in the control $\mathbf{u}$; i.e., the control input steps to different random values at randomly timed spacing with minimum time between changes of 5. Does the model fidelity change when the sequence of step functions can have different amplitudes versus when they are constrained to have the same amplitude of either 0 or 1?

(e) Repeat part (c) above using a pure tone sine wave for the control $\mathbf{u}$. Now, instead of a pure tone sine wave, use a square-wave pulse train with the same frequency. Explain the similarities or differences in the identified models. Compute Bode plots for each identified model and compare with the true Bode plot.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

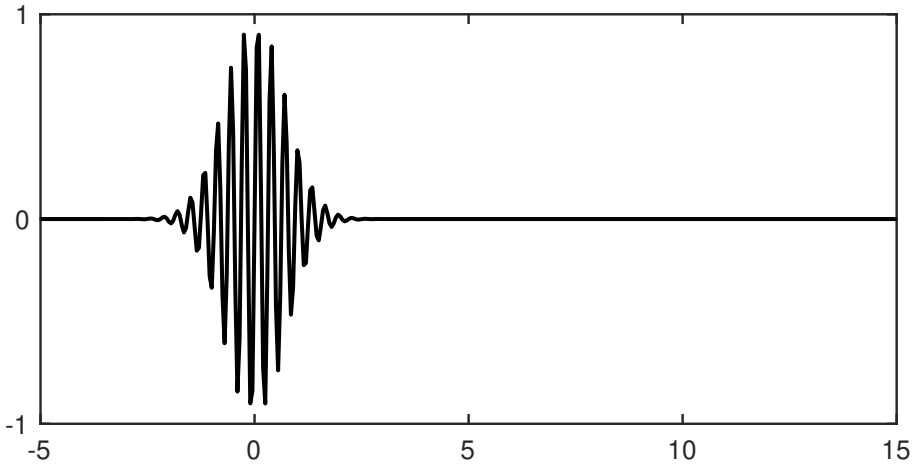
Exercise 9-5. Download a system matrix from the SLICOT benchmark website `http://slicot.org/20-site/126-benchmark-examples-for-model-reduction`. Repeat all of Exercise 9-4 above on this test system.

Exercise 9-6. In this exercise, we will explore how ERA handles spatio-temporal data, such as a decaying traveling wave.

Generate data for a traveling, decaying wave pulse

$$f(x,t) = e^{\lambda t} \exp^{-(x-ct)^2} \sin(\omega x),$$

shown below. Begin with the parameters $\lambda = -0.05$, $c = 1$, and $\omega = 20$. Use a spatial domain of $x \in [-5, 15]$ with $\Delta x = 0.05$ and a temporal domain of $t \in [0, 10]$ with $\Delta t = 0.05$. Plot this data, either as a movie or as a waterfall plot.



Now, use this data to train ERA models of various orders. Explore the performance of these models at different orders. Explain your results.

Exercise 9-7. Describe the connections between ERA and DMDc. How are the algorithms connected?

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Part IV

Advanced Data-Driven Modeling and Control

Chapter 10

Data-Driven Control

As described in Chapter 8, control design often begins with a model of the system being controlled. Notable exceptions include model-free adaptive control strategies, reinforcement learning, and many uses of proportional–integral–derivative (PID) control. For mechanical systems of moderate dimension, it may be possible to write down a model (e.g., based on the Newtonian, Lagrangian, or Hamiltonian formalism) and linearize the dynamics about a fixed point or periodic orbit. However, for modern systems of interest, as are found in neuroscience, turbulence, epidemiology, climate, and finance, typically there are no simple models suitable for control design. Chapter 9 described techniques to obtain control-oriented reduced-order models for high-dimensional systems from data, but these approaches are limited to *linear* systems. Real-world systems are usually nonlinear and the control objective is not readily achieved via linear techniques. Nonlinear control can still be posed as an optimization problem with a high-dimensional, non-convex cost function landscape with multiple local minima. Modern data-driven methods, such as machine learning, are complementary, as they constitute a growing set of techniques that may be broadly described as performing nonlinear optimization in a high-dimensional space from data.

This chapter describes data-driven control techniques that are specifically designed for systems that lack a principled model. Thus, we describe emerging techniques that use machine learning to characterize and control strongly nonlinear, high-dimensional, and multi-scale systems, leveraging the increasing availability of high-quality measurement data. Machine learning techniques may be used (1) to characterize a system for later use with model-based control, or (2) to directly characterize a control law that effectively interacts with a system. This is illustrated schematically in Fig. 10.1, where data-driven techniques may be applied to either the *System* or *Controller* blocks. Related methods may also be used to identify good sensors and actuators, as discussed previously in Section 3.8. Section 10.1 will introduce model predictive control (MPC), which is a powerful and flexible approach for controlling nonlinear systems with con-

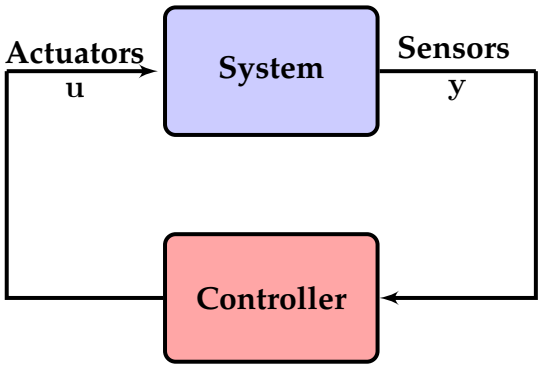


Figure 10.1: In the standard control framework from Chapter 8, machine learning may be used (1) to develop a model of the system or (2) to learn a controller.

straints and uncertainty. However, MPC relies on a system model, and so Section 10.2 demonstrates how to use machine learning and system identification to learn nonlinear input–output models that may be used with MPC. In Section 10.3 we explore machine learning techniques to directly identify controllers from input–output data. Here we explore the use of genetic algorithms to learn control laws, as demonstrated on a simple example of tuning a PID controller. It is important to emphasize the breadth and depth of this field, and there are many powerful methods, including reinforcement learning, which is the subject of Chapter 11. Finally, in Section 10.4 we describe the adaptive extremum-seeking control strategy, which optimizes the control signal based on how the system responds to perturbations.

10.1 Model Predictive Control (MPC)

Model predictive control (MPC) [149, 231, 262, 263, 434, 512, 573, 574, 587] has become a cornerstone of modern process control and is ubiquitous in the industrial landscape. MPC is used to control strongly nonlinear systems with constraints, time delays, non-minimum-phase dynamics, and instability. Most industrial applications of MPC use empirical models based on linear system identification (see Chapter 8), neural networks (see Chapter 6), Volterra series [102, 118], and autoregressive models [8] (e.g., ARX, ARMA, NARX, and NARMAX). Recently, deep learning and reinforcement learning have been combined with MPC [438, 773] with impressive results. However, deep learning requires large volumes of data and may not be readily interpretable. A complementary line of research seeks to identify models for MPC based on limited data to characterize systems in response to abrupt changes. For example, Kaiser et al. [366] recently showed that it is possible to rapidly identify DMD and SINDy models from Chapter 7, based on limited data, and then use these

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

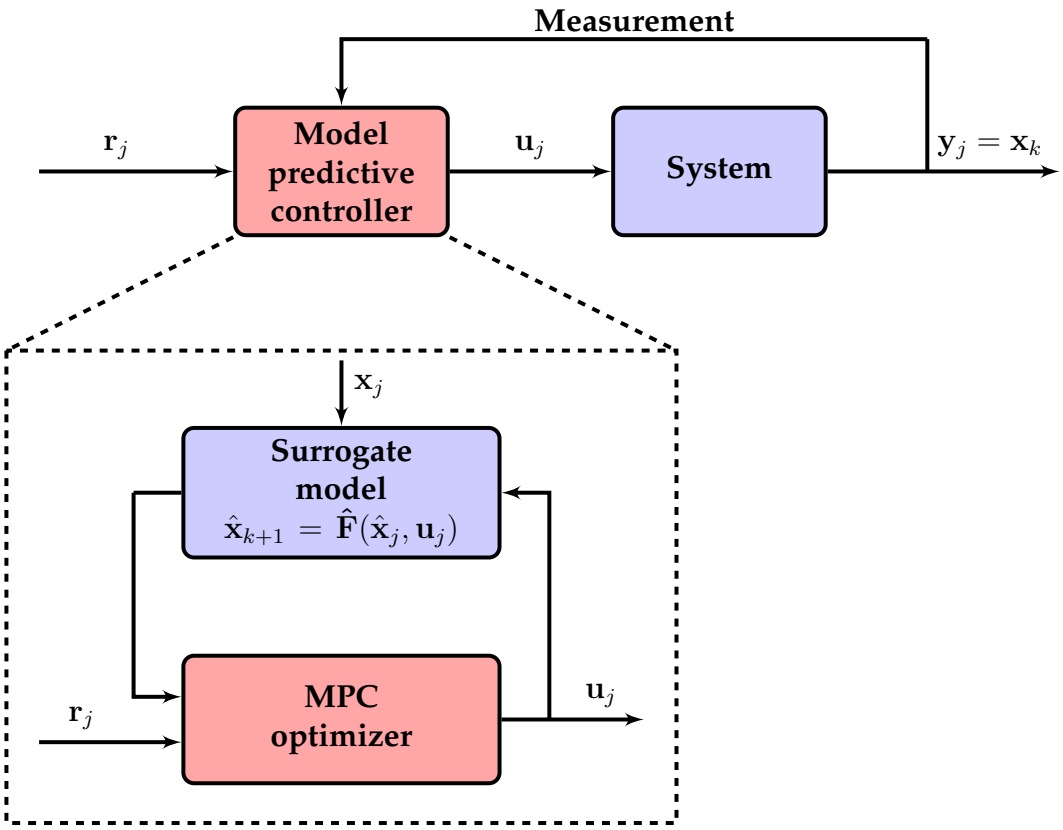


Figure 10.2: Schematic of model predictive control, where a surrogate model is used to run an optimization directly inside the control loop. This diagram assumes full-state measurements $\mathbf{y} = \mathbf{x}$ for simplicity, although this is not strictly necessary for MPC.

for MPC.

Model predictive control is shown schematically in Fig. 10.2. MPC determines the next immediate control action by solving a constrained optimal control problem over a receding horizon. In particular, the open-loop actuation signal $\mathbf{u}$ is optimized on a receding time horizon $t_c = m_c \Delta t$ to minimize a cost J over some prediction horizon $t_p = m_p \Delta t$. The control horizon is typically less than or equal to the prediction horizon, and the control is held constant between t_c and t_p. The cost function is typically of a form similar to an LQR cost function

$$J(\mathbf{x}_j) = \sum_{k=0}^{m_p-1} \|\hat{\mathbf{x}}_{j+k} - \mathbf{r}_{j+k}\|_{\mathbf{Q}}^2 + \sum_{k=1}^{m_c-1} (\|\mathbf{u}_{j+k}\|_{\mathbf{R}}^2 + \|\Delta \mathbf{u}_{j+k}\|_{\mathbf{R}_\Delta}^2), \tag{10.1}$$

where $\mathbf{r}_j$ is the reference trajectory to be tracked by the MPC, $\hat{\mathbf{x}}$ is the predicted

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

state, $\|\mathbf{x}\|_{\mathbf{Q}}^2 = \mathbf{x}^T\mathbf{Q}\mathbf{x}$, and there is an additional penalty on large changes in the control, i.e., on $\Delta\mathbf{u}_j = \mathbf{u}_j - \mathbf{u}_{j-1}$. Note that the weight matrix $\mathbf{Q}$ must be positive semi-definite, and $\mathbf{R}$ and $\mathbf{R}_\Delta$ must be positive semi-definite, as in LQR. It is also possible to add a terminal cost on the final state. This cost function is then optimized over the control sequences $\{\mathbf{u}_{j+1}, \ldots, \mathbf{u}_{j+k}, \ldots, \mathbf{u}_{j+m_c}\}$ subject to a surrogate model

$$\hat{\mathbf{x}}_{k+1} = \hat{\mathbf{F}}(\hat{\mathbf{x}}_j, \mathbf{u}_j) \tag{10.2}$$

with constraints on the inputs $\mathbf{u}_j$,

$$\mathbf{u}_{\min} \leq \mathbf{u}_j \leq \mathbf{u}_{\max}, \tag{10.3}$$

and on $\Delta\mathbf{u}_j$,

$$\Delta\mathbf{u}_{\min} \leq \Delta\mathbf{u}_j \leq \Delta\mathbf{u}_{\max}. \tag{10.4}$$

The optimal control is then applied for one time-step, and the procedure is repeated and the receding horizon control re-optimized at each subsequent time-step. This results in the control law

$$\mathbf{K}(\mathbf{x}_j) = \mathbf{u}_{j+1}(\mathbf{x}_j), \tag{10.5}$$

where $\mathbf{u}_{j+1}$ is the first time-step of the optimized actuation starting at $\mathbf{x}_j$. This is illustrated in Fig. 10.3. For more details, see Kaiser et al. [366] and Fonzi et al. [247].

It is possible to optimize highly customized cost functions, subject to nonlinear dynamics, with constraints on the actuation and state. However, the computational requirements of re-optimizing at each time-step are considerable, putting limits on the complexity of the model and optimization techniques. Fortunately, rapid advances in computing power and optimization are enabling MPC for real-time nonlinear control.

There have been tremendous recent advances in MPC, especially deep MPC, which uses deep learning for the surrogate model [438]. Deep MPC has been used in a wide range of applications, including for vision-based driving systems [212], controlling laser systems [58], fluid flows [84, 513], and aeroelastic systems [247]. Tube MPC is a robust strategy that keeps the system within a tube around the reference trajectory [242, 458], which is useful for safety-critical systems with model error and disturbances. More generally, developing robust and distributed MPC algorithms with guarantees [600, 601] is a major avenue of research in autonomy and robotics. MPC may also be used for model-based reinforcement learning [756], essentially codifying the MPC controller into a reinforcement learning policy. Finally, recent years have seen efforts to combine differentiable programming, which is a key enabler of modern neural network training, with MPC, resulting in differentiable predictive control [13, 213]

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

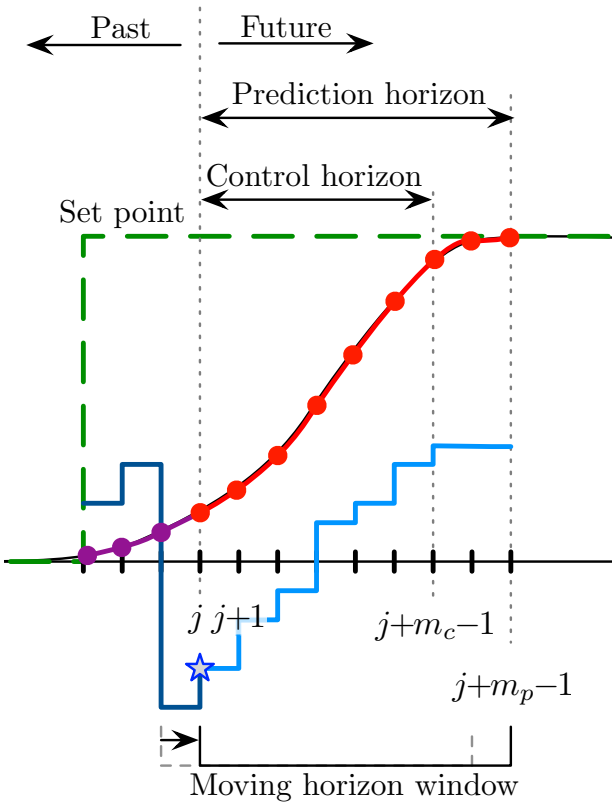


Figure 10.3: Illustration of model predictive control used to track a set point, where the actuation input $\mathbf{u}$ is iteratively optimized over a receding horizon. Reproduced with permission from Kaiser et al. [366].

10.2 Nonlinear System Identification for Control

The data-driven modeling and control of complex systems is undergoing a revolution, driven by the rise of big data, advanced algorithms in machine learning and optimization, and modern computational hardware. Despite the increasing use of equation-free and adaptive control methods, there remains a wealth of powerful model-based control techniques, such as linear optimal control (see Chapter 8) and model predictive control (MPC) [149, 262]. Increasingly, these model-based control strategies are aided by data-driven techniques that characterize the input–output dynamics of a system of interest from measurements alone, without relying on first-principles modeling. Broadly speaking, this is known as *system identification*, which has a long and rich history in control theory going back decades to the time of Kalman. However, with increasingly powerful data-driven techniques, such as those described in Chapter 7, nonlinear system identification is the focus of renewed interest.

The goal of system identification is to identify a low-order model of the input–output dynamics from actuation $\mathbf{u}$ to measurements $\mathbf{y}$. If we are able

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

to measure the full state $\mathbf{x}$ of the system, then this reduces to identifying the dynamics $\mathbf{f}$ that satisfy

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{x} = \mathbf{f}(\mathbf{x}, \mathbf{u}). \tag{10.6}$$

This problem may be formulated in discrete time, since data is typically collected at discrete instants in time and control laws are often implemented digitally. In this case, the dynamics read

$$\mathbf{x}_{k+1} = \mathbf{F}(\mathbf{x}_k, \mathbf{u}_k). \tag{10.7}$$

When the dynamics are approximately linear, we may identify a linear system

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k, \tag{10.8}$$

which is the approach taken in the dynamic mode decomposition with control (DMDc) algorithm below.

It may also be advantageous to identify a set of measurements $\mathbf{y} = \mathbf{g}(\mathbf{x})$, in which the unforced nonlinear dynamics appear linear:

$$\mathbf{y}_{k+1} = \mathbf{A}_\mathbf{Y}\mathbf{y}_k. \tag{10.9}$$

This is the approach taken in the Koopman control method below. In this way, nonlinear dynamics may be estimated and controlled using standard textbook linear control theory in the intrinsic coordinates $\mathbf{y}$ [365, 404].

Finally, the nonlinear dynamics in (10.6) or (10.7) may be identified directly using the SINDy with control algorithm. The resulting models may be used with model predictive control for the control of fully nonlinear systems [366].

DMD with Control

Proctor et al. [570] extended the DMD algorithm to include the effect of actuation and control, in the so-called DMD with control (DMDc) algorithm. It was observed that naively applying DMD to data from a system with actuation would often result in incorrect dynamics, as the effects of internal dynamics are confused with the effects of actuation. DMDc was originally motivated by the problem of characterizing and controlling the spread of disease, where it is unreasonable to stop intervention efforts (e.g., vaccinations) just to obtain a characterization of the unforced dynamics [568]. Instead, if the actuation signal is measured, a new DMD regression may be formulated in order to disambiguate the effect of internal dynamics from that of actuation and control. Subsequently, this approach has been extended to perform DMDc on heavily subsampled or compressed measurements by Bai et al. [41].

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

The DMDc method seeks to identify the best-fit linear operators $\mathbf{A}$ and $\mathbf{B}$ that approximately satisfy the following dynamics on measurement data:

$$\mathbf{x}_{k+1} \approx \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k. \tag{10.10}$$

In addition to the snapshot matrix $\mathbf{X} = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_m \end{bmatrix}$ and the time-shifted snapshot matrix $\mathbf{X}' = \begin{bmatrix} \mathbf{x}_2 & \mathbf{x}_3 & \cdots & \mathbf{x}_{m+1} \end{bmatrix}$ from (7.24), a matrix of the actuation input history is assembled:

$$\mathbf{\Upsilon} = \begin{bmatrix} | & | & & | \\ \mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_m \\ | & | & & | \end{bmatrix}. \tag{10.11}$$

The dynamics in (10.10) may be written in terms of the data matrices:

$$\mathbf{X}' \approx \mathbf{A}\mathbf{X} + \mathbf{B}\mathbf{\Upsilon}. \tag{10.12}$$

As in the DMD algorithm (see Section 7.2), the leading eigenvalues and eigenvectors of the best-fit linear operator $\mathbf{A}$ are obtained via dimensionality reduction and regression. If the actuation matrix $\mathbf{B}$ is known, then it is straightforward to correct for the actuation and identify the spectral decomposition of $\mathbf{A}$ by replacing $\mathbf{X}'$ with $\mathbf{X}' - \mathbf{B}\mathbf{\Upsilon}$ in the DMD algorithm:

$$(\mathbf{X}' - \mathbf{B}\mathbf{\Upsilon}) \approx \mathbf{A}\mathbf{X}. \tag{10.13}$$

When $\mathbf{B}$ is unknown, both $\mathbf{A}$ and $\mathbf{B}$ must be simultaneously identified. In this case, the dynamics in (10.12) may be recast as

$$\mathbf{X}' \approx \begin{bmatrix} \mathbf{A} & \mathbf{B} \end{bmatrix} \begin{bmatrix} \mathbf{X} \\ \mathbf{\Upsilon} \end{bmatrix} = \mathbf{G}\mathbf{\Omega}, \tag{10.14}$$

and the matrix $\mathbf{G} = \begin{bmatrix} \mathbf{A} & \mathbf{B} \end{bmatrix}$ is obtained via least-squares regression:

$$\mathbf{G} \approx \mathbf{X}'\mathbf{\Omega}^\dagger. \tag{10.15}$$

The matrix $\mathbf{\Omega} = \begin{bmatrix} \mathbf{X}^* & \mathbf{\Upsilon}^* \end{bmatrix}^*$ is generally a high-dimensional data matrix, which may be approximated using the SVD:

$$\mathbf{\Omega} = \tilde{\mathbf{U}}\tilde{\mathbf{\Sigma}}\tilde{\mathbf{V}}^*. \tag{10.16}$$

The matrix $\tilde{\mathbf{U}}$ must be split into two matrices, $\tilde{\mathbf{U}} = \begin{bmatrix} \tilde{\mathbf{U}}_1^* & \tilde{\mathbf{U}}_2^* \end{bmatrix}^*$, to provide bases for $\mathbf{X}$ and $\mathbf{\Upsilon}$. Unlike the DMD algorithm, $\tilde{\mathbf{U}}$ provides a reduced basis for the *input space*, while $\hat{\mathbf{U}}$ from

$$\mathbf{X}' = \hat{\mathbf{U}}\hat{\mathbf{\Sigma}}\hat{\mathbf{V}}^* \tag{10.17}$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

defines a reduced basis for the *output space*. It is then possible to approximate $\mathbf{G} = \begin{bmatrix}\mathbf{A} & \mathbf{B}\end{bmatrix}$ by projecting onto this basis:

$$\tilde{\mathbf{G}} = \hat{\mathbf{U}}^* \mathbf{G} \begin{bmatrix}\hat{\mathbf{U}} \\ \mathbf{I}\end{bmatrix}. \tag{10.18}$$

The resulting projected matrices $\tilde{\mathbf{A}}$ and $\tilde{\mathbf{B}}$ in $\tilde{\mathbf{G}}$ are

$$\tilde{\mathbf{A}} = \hat{\mathbf{U}}^* \mathbf{A} \hat{\mathbf{U}} = \hat{\mathbf{U}}^* \mathbf{X}' \tilde{\mathbf{V}} \tilde{\mathbf{\Sigma}}^{-1} \tilde{\mathbf{U}}_1^* \hat{\mathbf{U}}, \tag{10.19a}$$

$$\tilde{\mathbf{B}} = \hat{\mathbf{U}}^* \mathbf{B} \quad = \hat{\mathbf{U}}^* \mathbf{X}' \tilde{\mathbf{V}} \tilde{\mathbf{\Sigma}}^{-1} \tilde{\mathbf{U}}_2^*. \tag{10.19b}$$

More importantly, it is possible to recover the DMD eigenvectors $\mathbf{\Phi}$ from the eigendecomposition $\tilde{\mathbf{A}}\mathbf{W} = \mathbf{W}\mathbf{\Lambda}$:

$$\mathbf{\Phi} = \mathbf{X}' \tilde{\mathbf{V}} \tilde{\mathbf{\Sigma}}^{-1} \tilde{\mathbf{U}}_1^* \hat{\mathbf{U}} \mathbf{W}. \tag{10.20}$$

Ambiguity in Identifying Closed-Loop Systems

For systems that are being actively controlled via feedback, with $\mathbf{u} = -\mathbf{K}\mathbf{x}$,

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k \tag{10.21a}$$

$$= \mathbf{A}\mathbf{x}_k - \mathbf{B}\mathbf{K}\mathbf{x}_k \tag{10.21b}$$

$$= (\mathbf{A} - \mathbf{B}\mathbf{K})\mathbf{x}_k, \tag{10.21c}$$

it is impossible to disambiguate the dynamics $\mathbf{A}$ and the actuation $\mathbf{BK}$. In this case, it is important to add perturbations to the actuation signal $\mathbf{u}$ to provide additional information. These perturbations may be a white noise process or occasional impulses that provide a kick to the system, providing a signal to disambiguate the dynamics from the feedback signal.

Koopman Operator Nonlinear Control

For nonlinear systems, it may be advantageous to identify data-driven coordinate transformations that make the dynamics appear linear. These coordinate transformations are related to *intrinsic* coordinates defined by eigenfunctions of the Koopman operator (see Section 7.4). Koopman analysis has thus been leveraged for nonlinear estimation [679, 680] and control [365, 404, 558].

It is possible to design estimators and controllers directly from DMD or eDMD models, and Korda et al. [404] used model predictive control (MPC) to control nonlinear systems with eDMD models. MPC performance is also surprisingly good for DMD models, as shown in Kaiser et al. [366]. In addition, Peitz et al. [558] demonstrated the use of MPC for switching control between a small number of actuation values to track a reference value of lift in

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

an unsteady fluid flow; for each constant actuation value, a separate eDMD model was characterized. Surana [679] and Surana and Banaszuk [680] have also demonstrated excellent nonlinear estimators based on Koopman Kalman filters. However, as discussed previously, eDMD models may contain many spurious eigenvalues and eigenvectors because of closure issues related to finding a Koopman-invariant subspace. Instead, it may be advantageous to identify a handful of relevant Koopman eigenfunctions and perform control directly in these coordinates [365].

In Section 7.5, we described several strategies to approximate Koopman eigenfunctions, $\varphi(\mathbf{x})$, where the dynamics become linear:

$$\frac{\mathrm{d}}{\mathrm{d}t}\varphi(\mathbf{x}) = \lambda\varphi(\mathbf{x}). \tag{10.22}$$

In Kaiser et al. [365] the Koopman eigenfunction equation was extended for control-affine nonlinear systems:

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{x} = \mathbf{f}(\mathbf{x}) + \mathbf{B}\mathbf{u}. \tag{10.23}$$

For these systems, it is possible to apply the chain rule to $\mathrm{d}\varphi(\mathbf{x})/\mathrm{d}t$, yielding

$$\frac{\mathrm{d}}{\mathrm{d}t}\varphi(\mathbf{x}) = \nabla\varphi(\mathbf{x}) \cdot (\mathbf{f}(\mathbf{x}) + \mathbf{B}\mathbf{u}) \tag{10.24a}$$

$$= \lambda\varphi(\mathbf{x}) + \nabla\varphi(\mathbf{x}) \cdot \mathbf{B}\mathbf{u}. \tag{10.24b}$$

Note that, even with actuation, the dynamics of Koopman eigenfunctions remain linear, and the effect of actuation is still additive. However, now the actuation mode $\nabla\varphi(\mathbf{x}) \cdot \mathbf{B}$ may be state-dependent. In fact, the actuation *will* be state-dependent unless the directional derivative of the eigenfunction is constant in the $\mathbf{B}$ direction. Fortunately, there are many powerful generalizations of standard Riccati-based linear control theory (e.g., LQR, Kalman filters, etc.) for systems with a *state-dependent* Riccati equation.

SINDy with Control

Although it is appealing to identify intrinsic coordinates along which nonlinear dynamics appear linear, these coordinates are challenging to discover, even for relatively simple systems. Instead, it may be beneficial to directly identify the nonlinear actuated dynamical system in (10.6) or (10.7), for use with standard model-based control. Using the sparse identification of nonlinear dynamics (SINDy) method (see Section 7.3) results in computationally efficient models that may be used in real time with model predictive control [366]. Moreover, these models may be identified from relatively small amounts of training data, compared with neural networks and other leading machine learning

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

methods, so that they may even be characterized online and in response to abrupt changes to the system dynamics.

The SINDy algorithm is readily extended to include the effects of actuation [133, 366]. In addition to collecting measurements of the state snapshots $\mathbf{x}$ in the matrix $\mathbf{X}$, actuation inputs $\mathbf{u}$ are collected in the matrix $\boldsymbol{\Upsilon}$ from (10.11) as in DMDc. Next, an augmented library of candidate right-hand side functions $\boldsymbol{\Theta}(\begin{bmatrix}\mathbf{X} & \boldsymbol{\Upsilon}\end{bmatrix})$ is constructed:

$$\boldsymbol{\Theta}(\begin{bmatrix}\mathbf{X} & \boldsymbol{\Upsilon}\end{bmatrix}) = \begin{bmatrix}\mathbf{1} & \mathbf{X} & \boldsymbol{\Upsilon} & \mathbf{X}^2 & \mathbf{X}\otimes\boldsymbol{\Upsilon} & \boldsymbol{\Upsilon}^2 & \cdots\end{bmatrix}. \tag{10.25}$$

Here, $\mathbf{X}\otimes\boldsymbol{\Upsilon}$ denotes quadratic cross-terms between the state $\mathbf{x}$ and the actuation $\mathbf{u}$, evaluated on the data.

In SINDy with control (SINDYc), the same sparse regression is used to determine the fewest active terms in the library required to describe the observed dynamics. As in DMDc, if the system is being actively controlled via feedback $\mathbf{u} = \mathbf{K}(\mathbf{x})$, then it is impossible to disambiguate from the internal dynamics and the actuation, unless an additional perturbation signal is added to the actuation to provide additional information.

Model Predictive Control Example

In this example, we will use SINDYc to identify a model of the forced Lorenz equations from data and then control this model using MPC. The basic code is the same as SINDy, except that the actuation is included as a variable when building the library $\boldsymbol{\Theta}$.

We test the SINDYc model identification on the forced Lorenz equations:

$$\dot{x} = \sigma(y - x) + g(u), \tag{10.26a}$$
$$\dot{y} = x(\rho - z) - y, \tag{10.26b}$$
$$\dot{z} = xy - \beta z. \tag{10.26c}$$

In this example, we train a model using 20 time units of controlled data, and validate it on another 20 time units, where we switch the forcing to a periodic signal $u(t) = 50\sin(10t)$. The SINDy algorithm does not capture the effect of actuation, while SINDYc correctly identifies the forced model and predicts the behavior in response to a new actuation that was not used in the training data, as shown in Fig. 10.4.

Finally, SINDYc and neural network models of Lorenz are both used to design model predictive controllers, as shown in Fig. 10.5. Both methods identify accurate models that capture the dynamics, although the SINDYc procedure requires less data, identifies models more rapidly, and is more robust to noise than the neural network model. This added efficiency and robustness are due

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

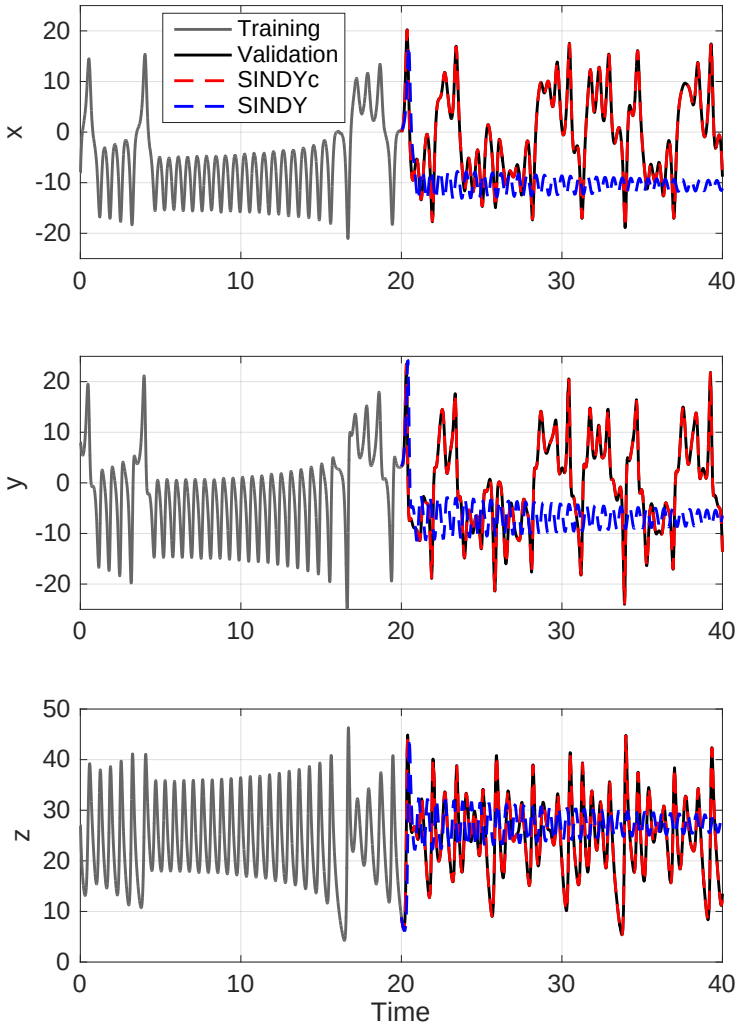


Figure 10.4: SINDy and SINDYc predictions for the controlled Lorenz system in (10.26). Training data consists of the Lorenz system with state feedback. For the training period, the input is $u(t) = 26 - x(t) + d(t)$ with a Gaussian disturbance d. Afterward the input u switches to a periodic signal $u(t) = 50\sin(10t)$. Reproduced with permission from [133].

to the sparsity-promoting optimization, which regularizes the model identification problem. In addition, identifying a sparse model requires less data.

10.3 Machine Learning Control

Machine learning is a rapidly developing field that is transforming our ability to describe complex systems from observational data, rather than first-principles

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

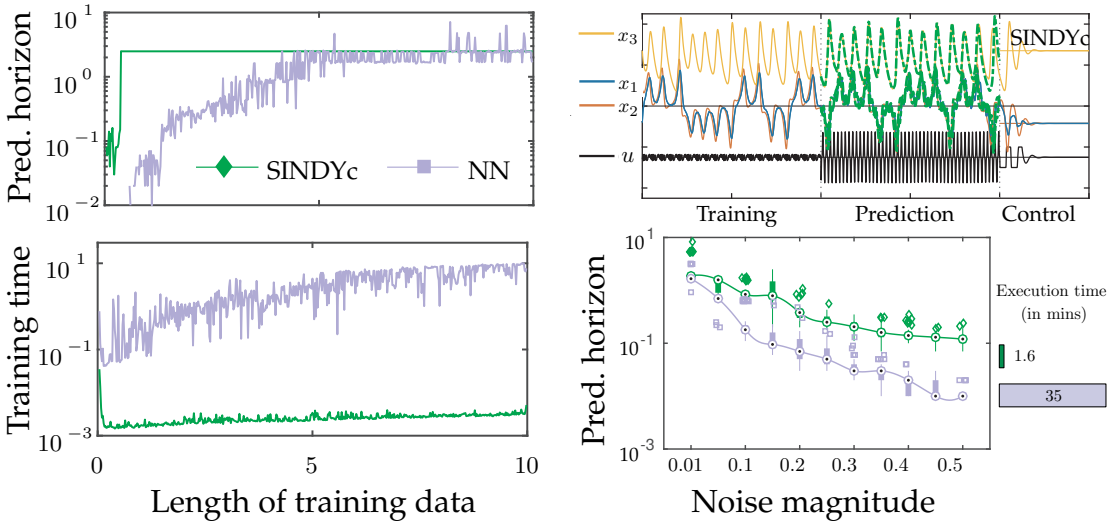


Figure 10.5: Model predictive control of the Lorenz system with a neural network model and a SINDy model. Reproduced with permission from Kaiser et al. [366].

modeling [91, 218, 504, 518]. Until recently, these methods have largely been developed for static data, although there is a growing emphasis on using machine learning to characterize dynamical systems. The use of machine learning to learn control laws (i.e., to determine an effective map from sensor outputs to actuation inputs) is even more recent [246]. As machine learning encompasses a broad range of high-dimensional, possibly nonlinear, optimization techniques, it is natural to apply machine learning to the control of complex, nonlinear systems. Specific machine learning methods for control include adaptive neural networks, genetic algorithms, genetic programming, and reinforcement learning. A general machine learning control architecture is shown in Fig. 10.6. Many of these machine learning algorithms are based on biological principles, such as neural networks, reinforcement learning, and evolutionary algorithms.

It is important to note that model-free control methodologies may be applied to numerical or experimental systems with little modification. All of these model-free methods have some sort of macroscopic objective function, typically based on sensor measurements (past and present). Some challenging real-world example objectives in different disciplines include the following.

(a) **Fluid dynamics:** In aerodynamic applications, the goal is often some combination of drag reduction, lift increase, and noise reduction; while in pharmaceutical and chemical engineering applications, the goal may involve mixing enhancement.

(b) **Finance:** The goal is often to maximize profit at a given level of risk toler-

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

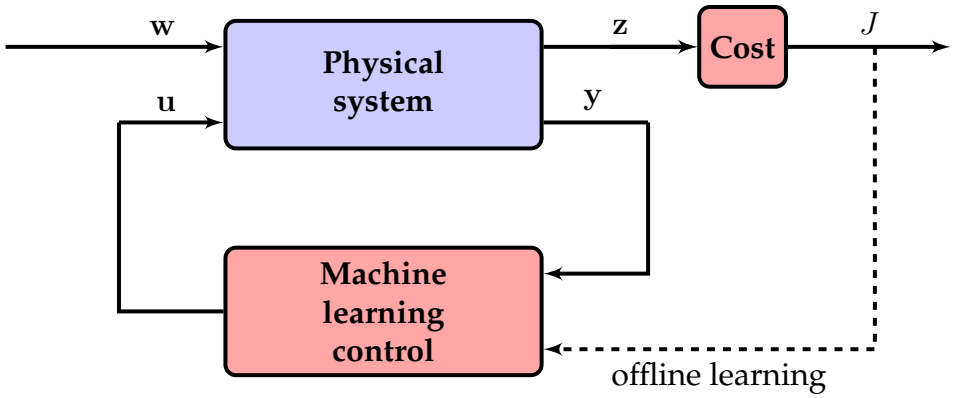


Figure 10.6: Schematic of machine learning control wrapped around a complex system using noisy sensor-based feedback. The control objective is to minimize a well-defined cost function J within the space of possible control laws. An offline learning loop provides experiential data to train the controller. Genetic programming provides a particularly flexible algorithm to search out effective control laws. The vector $\mathbf{z}$ contains information that may factor into the cost.

ance, subject to the law.

(c) **Epidemiology:** The goal may be to effectively suppress a disease with constraints of sensing (e.g., blood samples, clinics, etc.) and actuation (e.g., vaccines, bed nets, etc.).

(d) **Industry:** The goal of increasing productivity must be balanced with several constraints, including labor and work safety laws, as well as environmental impact, which often have significant uncertainty.

(e) **Autonomy and robotics:** The goal of self-driving cars and autonomous robots is to achieve a task while interacting safely with a complex environment, including cooperating with human agents.

In the examples above, the objectives involve some minimization or maximization of a given quantity subject to some constraints. These constraints may be hard, as in the case of disease suppression on a fixed budget, or they may involve a complex multi-objective tradeoff. Often, constrained optimizations will result in solutions that live at the boundary of the constraint, which may explain why many companies operate at the fringe of legality. In all of the cases, the optimization must be performed with respect to the underlying dynamics of the system: fluids are governed by the Navier–Stokes equations, finance is governed by human behavior and economics, and disease spread is the result of a complex interaction of biology, human behavior, and geography.

These real-world control problems are extremely challenging, for a number of reasons. They are high-dimensional and strongly nonlinear, often with

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

millions or billions of degrees of freedom that evolve according to possibly unknown nonlinear interactions. In addition, it may be exceedingly expensive or infeasible to run different scenarios for system identification; for example, there are serious ethical issues associated with testing different vaccination strategies when human lives are at stake.

Increasingly, challenging optimization problems are being solved with machine learning, leveraging the availability of vast and increasing quantities of data. Many of the recent successes have been on static data (e.g., image classification, speech recognition, etc.), and marketing tasks (e.g., online sales and ad placement). However, current efforts are applying machine learning to analyze and control complex systems with dynamics, with the potential to revolutionize our ability to interact with and manipulate these systems.

The following sections describe a handful of powerful learning techniques that are being widely applied to control complex systems where models may be unavailable. Note that the relative importance of the following methods are not proportional to the amount of space dedicated to them here.

Reinforcement Learning

Reinforcement learning (RL) is an important discipline at the intersection of machine learning and control [683], and it is currently being used heavily by companies for generalized artificial intelligence, autonomous robots, and self-driving cars. In reinforcement learning, a control policy is refined over time, with improved performance achieved through experience. Because this is such an important research area, it is the topic of Chapter 11.

Iterative Learning Control

Iterative learning control (ILC) [7, 94, 115, 175, 457, 511] is a widely used technique that learns how to refine and optimize repetitive control tasks, such as the motion of a robot arm on a manufacturing line, where the robot arm will be repeating the same motion thousands of times. In contrast to the feedback control methods from Chapter 8, which adjust the actuation signal in real time based on measurements, ILC refines the entire open-loop actuation sequence after each iteration of a prescribed task. The refinement process may be as simple as a proportional correction based on the measured error, or may involve a more sophisticated update rule. Iterative learning control does not require one to know the system equations and has performance guarantees for linear systems. ILC is therefore a mainstay in industrial control for repetitive tasks in a well-controlled environment, such as trajectory control of a robot arm or printer-head control in additive manufacturing.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

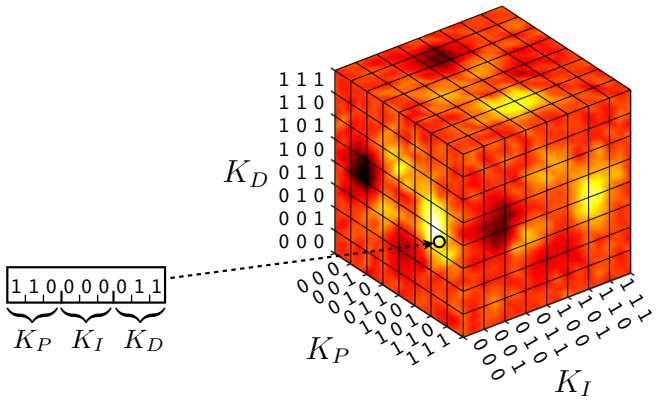


Figure 10.7: Depiction of parameter cube for PID control. The genetic algorithm represents a given parameter value as a *genetic sequence* that concatenates the various parameters. In this example, the parameters are expressed in binary representation that is scaled so that **000** is the minimum bound and **111** is the upper bound. Color indicates the cost associated with each parameter value.

Genetic Algorithms

The genetic algorithm (GA) is one of the earliest and simplest algorithms for parameter optimization, based on the biological principle of optimization through natural selection and fitness [193, 284, 333]. GA is frequently used to tune and adapt the parameters of a controller. In GA, a population comprising many system realizations with different parameter values compete to minimize a given cost function, and successful parameter values are propagated to future generations through a set of *genetic* rules. The parameters of a system are generally represented by a binary sequence, as shown in Fig. 10.7 for a PID control system with three parameters, given by the three control gains K_P, K_I, and K_D. Next, a number of realizations with different parameter values, called *individuals*, are initialized in a population and their performance is evaluated and compared on a given well-defined task. Successful individuals with a lower cost have a higher probability of being selected to advance to the next generation, according to the following genetic operations.

(a) **Elitism (optional):** A set number of the most fit individuals with the best performance are advanced directly to the next generation.

(b) **Replication:** An individual is selected to advance to the next generation.

(c) **Crossover:** Two individuals are selected to exchange a portion of their code and then advance to the next generation; crossover serves to exploit and enhance existing successful strategies.

(d) **Mutation:** An individual is selected to have a portion of its code modified

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

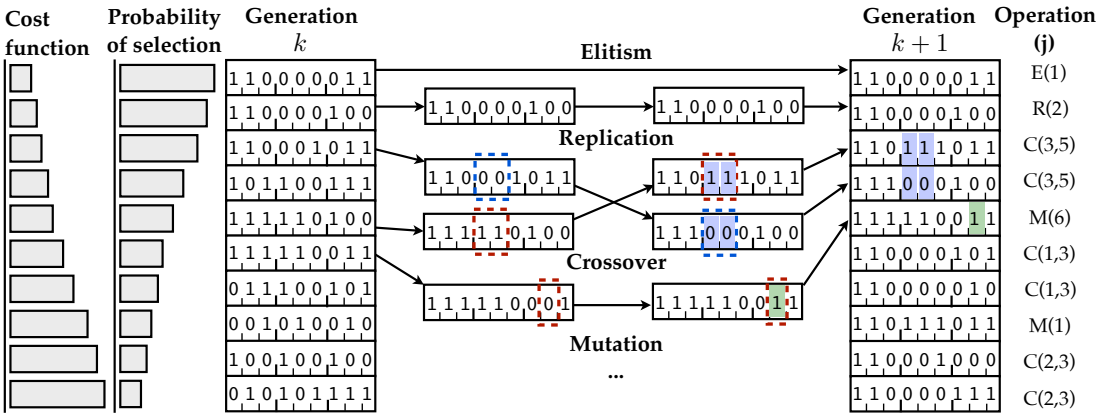


Figure 10.8: Schematic illustrating evolution in a genetic algorithm. The individuals in generation k are each evaluated and ranked in ascending order based on their cost function, which is inversely proportional to their probability of selection for genetic operations. Then, individuals are chosen based on this weighted probability for advancement to generation $k+1$ using the four operations: elitism, replication, crossover, and mutation. This forms generation $k+1$, and the sequence is repeated until the population statistics converges or another suitable stopping criterion is reached.

with new values; mutation promotes diversity and serves to increase the exploration of parameter space.

For the replication, crossover, and mutation operations, individuals are randomly selected to advance to the next generation with the probability of selection increasing with fitness. The genetic operations are illustrated for the PID control example in Fig. 10.8. These generations are evolved until the fitness of the top individuals converges or other stopping criteria are met.

Genetic algorithms are generally used to find nearly globally optimal parameter values, as they are capable of exploring and exploiting local wells in the cost function. GA provides a middle ground between a brute-force search and a convex optimization, and is an alternative to expensive Monte Carlo sampling, which does not scale to high-dimensional parameter spaces. However, there is no guarantee that genetic algorithms will converge to a globally optimal solution. There are also a number of hyperparameters that may affect performance, including the size of the populations, number of generations, and relative selection rates of the various genetic operations.

Genetic algorithms have been widely used for optimization and control in nonlinear systems [246]. For example, GA was used for parameter tuning in open-loop control [516], with applications in jet mixing [406], combustion processes [137], wake control [259, 567], and drag reduction [270]. GA has also been

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

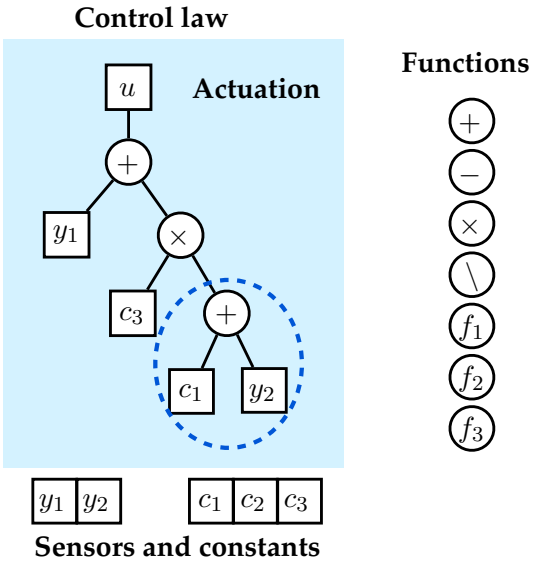


Figure 10.9: Illustration of the function tree used to represent the control law $\mathbf{u}$ in genetic programming control.

employed to tune an $\mathcal{H}_\infty$ controller in a combustion experiment [312].

Genetic Programming

Genetic programming (GP) [409, 410] is a powerful generalization of genetic algorithms that simultaneously optimizes both the structure and parameters of an input–output map. Recently, genetic programming has also been used to obtain control laws that map sensor outputs to actuation inputs, as shown in Fig. 10.9. The function tree representation in GP is quite flexible, enabling the encoding of complex functions of the sensor signal $\mathbf{y}$ through a recursive tree structure. Each branch is a signal, and the merging points are mathematical operations. Sensors and constants are the leaves, and the overall control signal u is the root. The genetic operations of crossover, mutation, and replication are shown schematically in Fig. 10.10. This framework is readily generalized to include delay coordinates and temporal filters, as discussed in Duriez et al. [225].

Genetic programming has been recently used with impressive results in turbulence control experiments, led by Bernd Noack and collaborators [226, 227, 266, 525, 546, 547]. This provides a new paradigm of control for strongly nonlinear systems, where it is now possible to identify the structure of nonlinear control laws. Genetic programming control is particularly well suited to experiments where it is possible to rapidly evaluate a given control law, en-

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

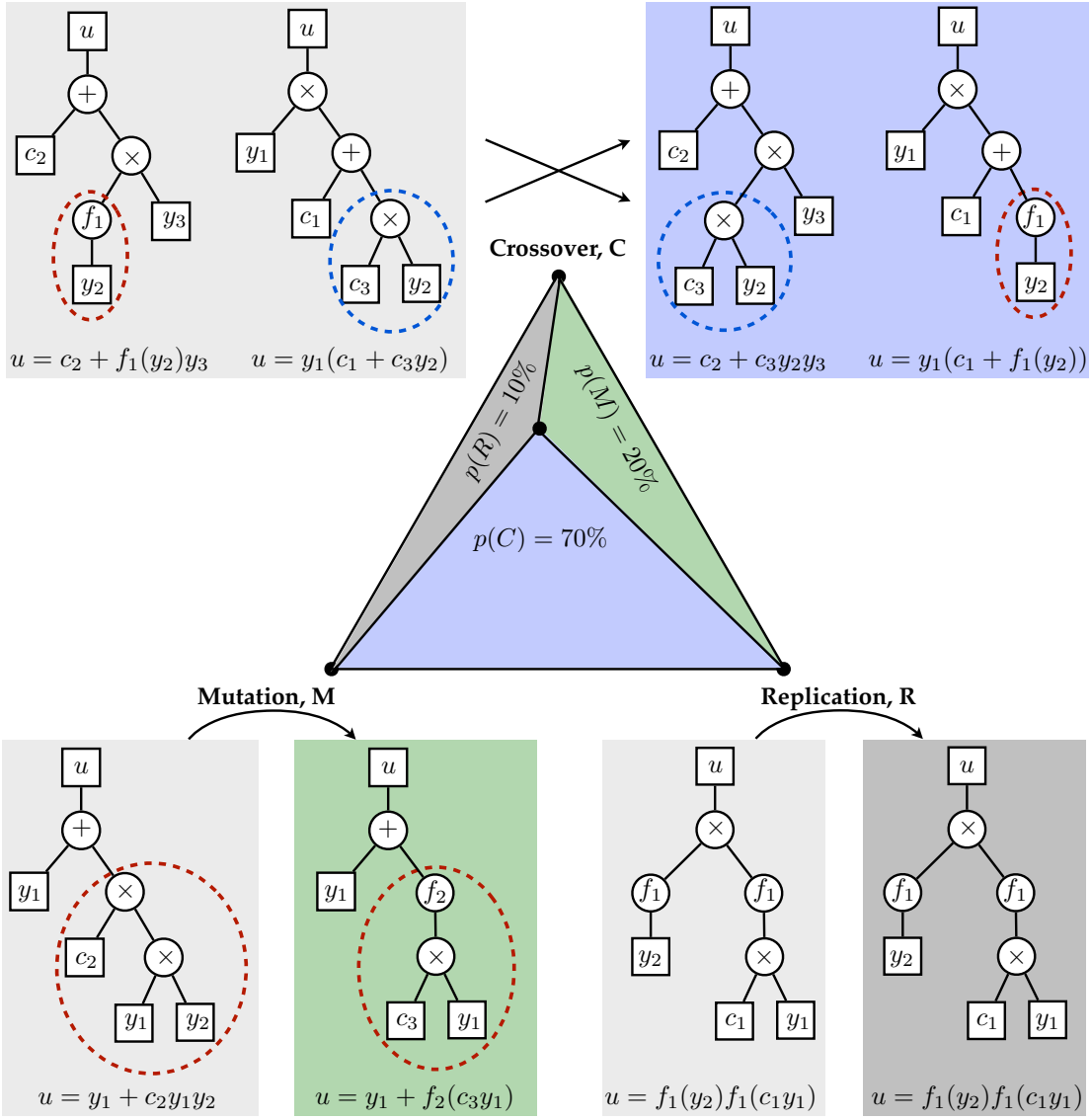


Figure 10.10: Genetic operations used to advance function trees across generations in genetic programming control. The relative selection rates of replication, crossover, and mutation are $p(R) = 0.1$, $p(C) = 0.7$, and $p(M) = 0.2$, respectively.

abling the testing of hundreds or thousands of individuals in a short amount of time. Current demonstrations of genetic programming control in turbulence have produced several macroscopic behaviors, such as drag reduction and mixing enhancement, in an array of flow configurations. Specific flows include the mixing layer [226, 227, 546, 547], the backward-facing step [227, 266**?**], and a

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

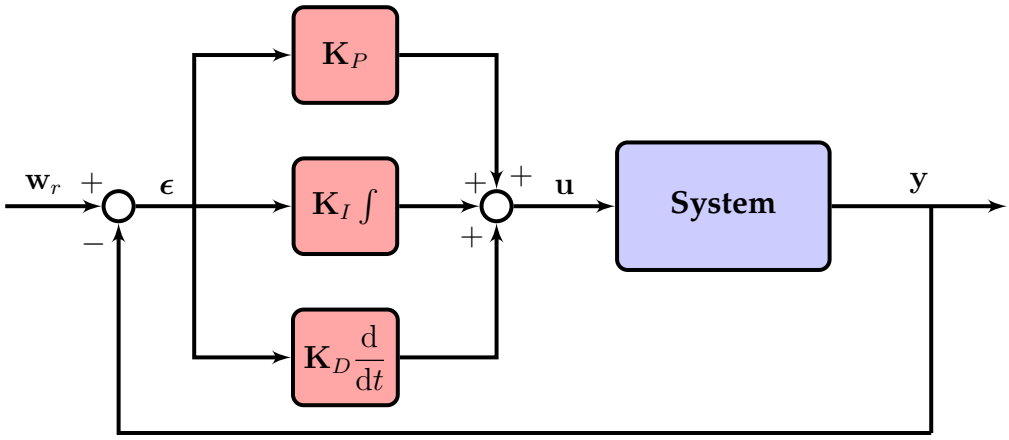


Figure 10.11: Proportional–integral–derivative (PID) control schematic. PID remains ubiquitous in industrial control.

turbulent separated boundary layer [227].

Example: Genetic Algorithm to Tune PID Control

In this example, we will use the genetic algorithm to tune a proportional–integral–derivative (PID) controller. However, it should be noted that this is just a simple demonstration of evolutionary algorithms, and such heavy machinery is not recommended to tune a PID controller in practice, as there are far simpler techniques.

PID control is among the simplest and most widely used control architectures in industrial control systems, including for motor position and velocity control, for tuning of various subsystems in an automobile, and for the pressure and temperature controls in modern espresso machines, to name only a few of the myriad applications. As its name suggests, PID control additively combines three terms to form the actuation signal, based on the error signal and its integral and derivative in time. A schematic of PID control is shown in Fig. 10.11.

In the cruise control example in Section 8.1, we saw that it was possible to reduce reference tracking error by increasing the proportional control gain K_P in the control law $u = -K_P(w_r - y)$. However, increasing the gain may eventually cause instability in some systems, and it will not completely eliminate the steady-state tracking error. The addition of an integral control term, $K_I \int_0^t (w_r - y)$ is useful to eliminate steady-state reference tracking error while alleviating the work required by the proportional term.

There are formal rules for how to choose the PID gains for various design specifications, such as fast response and minimal overshoot and ringing. In this example, we explore the use of a genetic algorithm to find effective PID gains

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

to minimize a cost function. We use an LQR cost function

$$J = \int_0^T Q(w_r - y)^2 + Ru^2 \, \mathrm{d}\tau,$$

with $Q = 1$ and $R = 0.001$ for a step response $w_r = 1$. The system to be controlled will be given by the transfer function

$$G(s) = \frac{1}{s^4 + s}.$$

The first step is to write a function that evaluates a given PID controller, as in Code 10.1. The three PID gains are stored in the variable **parms**.

Code 10.1: [MATLAB] Evaluate cost function for PID controller.

```
function J = pidtest(G,dt,parms)

s = tf('s');
K = parms(1) + parms(2)/s + parms(3)*s/(1+.001*s);
Loop = series(K,G);
ClosedLoop = feedback(Loop,1);
t = 0:dt:20;
[y,t] = step(ClosedLoop,t);

CTRLtf = K/(1+K*G);
u = lsim(K,1-y,t);

Q = 1;    R = .001;
J = dt*sum(Q*(1-y(:)).^2 + R*u(:).^2)
```

Code 10.1: [Python] Evaluate cost function for PID controller.

```
def pidtest(G,dt,parms):
    s = tf(1,1)
    K = parms[0] + parms[1]/s + parms[2]*s/(1+0.001*s)
    Loop = series(K,G)
    ClosedLoop = feedback(Loop,1)
    t = np.arange(0,20,dt)
    y,t = step(ClosedLoop,1)

    CTRLtf = K/(1+K*G)
    u = lsim(K,1-y,t)[0]

    Q = 1
    R = 0.001
    J = dt*np.sum(np.power(Q@(1-y.reshape(-1)),2) + R @ np.
       power(u.reshape(-1),2))
    return J
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

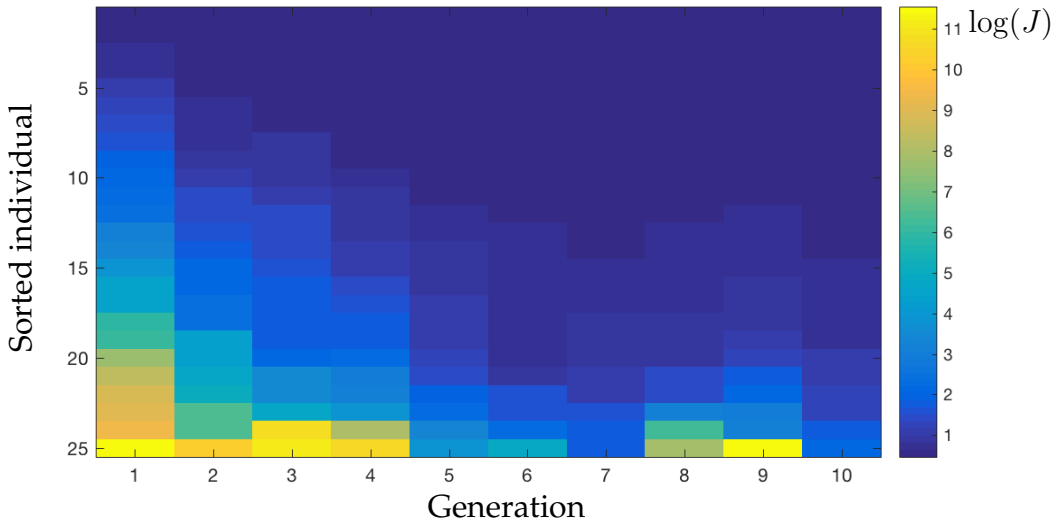


Figure 10.12: Cost function across generations, as GA optimizes PID gains.

Next, it is relatively simple to use a genetic algorithm to optimize the PID control gains, as in Code 10.2. In this example, we run the GA for 10 generations, with a population size of 25 individuals per generation.

Code 10.2: [MATLAB] Genetic algorithm to tune PID controller.

```
dt = 0.001;
PopSize = 25;
MaxGenerations = 10;
s = tf('s');
G = 1/(s*(s*s+s+1));

options = optimoptions(@ga,'PopulationSize',PopSize,'
   MaxGenerations',MaxGenerations,'OutputFcn',@myfun);
[x,fval] = ga(@(K)pidtest(G,dt,K),3,-eye(3),zeros(3,1)
   ,[],[],[],[],[],options);
```

It is also possible to reproduce this example in Python using the distributed evolutionary algorithms in Python (DEAP) package at `https://github.com/DEAP/deap`. Python code is available on the book's GitHub, although it is too long to reproduce here, as there is not a simple one-line `ga` command, as in MATLAB.

The results from intermediate generations may be saved using a custom output function, as described in the `myfun.m` code on the book's GitHub.

The evolution of the cost function across various generations is shown in Fig. 10.12. As the generations progress, the cost function steadily decreases.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

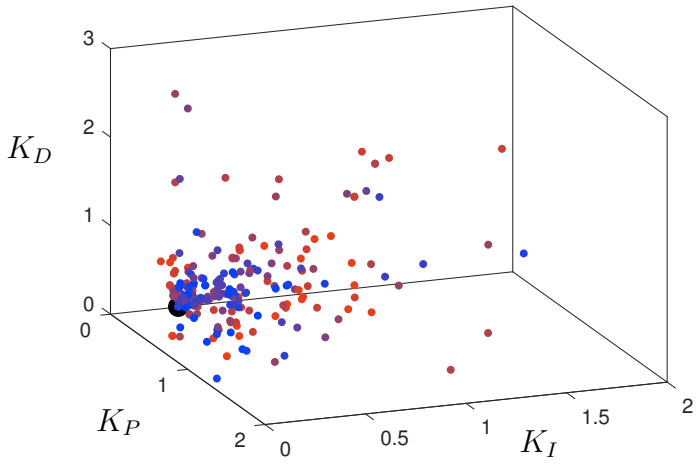


Figure 10.13: PID gains generated from genetic algorithm. Red points correspond to early generations while blue points correspond to later generations. The black point is the best individual found by GA.

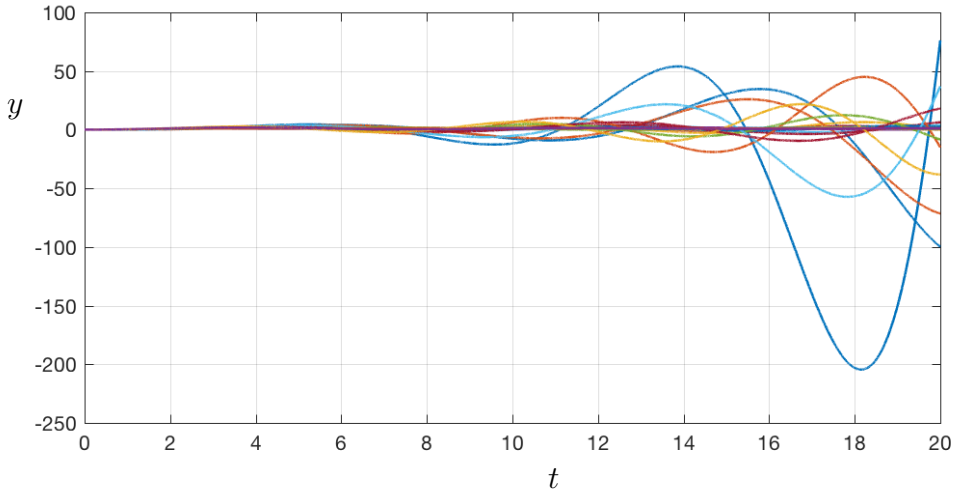


Figure 10.14: PID controller response from first generation of genetic algorithm.

The individual gains are shown in Fig. 10.13, with redder dots corresponding to early generations and bluer dots corresponding to later generations. As the genetic algorithm progresses, the PID gains begin to cluster around the optimal solution (black circle).

Figure 10.14 shows the output in response to the PID controllers from the first generation. It is clear from this plot that many of the controllers fail to

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

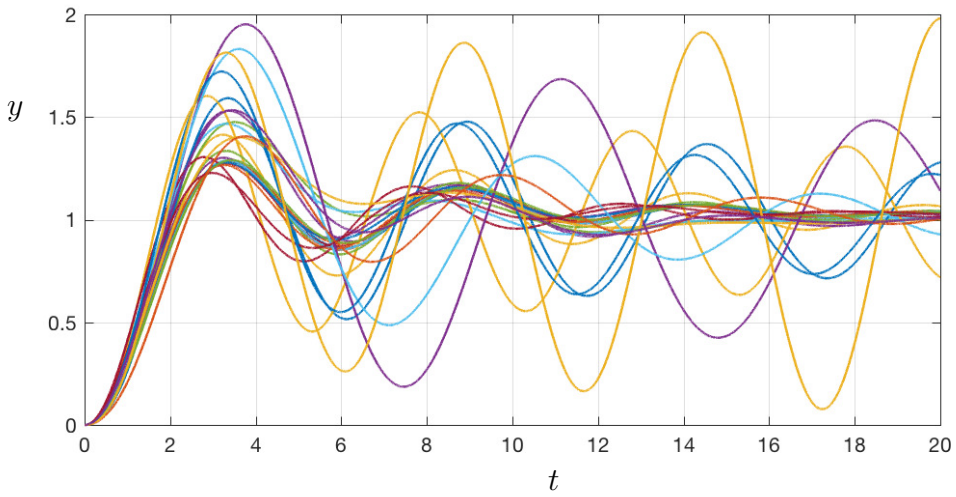


Figure 10.15: PID controller response from last generation of genetic algorithm.

stabilize the system, resulting in large deviations in y. In contrast, Fig. 10.15 shows the output in response to the PID controllers from the last generation. Overall, these controllers are more effective at producing a stable step response.

The best controllers from each generation are shown in Fig. 10.16. In this plot, the controllers from early generations are redder, while the controllers from later generations are bluer. As the GA progresses, the controller is able to minimize output oscillations and achieves fast rise-time.

10.4 Adaptive Extremum-Seeking Control

Although there are many powerful techniques for model-based control design, there are also a number of drawbacks. First, in many systems, there may not be access to a model, or the model may not be suitable for control (i.e., there may be strong nonlinearities or the model may be represented in a non-traditional form). Next, even after an attractor has been identified and the dynamics characterized, control may invalidate this model by modifying the attractor, giving rise to new and uncharacterized dynamics. The obvious exception is stabilizing a fixed point or a periodic orbit, in which case effective control keeps the system in a neighborhood where the linearized model remains accurate. Finally, there may be slow changes to the system that modify the underlying dynamics, and it may be difficult to measure and model these effects.

The field of *adaptive* control broadly addresses these challenges, by allowing the control law the flexibility to modify its action based on the changing dynamics of a system. Extremum-seeking control (ESC) [26, 415] is a particu-

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

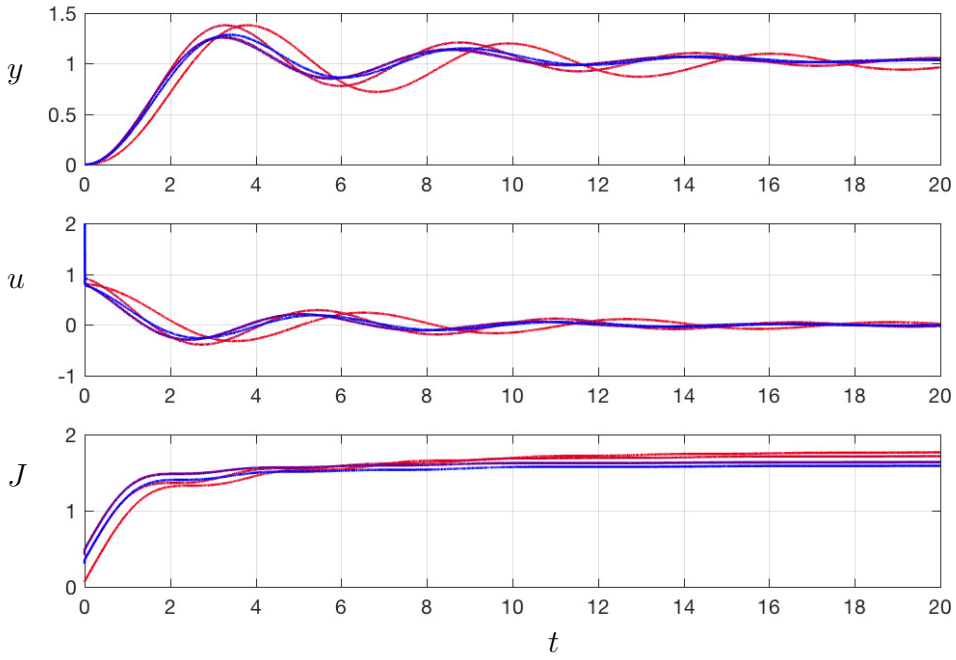


Figure 10.16: Best PID controllers from each generation. Red trajectories are from early generations, and blue trajectories correspond to the last generation.

larly attractive form of adaptive control for complex systems because it does not rely on an underlying model and it has guaranteed convergence and stability under a set of well-defined conditions. Extremum-seeking may be used to track local maxima of an objective function, despite disturbances, varying system parameters, and nonlinearities. Adaptive control may be implemented for in-time control or used for slow tuning of parameters in a working controller.

Extremum-seeking control may be thought of as an advanced *perturb-and-observe* method, whereby a sinusoidal perturbation is additively injected in the actuation signal and used to estimate the gradient of an objective function J that should be maximized or minimized. The objective function is generally computed based on sensor measurements of the system, although it ultimately depends on the internal dynamics and the choice of the input signal. In extremum-seeking, the control variable $\mathbf{u}$ may refer either to the actuation signal or to a set of parameters that describe the control behavior, such as the frequency of periodic forcing or the gains in a PID controller.

The extremum-seeking control architecture is shown in Fig. 10.17. This schematic depicts ESC for a scalar input u, although the methods readily generalize for vector-valued inputs $\mathbf{u}$. A convex objective function $J(u)$ is shown in Fig. 10.18 for static plant dynamics (i.e., for $y = u$). The extremum-seeking controller uses

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

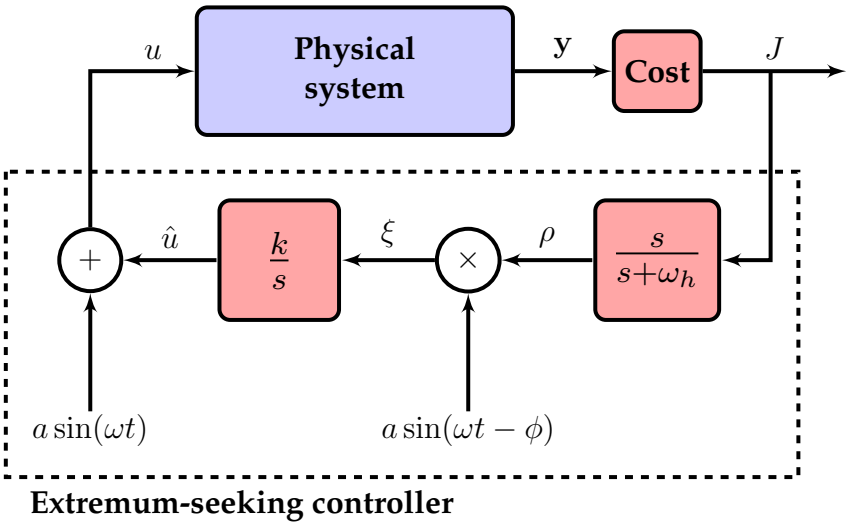


Figure 10.17: Schematic illustrating an extremum-seeking controller. A sinusoidal perturbation is added to the best guess of the input $\hat{u}$, and it passes through the plant, resulting in a sinusoidal output perturbation that may be observed in the sensor signal y and the cost J. The high-pass filter results in a zero-mean output perturbation, which is then multiplied (demodulated) by the same input perturbation, resulting in the signal ξ. This demodulated signal is finally integrated into the best guess $\hat{u}$ for the optimizing input u.

an input perturbation to estimate the gradient of the objective function J and steer the mean actuation signal towards the optimizing value.

Three distinct timescales are relevant for extremum-seeking control:

(a) slow – external disturbances and parameter variation;

(b) medium – perturbation frequency ω;

(c) fast – system dynamics.

In many systems, the internal system dynamics evolve on a fast timescale. For example, turbulent fluctuations may equilibrate rapidly compared to actuation timescales. In optical systems, such as a fiber laser [129], the dynamics of light inside the fiber are extremely fast compared to the timescales of actuation.

In extremum-seeking control, a sinusoidal perturbation is added to the estimate of the input that maximizes the objective function, $\hat{u}$:

$$u = \hat{u} + a\sin(\omega t). \tag{10.27}$$

This input perturbation passes through the system dynamics and output, resulting in an objective function J that varies sinusoidally about some mean

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

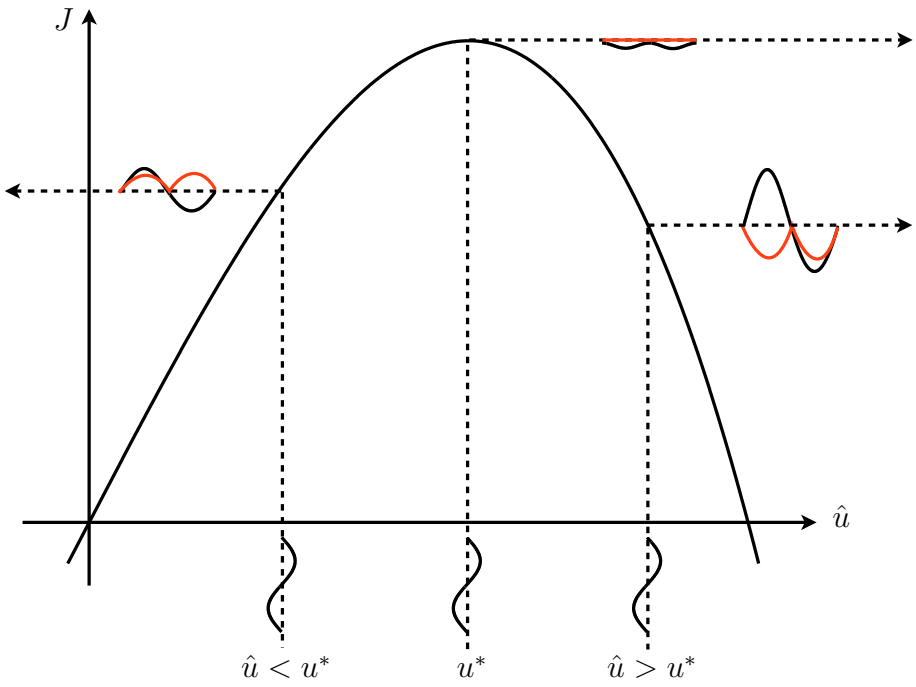


Figure 10.18: Schematic illustrating extremum-seeking control on a static objective function $J(u)$. The output perturbation (red) is in phase when the input is left of the peak value (i.e., $u < u^*$) and out of phase when the input is to the right of the peak (i.e., $u > u^*$). Thus, integrating the product of input and output sinusoids moves $\hat{u}$ towards u^*.

value, as shown in Fig. 10.18. The output J is high-pass-filtered to remove the mean (DC component), resulting in the oscillatory signal ρ. A simple high-pass filter is represented in the frequency domain as

$$\frac{s}{s+\omega_h}, \tag{10.28}$$

where s is the Laplace variable and ω_h is the filter frequency. The high-pass filter is chosen to pass the perturbation frequency ω. The high-pass-filtered output is then multiplied by the input sinusoid, possibly with a phase shift ϕ, resulting in the *demodulated* signal ξ:

$$\xi = a\sin(\omega t - \phi)\rho. \tag{10.29}$$

This signal ξ is mostly positive if the input u is to the left of the optimal value u^* and it is mostly negative if u is to the right of the optimal value u^*, shown as red curves in Fig. 10.18. Thus, the demodulated signal ξ is integrated into $\hat{u}$, the best estimate of the optimizing value,

$$\frac{\mathrm{d}}{\mathrm{d}t}\hat{u} = k\xi, \tag{10.30}$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

so that the system estimate $\hat{u}$ is steered towards the optimal input u^*. Here, k is an integral gain, which determines how aggressively the actuation climbs gradients in J.

Roughly speaking, the demodulated signal ξ measures gradients in the objective function, so that the algorithm climbs to the optimum more rapidly when the gradient is larger. This is simple to see for constant plant dynamics, where J is simply a function of the input, $J(u) = J(\hat{u} + a\sin(\omega t))$. Expanding $J(u)$ in the perturbation amplitude a, which is assumed to be small, yields

$$J(u) = J(\hat{u} + a\sin(\omega t)) \tag{10.31a}$$

$$= J(\hat{u}) + \left.\frac{\partial J}{\partial u}\right|_{u=\hat{u}} a\sin(\omega t) + \mathcal{O}(a^2). \tag{10.31b}$$

The leading-order term in the high-pass-filtered signal is $\rho \approx \partial J/\partial u|_{u=\hat{u}}\ a\sin(\omega t)$. Averaging $\xi = a\sin(\omega t - \phi)\rho$ over one period yields

$$\xi_{\text{avg}} = \frac{\omega}{2\pi}\int_0^{2\pi/\omega} a\sin(\omega t - \phi)\rho\,\mathrm{d}t \tag{10.32a}$$

$$= \frac{\omega}{2\pi}\int_0^{2\pi/\omega} \left.\frac{\partial J}{\partial u}\right|_{u=\hat{u}} a^2\sin(\omega t - \phi)\sin(\omega t)\,\mathrm{d}t \tag{10.32b}$$

$$= \frac{a^2}{2}\left.\frac{\partial J}{\partial u}\right|_{u=\hat{u}} \cos(\phi). \tag{10.32c}$$

Thus, for the case of trivial plant dynamics, the average signal ξ_{avg} is proportional to the gradient of the objective function J with respect to the input u.

In general, extremum-seeking control may be applied to systems with nonlinear dynamics relating the input u to the outputs $\mathbf{y}$ that act on a faster timescale than the perturbation ω. Thus, J may be time-varying, which complicates the simplistic averaging analysis above. The general case of extremum-seeking control of nonlinear systems is analyzed by Krstić and Wang [415], where they develop powerful stability guarantees based on a separation of timescales and a singular perturbation analysis. The basic algorithm may also be modified to add a phase ϕ to the sinusoidal input perturbation in (10.29). In [415], there was an additional low-pass filter $\omega_l/(s+\omega_l)$ placed before the integrator to extract the DC component of the demodulated signal ξ. There is also an extension to extremum-seeking called slope-seeking, where a specific slope is sought [26] instead of the standard zero slope corresponding to a maximum or minimum. Slope-seeking is preferred when there is not an extremum, as in the case when control inputs saturate. Extremum-seeking is often used for frequency selection and slope-seeking is used for amplitude selection when tuning an open-loop periodic forcing.

It is important to note that extremum-seeking control will only find local maxima of the objective function, and there are no guarantees that this will correspond to a global maximum. Thus, it is important to start with a good initial

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

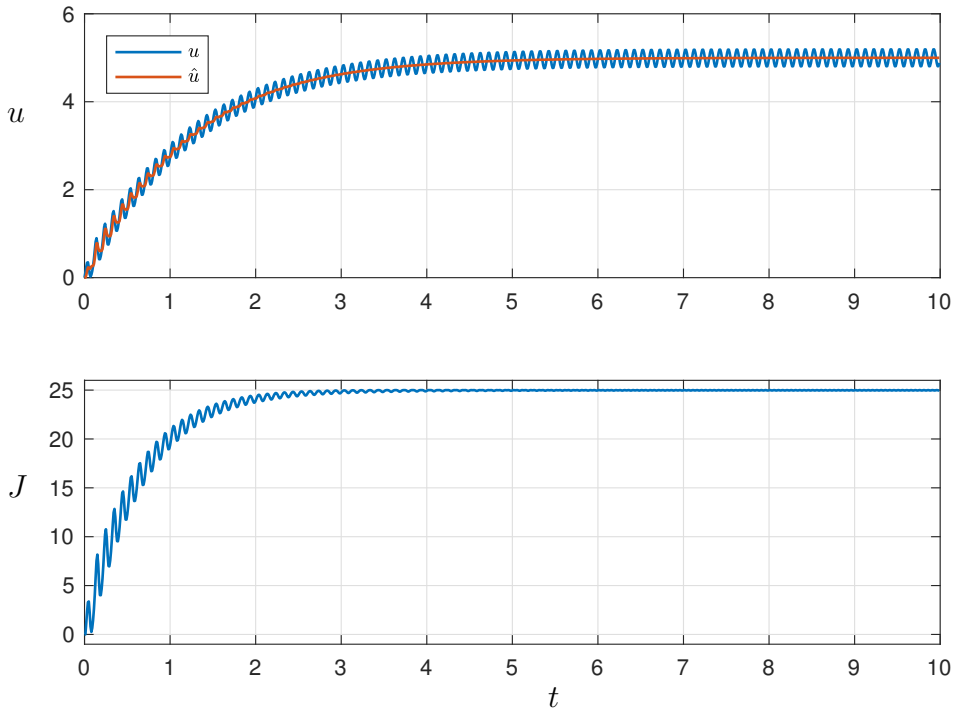


Figure 10.19: Extremum-seeking control response for cost function in (10.33).

condition for the optimization. In a number of studies, extremum-seeking control is used in conjunction with other global optimization techniques, such as a genetic algorithm, or sparse representation for classification [130, 256].

Simple Example of Extremum-Seeking Control

Here we consider a simple application of extremum-seeking control to find the maximum of a static quadratic cost function,

$$J(u) = 25 - (5 - u)^2. \tag{10.33}$$

This function has a single global maximum at $u^* = 5$. Starting at $u = 0$, we apply extremum-seeking control with a perturbation frequency of $\omega = 10\,\text{Hz}$ and an amplitude of $a = 0.2$. Figure 10.19 shows the controller response and the rapid tracking of the optimal value $u^* = 5$. MATLAB and Python codes that implement extremum-seeking using a simple Butterworth high-pass filter are provided on the book's GitHub.

Notice that when the gradient of the cost function is larger (i.e., closer to $u = 0$), the oscillations in J are larger, and the controller climbs more rapidly.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

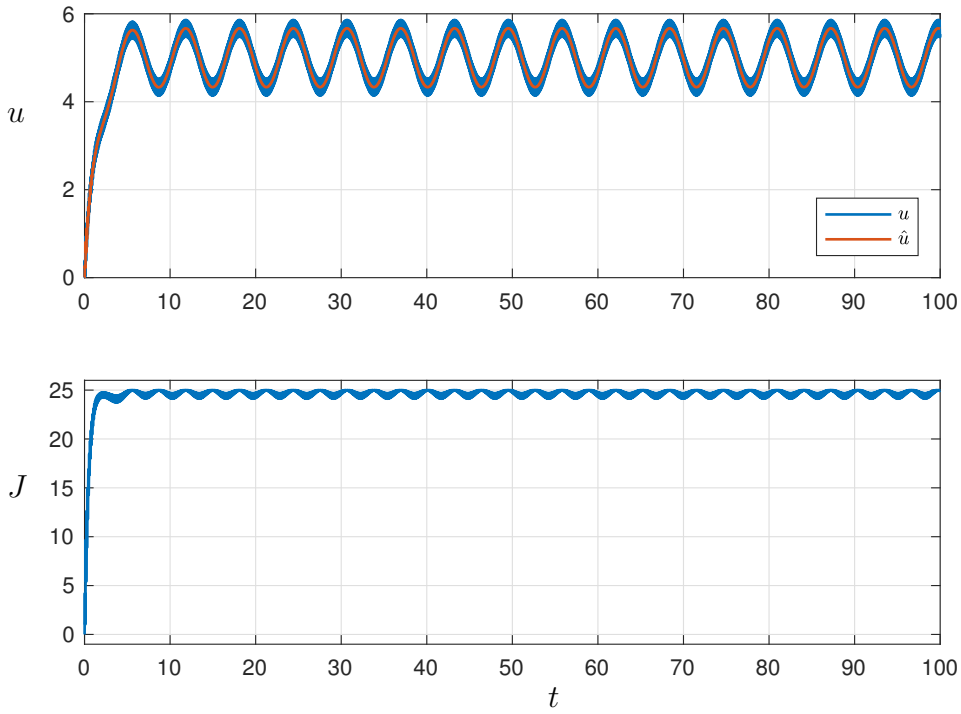


Figure 10.20: Extremum-seeking control response with a slowly changing cost function $J(u,t)$.

When the input u gets close to the optimum value at $u^* = 5$, even though the input perturbation has the same amplitude a, the output perturbation is nearly zero (on the order of a^2), since the quadratic cost function is flat near the peak. Thus we achieve fast tracking far away from the optimum value and small deviations near the peak.

To see the ability of extremum-seeking control to handle varying system parameters, consider the time-dependent cost function given by

$$J(u) = 25 - (5 - u - \sin(t))^2. \tag{10.34}$$

The varying parameters, which oscillate at $1/(2\pi)$ Hz, may be considered slow compared with the perturbation frequency 10 Hz. The response of extremum-seeking control for this slowly varying system is shown in Fig. 10.20. In this response, the actuation signal is able to maintain good performance by oscillating back and forth to approximately track the oscillating optimal u^*, which oscillates between 4 and 6. The output function J remains close to the optimal value of 25, despite the unknown varying parameter.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

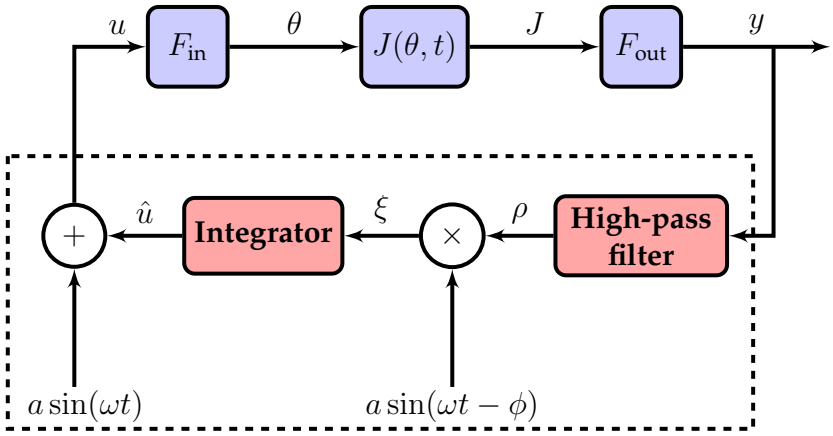


Figure 10.21: Schematic of a specific extremum-seeking control architecture that benefits from a wealth of design techniques [26, 178].

Challenging Example of Extremum-Seeking Control

Here we consider an example inspired by a challenging benchmark problem in [26, Section 1.3]. This system has a time-varying objective function $J(t)$ and dynamics with a right half-plane zero, making it difficult to control.

In one formulation of extremum-seeking [26, 178], there are additional guidelines for designing the controller if the plant can be split into three blocks that define the input dynamics, a time-varying objective function with no internal dynamics, and the output dynamics, as shown in Fig. 10.21. In this case, there are procedures to design the high-pass filter and integrator blocks.

In this example, the objective function is given by

$$J(\theta) = 0.05\delta(t - 10) + (\theta - \theta^*(t))^2,$$

where δ is the Dirac delta function, and the optimal value $\theta^*(t)$ is given by

$$\theta^* = 0.01 + 0.001t.$$

The optimal objective is given by $J^* = 0.05\delta(t - 10)$. The input and output dynamics are taken from the example in [26], and are given by

$$F_{\text{in}}(s) = \frac{s-1}{(s+2)(s+1)} \quad \text{and} \quad F_{\text{out}}(s) = \frac{1}{s+1}.$$

Using the design procedure in [26], one arrives at the high-pass filter $s/(s + 5)$ and an integrator-like block given by $50(s - 4)/(s - 0.01)$. In addition, a perturbation with $\omega = 5$ and $a = 0.05$ is used, and the demodulating perturbation is phase-shifted by $\phi = 0.7955$; this phase is obtained by evaluating the input function F_{in} at $i\omega$. The response of this controller is shown in Fig. 10.22, along with the Simulink implementation in Fig. 10.23. The controller is able to accurately track the optimizing input, despite additive sensor noise.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

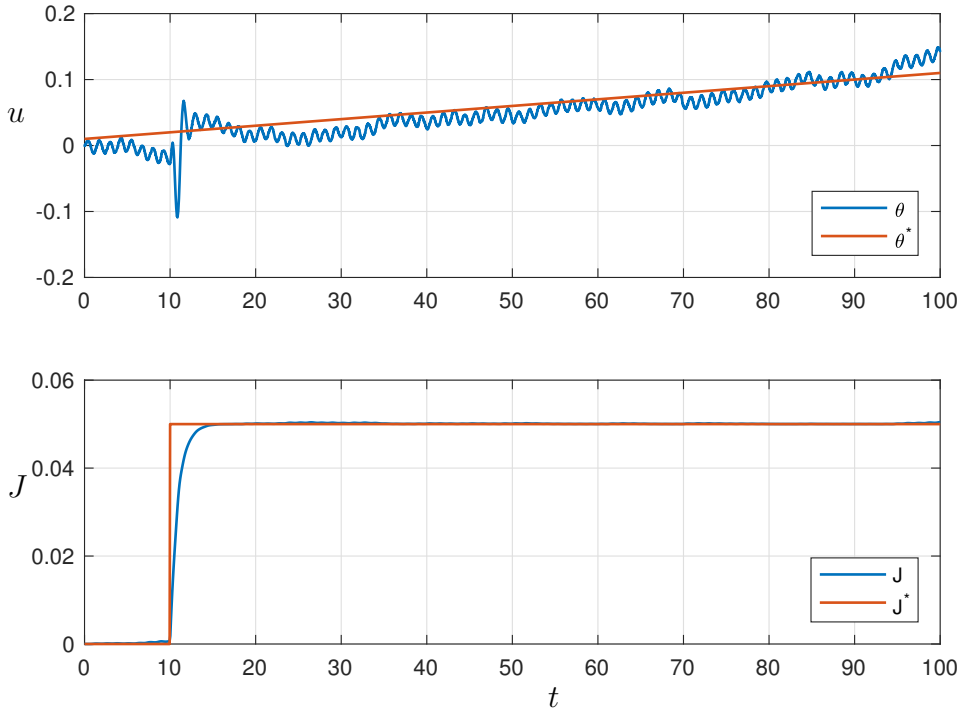


Figure 10.22: Extremum-seeking control response for a challenging test system with a right half-plane zero, inspired by [26].

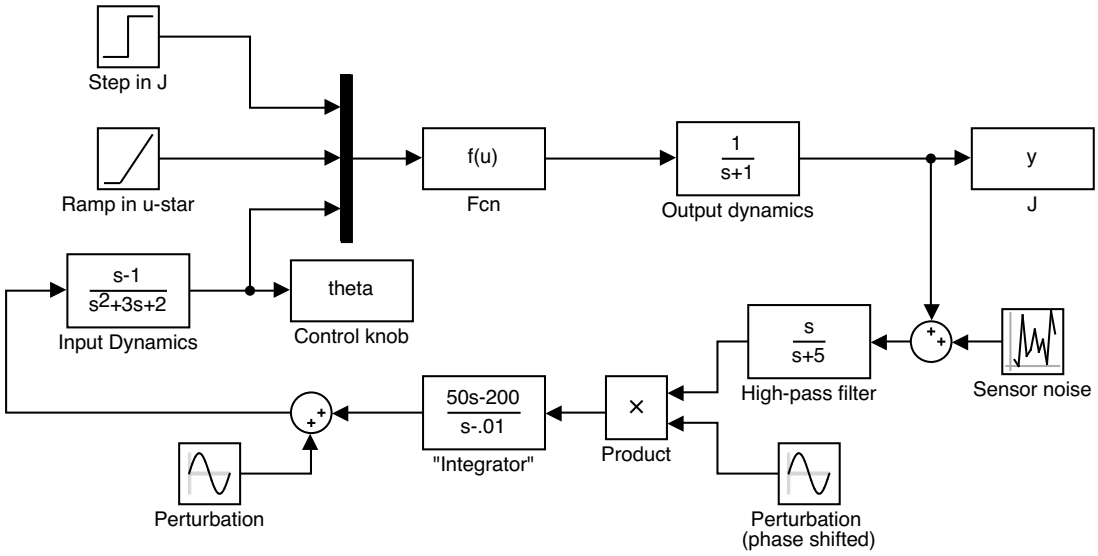


Figure 10.23: Simulink model for extremum-seeking controller used in Fig. 10.22.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Applications of Extremum-Seeking Control

Because of the lack of assumptions and ease of implementation, extremum-seeking control has been widely applied to a number of complex systems. Although ESC is generally applicable for in-time control of dynamical systems, it is also widely used as an online optimization algorithm that can adapt to slow changes and disturbances. Among the many uses of extremum-seeking control, here we highlight only a few.

Extremum-seeking has been used widely for maximum power point tracking algorithms in photovoltaics [106, 125, 237, 439], and wind energy conversion [517]. In the case of photovoltaics, the voltage or current ripple in power converters due to pulse-width modulation is used for the perturbation signal; and in the case of wind, turbulence is used as the perturbation. Atmospheric turbulent fluctuations were also used as the perturbation signal for the optimization of aircraft control [412]; in this example it is infeasible to add a perturbation signal to the aircraft control surfaces, and a natural perturbation is required. ESC has also been used in optics and electronics for laser pulse shaping [599], for tuning high-gain fiber lasers [129, 130], and for beam control in a reconfigurable holographic metamaterial antenna array [349]. Other applications include formation flight optimization [87], bioreactors [741], PID [384] and PI [416] tuning, active braking systems [771], and control of Tokamaks [541].

Extremum-seeking has also been broadly applied in turbulent flow control. Despite the ability to control dynamics in-time with ESC, it is often used as a slow feedback optimization to tune the parameters of a working open-loop controller. This slow feedback has many benefits, such as maintaining performance despite slow changes to environmental conditions. Extremum-seeking has been used to control an axial flow compressor [742], to reduce drag over a bluff body in an experiment [61, 62] using a rotating cylinder on the upper trailing edge of the rear surface, and for separation control in a high-lift airfoil configuration [63] using pressure sensors and pulsed jets on the leading edge of a single-slotted flap. There have also been impressive industrial-scale uses of extremum-seeking control, for example to control thermoacoustic modes across a range of frequencies in a 4 MW gas turbine combustor [48, 50]. It has also been utilized for separation control in a planar diffusor that is fully turbulent and stalled [49], and to control jet noise [493].

There are numerous extensions to extremum-seeking that improve performance. For example, extended Kalman filters were used as the filters in [272] to control thermoacoustic instabilities in a combustor experiment, reducing pressure fluctuations by nearly 40 dB. Kalman filters were also used with ESC to reduce the flow separation and increase the pressure ratio in a high-pressure axial fan using an injected pulsed air stream [752]. Including the Kalman filter improved the controller bandwidth by a factor of 10 over traditional ESC.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Suggested Reading

Texts

(1) Real-time optimization by extremum-seeking control, by K. B. Ariyur and M. Krstić, 2003 [26].

(2) Machine learning control: Taming nonlinear dynamics and turbulence, by T. Duriez, S. L. Brunton, and B. R. Noack, 2016 [225].

(3) Model predictive control, by E. F. Camacho and C. B. Alba, 2013 [149].

Papers and reviews

(1) Stability of extremum seeking feedback for general nonlinear dynamic systems, by M. Krstić and H. H. Wang, *Automatica*, 2000 [415].

(2) Dynamic mode decomposition with control, by J. L. Proctor, S. L. Brunton, and J. N. Kutz, *SIAM Journal on Applied Dynamical Systems*, 2016 [570].

(3) Model predictive control: Theory and practice – a survey, by C. E. Garcia, D. M. Prett, and M. Morari, *Automatica*, 1989 [262].

(4) Closed-loop turbulence control: Progress and challenges, by S. L. Brunton and B. R. Noack, *Applied Mechanics Reviews*, 2015 [124].

(5) Sparse identification of nonlinear dynamics for model predictive control in the low-data limit, by E. Kaiser, J. N. Kutz, and S. L. Brunton, *Proceedings of the Royal Society of London A*, 2018 [366].

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Homework

Exercise 10-1. In this exercise, we will compare DMDc, SINDYc, and a neural network (NN) for use with MPC to control the Lorenz system, following Kaiser et al. [366].

First, generate training data by simulating the forced Lorenz system:

$$\begin{aligned}\dot{x} &= 10(y-x)+u,\\ \dot{y} &= x(28-z)-y,\\ \dot{z} &= xy-(8/3)z.\end{aligned}$$

Generate data using a small time-step $\Delta t = 0.001$ from $t = 0$ to $t = 10$ with a rich control input $u(t) = (2\sin(t) + \sin(0.1t))^2$. Use this data to train a DMDc, SINDYc, and NN model. For the NN model, start with a single hidden layer with 10 neurons using hyperbolic tangent sigmoid activation functions. Compare the performance of all of these models to predict the response to a new forcing $u(t) = (5\sin(30t))^3$ for $t = 10$ to $t = 20$.

Finally, design an MPC controller based on each of these models to stabilize the fixed point $(x, y, z) = (-\sqrt{72}, -\sqrt{72}, 27)$.

(Bonus) Compare the prediction and control performance of the various models as a function of the amount of data used in the training phase.

Exercise 10-2. (Advanced) This exercise will develop a model predictive controller for the fluid flow past a cylinder. There are several open-source codes that can be used to simulate simple fluid flows, such as the IBPM code at `https://github.com/cwrowley/ibpm/`.

(a) First, generate a training data set that simulates the vortex shedding behind a stationary cylinder at a Reynolds number of 100. Compute the mean flow field during the periodic vortex shedding portion, after initial transients have died out. In addition, generate training data corresponding to the cylinder rotating, which will be our control input. Since the absolute angle of the cylinder is irrelevant due to symmetry, we will consider the angular rate of the cylinder as the control input. For all data, subtract the mean flow computed above.

(b) Train a DMDc model based on this data, and test the performance of this model on a test data set of the cylinder rotating. Plot the various responses and discuss the performance.

(c) Use this DMDc model for MPC with the goal of stabilizing a symmetric configuration, where the DMDc state is equal to zero. Plot the performance and discuss the results.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

(d) Now, instead of using the full flow field to characterize a DMDc model, use the lift and drag coefficients, C_L and C_D; you may need to build an augmented state vector, using either these coefficients and their time derivatives or time-delayed values. Similarly, test the performance of this model on the test data and discuss.

(e) Use this force-based DMDc model to develop an MPC that tracks a given reference lift value, say $C_L = 1$ or $C_L = -1$. See if you can make your controller track a reference that switches between these values. What if the reference lift is much larger, say $C_L = 2$ or $C_L = 5$?

(f) Use this model to develop an MPC that tracks a reference drag value, such as $C_D = 0$. Is it possible to simultaneously track a reference lift and drag value? Why or why not?

Exercise 10-3. (Advanced) Repeat the exercise above, but instead of using a DMDc model, construct a neural network model for a deep MPC. This exercise follows the work of Morton et al. [513] and Bieker et al. [84].

Compare the results from the NN-based MPC and the DMDc-based MPC.

Exercise 10-4. Design an optimal full-state LQR controller using a genetic algorithm for the spring–mass–damper system:

$$\ddot{x} + 5\dot{x} + 4x = u(t).$$

Plot the closed-loop eigenvalues and the LQR cost J as a function of the generation. Note that you will need to specify the gain matrices $\mathbf{Q}$ and $\mathbf{R}$ to define the cost function J.

Exercise 10-5. Design an extremum-seeking controller to find the peak of a quartic cost function $J(u) = 16 - (2-u)^4$, similar to the example with quadratic cost in (10.33). Do you need to modify the extremum-seeking control parameters? Discuss the performance.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Chapter 11

Reinforcement Learning

Reinforcement learning (RL) is a major branch of machine learning that is concerned with how to learn control laws and policies to interact with a complex environment from experience [362, 683]. Thus, RL is situated at the growing intersection of control theory and machine learning [589], and it is among the most promising fields of research towards generalized artificial intelligence and autonomy. Both machine learning and control theory fundamentally rely on optimization, and, likewise, RL involves a set of optimization techniques within an experiential framework for learning how to interact with the environment.

In reinforcement learning, an *agent*[1] senses the state of its environment and learns to take appropriate actions to optimize future rewards. The ultimate goal in RL is to learn an effective control strategy or set of actions through positive or negative reinforcement. This search may involve trial-and-error learning, model-based optimization, or a combination of both. In this way, reinforcement learning is fundamentally biologically inspired, mimicking how animals learn to interact with their environment through positive and negative reward feedback from trial-and-error experience. Much of the history of reinforcement learning, and machine learning more broadly, has been linked to studies of animal behavior and the neurological basis of decisions, control, and learning [194, 196, 508, 645]. For example, Pavlov's dog is an illustration that animals learn to associate environmental cues with a food reward [551]. The term *reinforcement* refers to the rewards, such as food, used to reinforce desirable actions in humans and animals. However, in animal systems, reinforcement is ultimately achieved through cellular and molecular learning rules.

Multiple textbooks have been written on this topic, which spans almost a century of progress. Major advances in deep reinforcement learning are also rapidly changing the landscape. This chapter is not meant to be comprehensive; rather, it aims to provide a solid foundation, to introduce key concepts and leading approaches, and to lower the barrier to entry in this exciting field.

[1] Ironically, from the perspective of reinforcement learning, in *The Matrix*, Neo is actually the agent learning to interact with his environment.

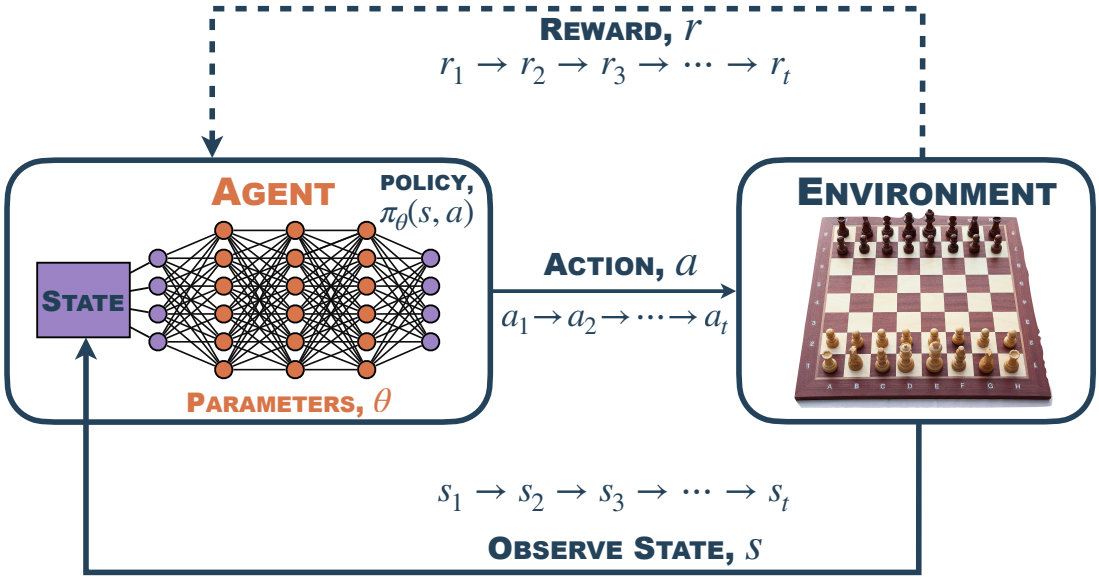


Figure 11.1: Schematic of reinforcement learning, where an agent senses its environmental state $\mathbf{s}$ and takes actions $\mathbf{a}$ according to a policy $\boldsymbol{\pi}$ that is optimized through learning to maximize future rewards $\mathbf{r}$. In this case, a deep neural network is used to represent the policy $\boldsymbol{\pi}$. This is known as a *deep policy network*.

11.1 Overview and Mathematical Formulation

Figure 11.1 provides a schematic overview of the reinforcement learning framework. An RL agent senses the state of its environment and learns to take appropriate actions to achieve optimal immediate or delayed rewards. Specifically, the RL agent arrives at a sequence of different states $\mathbf{s}_k \in \mathcal{S}$ by performing actions $\mathbf{a}_k \in \mathcal{A}$, with the selected actions leading to positive or negative rewards $\mathbf{r}_k$ used for learning. The sets $\mathcal{S}$ and $\mathcal{A}$ denote the sets of possible states and actions, respectively. Importantly, the RL agent is capable of learning from delayed rewards, which is critical for systems where the optimal solution involves a multi-step procedure. Rewards may be thought of as sporadic and time-delayed labels, leading to RL being considered a third major branch of machine learning, called *semi-supervised* learning, which complements the other two branches of supervised and unsupervised learning. One canonical example is learning a set of moves, or a long-term strategy, to win a game of chess. As is the case with human learning, RL often begins with an unstructured *exploration*, where trial and error are used to learn the rules, followed by *exploitation*, where a strategy is chosen and optimized within the learned rules.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

The Policy

An RL agent senses the state of its environment $\mathbf{s}$ and takes actions $\mathbf{a}$ through a policy $\boldsymbol{\pi}$ that is optimized through learning to maximize future rewards $\mathbf{r}$. Reinforcement learning is often formulated as an optimization problem to learn the policy $\boldsymbol{\pi}(\mathbf{s}, \mathbf{a})$,

$$\boldsymbol{\pi}(\mathbf{s}, \mathbf{a}) = \Pr(\mathbf{a} = \mathbf{a} \mid \mathbf{s} = \mathbf{s}), \tag{11.1}$$

which is the probability of taking action $\mathbf{a}$, given state $\mathbf{s}$, to maximize the total future rewards. In the simplest formulation, the policy may be a look-up table that is defined on the discrete state and action spaces $\mathcal{S}$ and $\mathcal{A}$, respectively. However, for most problems, representing and learning this policy becomes prohibitively expensive, and $\boldsymbol{\pi}$ must instead be represented as an approximate function that is parameterized by a lower-dimensional vector $\boldsymbol{\theta}$:

$$\boldsymbol{\pi}(\mathbf{s}, \mathbf{a}) \approx \boldsymbol{\pi}(\mathbf{s}, \mathbf{a}, \boldsymbol{\theta}). \tag{11.2}$$

Often, this parameterized function will be denoted $\boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{s}, \mathbf{a})$. Function approximation is the basis of deep reinforcement learning in Section 11.4, where it is possible to represent these complex functions using deep neural networks.

Note that, in the literature, there is often an abuse of notation, where $\boldsymbol{\pi}(\mathbf{s}, \mathbf{a})$ is used to denote the action taken, rather than the *probability* of taking an action $\mathbf{a}$ given a state observation $\mathbf{s}$. In the case of a deterministic policy, such as a greedy policy, then it may be possible to use $\mathbf{a} = \boldsymbol{\pi}(\mathbf{s})$ to represent the action taken. We will attempt to be clear throughout when choosing one convention over another.

The Environment: a Markov Decision Process (MDP)

In general, the measured state of the system may be a partial measurement of a higher-dimensional environmental state that evolves according to a stochastic, nonlinear dynamical system. For simplicity, most introductions to RL assume that the full state is measured and that it evolves according to a Markov decision process (MDP), so that the probability of the system occurring in the current state is determined only by the previous state. We will begin with this simple formulation. Even when it is assumed that the state evolves according to an MDP, it is often the case that this model is not known, motivating the use of "*model-free*" RL strategies discussed in Section 11.3. Similarly, when a model is not known, it may be possible to first learn an MDP using data-driven methods and then use this for "*model-based*" reinforcement learning, as in Section 11.2.

An MDP consists of a set of states $\mathcal{S}$, a set of actions $\mathcal{A}$, and a set of rewards $\mathcal{R}$, along with the probability of transitioning from state $\mathbf{s}_k$ at time t_k to state $\mathbf{s}_{k+1}$ at time t_{k+1} given action $\mathbf{a}_k$,

$$P(\mathbf{s}', \mathbf{s}, \mathbf{a}) = \Pr(\mathbf{s}_{k+1} = \mathbf{s}' \mid \mathbf{s}_k = \mathbf{s},\ \mathbf{a}_k = \mathbf{a}), \tag{11.3}$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

and a reward function R,

$$R(\mathbf{s}', \mathbf{s}, \mathbf{a}) = \Pr(\mathbf{r}_{k+1} \mid \mathbf{s}_{k+1} = \mathbf{s}', \mathbf{s}_k = \mathbf{s}, \mathbf{a}_k = \mathbf{a}). \tag{11.4}$$

Sometimes the transition probability $P(\mathbf{s}', \mathbf{s}, \mathbf{a})$ will be written as $P(\mathbf{s}' \mid \mathbf{s}, \mathbf{a})$. Again, sometimes there will be an abuse of notation, where a chosen policy $\boldsymbol{\pi}$ will be used instead of the action $\mathbf{a}$ in the argument of either P or R above. In this case, it is assumed that this applies a sum over states, as in

$$P(\mathbf{s}', \mathbf{s}, \boldsymbol{\pi}) = \sum_{\mathbf{a} \in \mathcal{A}} \boldsymbol{\pi}(\mathbf{s}, \mathbf{a}) P(\mathbf{s}', \mathbf{s}, \mathbf{a}). \tag{11.5}$$

Thus, an MDP generalizes the notion of a Markov process to include actions and rewards, making it suitable for decision making and control. A simple Markov process is a set of states $\mathcal{S}$ and a probability of transitioning from one state to the next. The defining property of a Markov process and an MDP is that the probability of being in a future state is entirely determined by the current state, and not by previous states or hidden variables. The MDP framework is closely related to transition state theory and the Perron–Frobenius operator, which is the adjoint of the Koopman operator from Section 7.4.

In the case of a simple Markov process with a finite set of states $\mathcal{S}$, then it is possible to let $\mathbf{s} \in \mathbb{R}^n$ be a vector of the probability of being in each of the n states, in which case the Markov process $P(\mathbf{s}', \mathbf{s})$ may be written in terms of a transition matrix, also known as a stochastic matrix, or a probability matrix, $\mathbf{T}$:

$$\mathbf{s}' = \mathbf{T}\mathbf{s}, \tag{11.6}$$

where each column of $\mathbf{T}$ must add up to 1, which is a statement of conservation of probability that, given a particular state $\mathbf{s}$, *something* must happen after the transition to $\mathbf{s}'$. Similarly, for an MDP, given a policy $\boldsymbol{\pi}$, the transition process may be written as

$$\mathbf{s}' = \sum_{\mathbf{a} \in \mathcal{A}} \boldsymbol{\pi}(\mathbf{s}, \mathbf{a}) \mathbf{T}_{\mathbf{a}} \mathbf{s}. \tag{11.7}$$

Now, for each action $\mathbf{a}$, $\mathbf{T}_{\mathbf{a}}$ is a Markov process with all columns summing to 1.

One of the defining properties of a Markov process is that the system asymptotically approaches a steady state $\boldsymbol{\mu}$, which is the eigenvector of $\mathbf{T}$ corresponding to eigenvalue 1. Similarly, given a policy $\boldsymbol{\pi}$, an MDP asymptotically approaches a steady state $\boldsymbol{\mu}_{\boldsymbol{\pi}} = \sum_{\mathbf{a}} \boldsymbol{\pi}(\mathbf{s}, \mathbf{a}) \boldsymbol{\mu}_{\mathbf{a}}$.

This brings up another notational issue, where, for continuous processes, $\mathbf{s} \in \mathbb{R}^n$ describes the continuous state vector in an n-dimensional vector space, as in Chapters 7 and 8; while, for discrete state spaces, $\mathbf{s} \in \mathbb{R}^n$ denotes a vector of probabilities of belonging to one of n finite states. It is important to carefully consider which notation is being used for a given problem, as these formulations have different dynamics (i.e., differential equation versus MDP) and interpretations (i.e., deterministic dynamics versus probabilistic transitions).

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

The Value Function

Given a policy $\boldsymbol{\pi}$, we next define a value function that quantifies the desirability of being in a given state:

$$V_{\boldsymbol{\pi}}(\mathbf{s}) = \mathbb{E}\left(\sum_{k=0}^{\infty} \gamma^k \mathbf{r}_k \,\middle|\, \mathbf{s}_0 = \mathbf{s} \right), \tag{11.8}$$

where $\mathbb{E}$ is the expected future reward, given a policy $\boldsymbol{\pi}$ and subject to a *discount rate* γ. Future rewards are discounted, reflecting the economic principle that current rewards are more valuable than future rewards. Often, the subscript $\boldsymbol{\pi}$ is omitted from the value function, in which case we refer to the value function for the best possible policy:

$$V(\mathbf{s}) = \max_{\boldsymbol{\pi}} \mathbb{E}\left(\sum_{k=0}^{\infty} \gamma^k \mathbf{r}_k \,\middle|\, \mathbf{s}_0 = \mathbf{s} \right). \tag{11.9}$$

One of the most important properties of the value function is that the value at a state $\mathbf{s}$ may be written recursively as

$$V(\mathbf{s}) = \max_{\boldsymbol{\pi}} \mathbb{E}\left(\mathbf{r}_0 + \sum_{k=1}^{\infty} \gamma^k \mathbf{r}_k \,\middle|\, \mathbf{s}_1 = \mathbf{s}' \right), \tag{11.10}$$

which implies that

$$V(\mathbf{s}) = \max_{\boldsymbol{\pi}} \mathbb{E}(\mathbf{r}_0 + \gamma V(\mathbf{s}')), \tag{11.11}$$

where $\mathbf{s}' = \mathbf{s}_{k+1}$ is the next state after $\mathbf{s} = \mathbf{s}_k$ given action $\mathbf{a}_k$, and the expectation is over actions selected from the optimal policy $\boldsymbol{\pi}$. This expression, known as *Bellman's equation*, is a statement of Bellman's principle of optimality, and it is a central result that underpins modern RL.

Given the value function, it is possible to extract the optimal policy as

$$\boldsymbol{\pi} = \operatorname*{argmax}_{\boldsymbol{\pi}} \mathbb{E}(\mathbf{r}_0 + \gamma V(\mathbf{s}')). \tag{11.12}$$

Goals and Challenges of Reinforcement Learning

Learning the policy $\boldsymbol{\pi}$, the value function V, or jointly learning both, is the central challenge in RL. Depending on the assumed structure of $\boldsymbol{\pi}$, the size of $\mathcal{S}$ and evolution dynamics P, and the reward landscape R, determining an optimal policy may range from a closed-form optimization to a rather high-dimensional unstructured optimization. Thus, a large number of trials must often be evaluated in order to determine an optimal policy. In practice, reinforcement learning

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

may be very expensive to train, and it might not be the right strategy for problems where testing a policy is expensive or potentially unsafe. Similarly, there are often simpler control strategies than RL, such as LQR or MPC; when these approaches are effective, they are often preferable. Reinforcement learning is, therefore, well suited for situations where some combination of the following are true: evaluating a policy is inexpensive, as in board games; there are sufficient resources to perform a near-brute-force optimization, as in evolutionary optimization; and/or no other control strategy works.

Although RL is typically formulated within the mathematical framework of MDPs, many real-world applications do not satisfy these assumptions. For a partially observed MDP (POMDP), the dynamics depend on the state history or on hidden variables. Similarly, the evolution dynamics may be entirely deterministic, yet chaotic. However, as we will see, it is often possible to develop approximate probabilistic transition state models for chaotic dynamics or to augment the environment state to include past states for systems with memory or hidden variables. Often, the underlying MDP transition probability and reward functions are not known *a priori*, and either must be learned ahead of time through some exploration phase, or alternative model-free optimization techniques must be used. Finally, many of the theoretical convergence results, and indeed many of the fundamental RL algorithms, only apply to *finite* MDPs, which are characterized by finite sets of actions $\mathcal{A}$ and states $\mathcal{S}$. Games, such as chess, fall into this category, even though the number of states may be combinatorially large. Continuous dynamical systems, such as a pendulum on a cart, may be approximated by a finite MDP through a discretization or quantization process.

There is typically much less supervisory information available to an RL agent than is available in classical supervised learning. One of the central challenges of reinforcement learning is that rewards are often extremely rare and may be significantly delayed from a sequence of good control actions. This challenge leads to the so-called credit assignment problem, coined by Minsky [503] to describe the challenge of knowing what action sequence was responsible for the reward ultimately received. These sparse and delayed rewards have been a central challenge in RL for six decades, and they are still a focus of research today. The resulting optimization problem is computationally expensive and data-intensive, requiring considerable trial and error.

Today, reinforcement learning is being used to learn sophisticated control policies for complex open-world problems in autonomy and propulsion (e.g., self-driving cars, learning to swim and fly, etc.) and as a general learning environment for rule-constrained games (e.g., checkers, backgammon, chess, go, Atari video games, etc.). Much of the history of RL may be traced through the success on increasingly challenging board games, from checkers [616] to backgammon [700] and more recently to chess and go [659]. These games serve

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.



Figure 11.2: Reinforcement learning is inspired by biological learning with sparse rewards. Mordecai was trained to balance a treat on his nose until a command is given, after which he catches it in the air. Credit: Bing Brunton for image and training.

to illustrate many of the central challenges that are still faced in RL, including the curse of dimensionality and the credit assignment problem.

Motivating Examples

It is helpful to understand RL through simple examples. Consider a mouse in a maze. The mouse is the agent, and the environment is the maze. The mouse measures the local state in its environment; it does not have access to a full top-down view of the maze, but instead it knows its current local environment and what past actions it has taken. The mouse has *agency* to take some action about what to do next, for example, whether to turn left, turn right, or go forward. Typically, the mouse does not receive a reward until the end of the maze. If the mouse received a reward after each correct turn, it would have a much simpler supervised learning task. Setting such a curriculum is a strategy to help teach animals, whereby initially dense rewards are sparsified throughout the learning process.

More generally, RL may be used to understand animal behavior, ranging from semi-supervised training to naturalistic behaviors. Figure 11.2 shows a trained behavior where a treat is balanced on Mordecai's nose until a command is given, after which he is able to catch it out of the air. Often, training animals to perform complex tasks involves expert human guidance to provide intermediate rewards or secondary reinforcers, such as using a clicker to indicate a future reward. In animal training and in RL, the more proximal the reward is in time to the action, the easier it is to learn the task. The connection between learning and temporal proximity is the basis of *temporal difference* (TD) learning, which is a powerful concept in RL, and this is also important to our understanding of the chemical basis for addiction [593].

It is also helpful to consider two-player games, such as tic-tac-toe, checkers, backgammon, chess, and go. In these games, the agent is one of the players, and the environment encompasses the rules of the game along with an adversarial opponent. These examples are also interesting because there is an element of randomness or stochasticity in the environment, either because of

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

the fundamental rules (e.g., a dice roll in backgammon) or because of an opponent's probabilistic strategy. Thus, it may be advantageous for the agent to also adopt a probabilistic policy, in contrast to much of the theory of classical control for deterministic systems. Similarly, a probabilistic strategy may be important when learning how to play.

In most games, the reward signal comes at the end of the game after the agent has won or lost. Again, this makes the learning process exceedingly challenging, as it is initially unclear which sub-sequence of actions were particularly important in driving the outcome. For example, an agent may play an excellent chess opening and mid-game and then lose at the end because of a few bad moves. Should the agent discard the entire first half of the game, or, worse yet, attribute this to a negative reward? Thus, it is clear that a major part of learning an effective policy is understanding the value of being in a given state $\mathbf{s}$. In a game like chess, where the number of states is combinatorially large, there are too many states to count, and it is intractable to map out the exact value of all board states. Instead, players create simple heuristic rules about what are good board positions, e.g., assigning points to the various pieces to keep track of a rough score. This intermediate score provides a denser reward structure throughout the game. However, these heuristics are sub-optimal and may be susceptible to gambits, where the opponent sacrifices a piece for an immediate point loss in order to eventually move to a more favorable global state $\mathbf{s}$. In backgammon, an intermediate point total may be more explicitly computed as the total number of *pips*, or points that a player must roll to move all pieces home and off the board. Although this makes it relatively simple to estimate the strength of a board position, the discrete nature of the die roll and game mechanics makes this a sub-optimal approximation, and the number of required *dice rolls* or *turns* may be a more useful measure.

Thinking through games like these illustrates many of the modern strategies to improve the learning rates and sample efficiency of RL, including hindsight replay, temporal difference learning, look ahead, and reward shaping, which we will discuss in the following sections. For example, playing against a skilled teacher can dramatically improve the learning rate, as the teacher provides guidance about whether or not a move is good, and why, adding information to help shape proxy metrics that can be used as intermediate rewards and models that can accelerate the learning process.

Categorization of RL Techniques

Nearly all problems in machine learning and control theory involve challenging optimization problems. In the case of machine learning, the parameters of a model are optimized to best fit the training data, as measured by a loss function. In the case of control, a set of control performance metrics are optimized

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

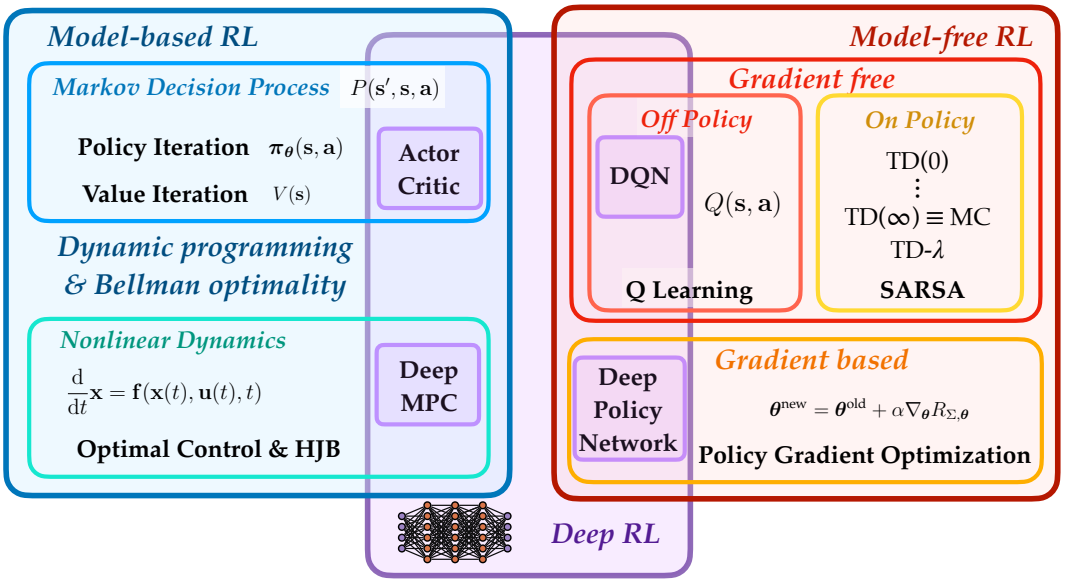


Figure 11.3: Rough categorization of reinforcement learning techniques. This organization is not comprehensive, and some of the lines are becoming blurred. The first major dichotomy is between model-based and model-free RL techniques. Next, within model-free RL, there is a dichotomy between gradient-based and gradient-free methods. Finally, within gradient-free methods, there is a dichotomy between on-policy and off-policy methods.

subject to the constraints of the dynamics. Reinforcement learning is no different, as it is at the intersection of machine learning and control theory.

There are many approaches to learn an optimal policy $\boldsymbol{\pi}$, which is the ultimate goal of RL. A major dichotomy in reinforcement learning is that of *model-based RL* versus *model-free RL*. When there is a known model for the environment, there are several strategies for learning either the optimal policy or value function through what is known as *policy iteration* or *value iteration*, which are forms of dynamic programming using the Bellman equation. When there is no model for the environment, alternative strategies, such as Q-learning, must be employed. The reinforcement learning optimization problem may be particularly challenging for high-dimensional systems with unknown, nonlinear, stochastic dynamics, and sparse and delayed rewards. All of these techniques may be combined with function approximation techniques, such as neural networks, for approximating the policy $\boldsymbol{\pi}$, the value function V, or the quality function Q (discussed in subsequent sections), making them more useful for high-dimensional systems. These model-based, model-free, and deep learning approaches will be discussed below. Figure 11.3 summarizes the main organization of these RL techniques.

Note that this section only provides a glimpse of the many optimization approaches used to solve RL problems, as this is a vast and rapidly growing field.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

11.2 Model-Based Optimization and Control

This section provides a high-level overview of some essential model-based optimization and control techniques. Some people do not consider these techniques to be reinforcement learning, as they do not involve learning an optimal strategy through trial-and-error experience. However, the techniques are closely related. It is possible to learn a model through trial and error, and then use this model with these techniques, which would be considered RL.

For the simplified case of a known model that is a finite MDP, it is possible to learn either the optimal policy or value function through what is known as *policy iteration* or *value iteration*, which are forms of dynamic programming using the Bellman equation. Dynamic programming [69, 70, 80, 81, 82, 606, 723] is a powerful approach that is used for general optimal nonlinear control and reinforcement learning, among other tasks. These algorithms provide a mathematically simplified optimization framework that helps to introduce essential concepts used throughout.

More generally, dynamic programming and RL optimization are related to the field of optimal nonlinear control, which has deep roots in variational theory going back to Bernoulli and the brachistochrone problem nearly four centuries ago. We will explore this connection to nonlinear control theory in Section 11.6.

Dynamic Programming

Dynamic programming is a mathematical framework introduced by Richard E. Bellman [69, 70] to solve large multi-step optimization problems, such as those found in decision making and control. Policy iteration and value iteration, discussed below, are two examples of the use of dynamic programming in reinforcement learning. To solve these multi-step optimizations, dynamic programming reformulates the large optimization problem as a recursive optimization in terms of smaller sub-problems, so that only a local decision need be optimized. This approach relies on Bellman's principle of optimality, which states that a large multi-step control policy must also be locally optimal in every sub-sequence of steps.

The Bellman equation in (11.11) indicates that the large optimization problem over an entire state–action trajectory $(\mathbf{s}_k, \mathbf{a}_k)$ may be broken into a recursive optimization at each point along the trajectory. As long as the value function is known at the next point $\mathbf{s}'$, it is possible to solve the optimization at point $\mathbf{s}$ simply by optimizing the policy $\boldsymbol{\pi}(\mathbf{s}, \mathbf{a})$ at this point. Of course, this assumes that the value function is known at *all* possible next states $\mathbf{s}' = \mathbf{s}_{k+1}$, which is a function of the current state $\mathbf{s}_k$, the current action $\mathbf{a}_k$, and the dynamics governing the system. This becomes even more complex for non-MDP dynamics,

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

such as the nonlinear control formulation in Section 11.6. For even moderately large problems, this approach suffers from the curse of dimensionality, and approximate solution methods must be employed.

When tractable, dynamic programming (i.e., the process of breaking a large problem into smaller overlapping sub-problems) provides a globally optimal solution. There are two main approaches to dynamic programming, referred to as *top down* and *bottom up*.

(a) **Top down:** The top-down approach involves maintaining a table of sub-problems that are referred to when solving larger problems. For a new problem, the table is checked to see if the relevant sub-problem has been solved. If so, it is used, and, if not, the sub-problem is solved. This tabular storage is called *memoization* and becomes combinatorially complex for many problems.

(b) **Bottom up:** The bottom-up approach involves starting by solving the smallest sub-problems first, and then combining these to form the larger problems. This may be thought of as working backwards from every possible goal state, finding the best previous action to get there, then going back two steps, then going back three steps, etc.

Although dynamic programming still represents a brute-force search through all sub-problems, it is still more efficient than a naive brute-force search. In some cases, it reduces the computational complexity to an algorithm that scales linearly with the number of sub-problems, although this may still be combinatorially large, as in the example of the game of chess. Dynamic programming is closely related to divide-and-conquer techniques, such as quick sort, except that divide and conquer applies to *non-overlapping* or *non-recursive* (i.e., independent) sub-problems, while dynamic programming applies to overlapping or recursively interdependent sub-problems.

The recursive structure of dynamic programming suggests approximate solution techniques, such as the alternating directions method, where a sub-optimal solution is initialized and the value function is iterated over. This will be discussed next.

Policy Iteration

Policy iteration is a two-step optimization procedure to simultaneously find an optimal value function V_π and the corresponding optimal policy $\boldsymbol{\pi}$.

First, a candidate policy $\boldsymbol{\pi}$ is evaluated, resulting in the value function for this fixed policy. This typically involves a brute-force calculation of the value function for this policy starting at many or all initial states. The policy may need to be simulated for a long time depending on the reward delay and discounting factor γ.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Next, the value function is fixed, and the policy is optimized to improve the expected rewards by taking different actions at a given state. This optimization relies on the recursive formulation of the value function due to Bellman's equation (11.11):

$$V_{\boldsymbol{\pi}}(\mathbf{s}) = \mathbb{E}(R(\mathbf{s}', \mathbf{s}, \boldsymbol{\pi}(\mathbf{s})) + \gamma V_{\boldsymbol{\pi}}(\mathbf{s}')) \tag{11.13a}$$

$$= \sum_{\mathbf{s}'} P(\mathbf{s}' \mid \mathbf{s}, \boldsymbol{\pi}(\mathbf{s}))(R(\mathbf{s}', \mathbf{s}, \boldsymbol{\pi}(\mathbf{s})) + \gamma V_{\boldsymbol{\pi}}(\mathbf{s}')). \tag{11.13b}$$

Note that, in this expression, we have assumed a deterministic policy $\mathbf{a} = \boldsymbol{\pi}(\mathbf{s})$, otherwise (11.13) would involve a second summation over $\mathbf{a} \in \mathcal{A}$, with the expression multiplied by $\boldsymbol{\pi}(\mathbf{s}, \mathbf{a})$.

It is then possible to fix $V_{\boldsymbol{\pi}}(\mathbf{s}')$ and optimize over the policy in the first term. In particular, the new deterministic optimal policy at the state $\mathbf{s}$ is given by

$$\boldsymbol{\pi}(\mathbf{s}) = \underset{\mathbf{a} \in \mathcal{A}}{\operatorname{argmax}}\ \mathbb{E}(R(\mathbf{s}', \mathbf{s}, \mathbf{a}) + \gamma V_{\boldsymbol{\pi}}(\mathbf{s}')). \tag{11.14}$$

Once the policy is updated, the process repeats, fixing this policy to update the value function, and then using this updated value function to improve the policy. The process is repeated until both the policy and the value function converge to within a specified tolerance. It is important to note that this procedure is both expensive and prone to finding local minima. It also resembles the alternating descent method that is widely used in optimization and machine learning.

The formulation in (11.13) makes it clear that it may be possible to optimize backwards from a state known to give a reward with high probability. Additionally, this approach requires having a model for P and R to predict the next state $\mathbf{s}'$, making this a *model-based* approach.

Value Iteration

Value iteration is similar to policy iteration, except that at every iteration only the value function is updated, and the optimal policy is extracted from this value function at the end. First, the value function is initialized, typically either with zeros or at random. Then, for all states $\mathbf{s} \in \mathcal{S}$, the value function is updated by returning the maximum value at that state across all actions $\mathbf{a} \in \mathcal{A}$, holding the value function fixed at all other states $\mathbf{s}' \in \mathcal{S} \setminus \mathbf{s}$:

$$V(\mathbf{s}) = \max_{\mathbf{a}} \sum_{\mathbf{s}'} P(\mathbf{s}' \mid \mathbf{s}, \mathbf{a})(R(\mathbf{s}', \mathbf{s}, \mathbf{a}) + \gamma V(\mathbf{s}')). \tag{11.15}$$

This iteration is repeated until a convergence criterion is met.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

After the value function converges, it is possible to extract the optimizing policy $\boldsymbol{\pi}$:

$$\boldsymbol{\pi}(\mathbf{s}, \mathbf{a}) = \underset{\mathbf{a}}{\operatorname{argmax}} \sum_{\mathbf{s}'} P(\mathbf{s}' \mid \mathbf{s}, \mathbf{a})(R(\mathbf{s}', \mathbf{s}, \mathbf{a}) + \gamma V(\mathbf{s}')). \tag{11.16}$$

Although value iteration typically requires fewer steps per iteration, policy iteration often converges in fewer iterations. This may be due to the fact that the value function is often more complex than the policy function, requiring more parameters to optimize over.

Note that the value function in RL typically refers to a discounted sum of future rewards that should be maximized, while in nonlinear control it refers to an integrated cost that should be minimized. The phrase *value function* is particularly intuitive when referring to accumulated rewards in the economic sense, as it quantifies the *value* of being in a given state. However, in the case of nonlinear control theory, the *value function* is more accurately thought of as quantifying the *numerical value* of the cost function evaluated on the optimal trajectory. This notation can be confusing and is worth careful consideration depending on the context.

Quality Function

Both policy iteration and value iteration rely on the quality function $Q(\mathbf{s}, \mathbf{a})$, which is defined as

$$\begin{aligned} Q(\mathbf{s}, \mathbf{a}) &= \mathbb{E}(R(\mathbf{s}', \mathbf{s}, \mathbf{a}) + \gamma V(\mathbf{s}')) && \text{(11.17a)} \\ &= \sum_{\mathbf{s}'} P(\mathbf{s}' \mid \mathbf{s}, \mathbf{a})(R(\mathbf{s}', \mathbf{s}, \mathbf{a}) + \gamma V(\mathbf{s}')). && \text{(11.17b)} \end{aligned}$$

In a sense, the optimal policy $\boldsymbol{\pi}(\mathbf{s}, \mathbf{a})$ and the optimal value function $V(\mathbf{s})$ contain redundant information, as one can be determined from the other via the quality function $Q(\mathbf{s}, \mathbf{a})$:

$$\begin{aligned} \boldsymbol{\pi}(\mathbf{s}, \mathbf{a}) &= \underset{\mathbf{a}}{\operatorname{argmax}}\, Q(\mathbf{s}, \mathbf{a}), && \text{(11.18a)} \\ V(\mathbf{s}) &= \max_{\mathbf{a}} Q(\mathbf{s}, \mathbf{a}). && \text{(11.18b)} \end{aligned}$$

This formulation will be used for model-free Q-learning [238, 722, 746] in Section 11.3.

11.3 Model-Free Reinforcement Learning and Q-Learning

Both policy iteration and value iteration above rely on the *quality* function $Q(\mathbf{s}, \mathbf{a})$, which describes the joint desirability of a given state–action pair. Policy

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

iteration (11.14) and value iteration (11.15) are both model-based reinforcement learning strategies, where it is assumed that the MDP model is known: each iteration requires a one-step look ahead, or model-based prediction of the next state $\mathbf{s}'$ given the current state $\mathbf{s}$ and action $\mathbf{a}$. Based on this model, it is possible to forecast and maximize over all possible actions.

When a model is not available, there are several reinforcement learning approaches to learn effective decision and control policies to interact with the environment. Perhaps the most straightforward approach is to first learn a model of the environment using some data-driven active learning strategy, and then use the standard model-based approaches discussed earlier. However, this may be infeasible for very large or particularly unstructured systems.

Q-learning is a leading model-free alternative, which learns the Q function directly from experience, without requiring access to a model. Thus, it is possible to generalize many of the model-based optimization strategies above to more unstructured settings, where a model is unavailable. The Q function has the one-step look ahead implicitly built into its representation, without needing to explicitly refer to a model. From this learned Q function, the optimal policy and value function may be extracted as in (11.18).

Before discussing the mechanics of Q-learning in detail, it is helpful to introduce several concepts, including Monte Carlo-based learning and temporal difference learning.

Monte Carlo Learning

In the simplest approach to learning from experience, the value function V or quality function Q may be learned through a Monte Carlo random sampling of the state–action space through repeated evaluation of many policies. Monte Carlo approaches require that the RL task is *episodic*, meaning that the task has a defined start and terminates after a finite number of actions, resulting in a total cumulative reward at the end of the episode. Games are good examples of episodic RL tasks.

In Monte Carlo learning, the total cumulative reward at the end of the task is used to estimate either the value function V or the quality function Q by dividing the final reward equally among all of the intermediate states or state–action pairs, respectively. This is the simplest possible approach to deal with the credit assignment problem, as credit is shared equally among all intermediate steps. However, for this reason, Monte Carlo learning is typically quite sample-inefficient, especially for problems with sparse rewards.

Consider the case of Monte Carlo learning of the value function. Given a new episode consisting of n steps, the cumulative discounted reward R_Σ is

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

computed as

$$R_\Sigma = \sum_{k=1}^{n} \gamma^k \mathbf{r}_k \tag{11.19}$$

and used to update the value function at every state $\mathbf{s}_k$ visited in this episode:

$$V^{\text{new}}(\mathbf{s}_k) = V^{\text{old}}(\mathbf{s}_k) + \frac{1}{n}\left(R_\Sigma - V^{\text{old}}(\mathbf{s}_k)\right) \quad \forall\, k \in [1, \ldots, n]. \tag{11.20}$$

This update, weighted by $1/n$, is equivalent to waiting until the end of the episode and then updating the value function at all states along the trajectory with an equal share of the reward. Similarly, in the case of Monte Carlo learning of the Q function, the discounted reward R_Σ is used to update the Q function at every state–action pair $(\mathbf{s}_k, \mathbf{a}_k)$ visited in this episode:

$$Q^{\text{new}}(\mathbf{s}_k, \mathbf{a}_k) = Q^{\text{old}}(\mathbf{s}_k, \mathbf{a}_k) + \frac{1}{n}\left(R_\Sigma - Q^{\text{old}}(\mathbf{s}_k, \mathbf{a}_k)\right) \quad \forall\, k \in [1, \ldots, n]. \tag{11.21}$$

In the limit of infinite data and infinite exploration, this approach will eventually sample all possible state–action pairs and converge to the true quality function Q. However, in practice, this often amounts to an intractable brute-force search.

It is also possible to discount past experiences by introducing a learning rate $\alpha \in [0, 1]$ and using this to update the Q function:

$$Q^{\text{new}}(\mathbf{s}_k, \mathbf{a}_k) = Q^{\text{old}}(\mathbf{s}_k, \mathbf{a}_k) + \alpha\left(R_\Sigma - Q^{\text{old}}(\mathbf{s}_k, \mathbf{a}_k)\right) \quad \forall\, k \in [1, \cdots, n]. \tag{11.22}$$

Larger learning rates $\alpha > 1/n$ will favor more recent experience.

There is a question about how to initialize the many episodes required to learn with Monte Carlo. When possible, the episode will be initialized randomly at every initial state or state–action pair, providing a random sampling; however, this might not be possible for many learning tasks. Typically, Monte Carlo learning is performed *on-policy*, meaning that the optimal policy is enacted, based on the current value or quality function, and the information from this locally optimal policy is used for the update. It is also possible to promote exploration by adding a small probability of taking a random action, rather than the action dictated by the optimal policy. Finally, there are off-policy Monte Carlo methods, but, in general, they are quite inefficient or unfeasible.

Temporal Difference (TD) Learning

Temporal difference learning [103, 197, 682, 699, 700], known as TD learning, is another sample-based learning strategy. In contrast to Monte Carlo learning, TD learning is not restricted to episodic tasks, but instead learns continuously by bootstrapping based on current estimates of the value function V or

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

quality function Q, as in dynamic programming (e.g., as in value iteration in (11.15)). TD learning is designed to mimic learning processes in animals, where time-delayed rewards are often learned through environmental cues that act as secondary reinforcers preceding the delayed reward; this is most popularly understood through the story of Pavlov's dog [551]. Thus, TD learning is typically more sample efficient than Monte Carlo learning, resulting in decreased variance, but at the cost of a bias in the learning due to the bootstrapping. We will demonstrate TD learning of the value function, although it can also be used to learn the quality function.

TD(0): One-Step Look Ahead

To understand TD learning, it is helpful to begin with the simplest algorithm: TD(0). In TD(0), the estimate of the one-step-ahead future reward is used to update the current value function.

Given a control trajectory generated through an optimal policy $\boldsymbol{\pi}$, the value function at state $\mathbf{s}_k$ is given by

$$V(\mathbf{s}_k) = \mathbb{E}(\mathbf{r}_k + \gamma V(\mathbf{s}_{k+1})). \tag{11.23}$$

In the language of Bayesian statistics, $\mathbf{r}_k + \gamma V(\mathbf{s}_{k+1})$ is an *unbiased estimator* for $V(\mathbf{s}_k)$.

Instead of using a model to predict $\mathbf{s}_{k+1}$, which is required to evaluate $V(\mathbf{s}_{k+1})$, it is possible to wait until the next step is actually taken and retroactively adjust the value function:

$$V^{\text{new}}(\mathbf{s}_k) = V^{\text{old}}(\mathbf{s}_k) + \alpha \overbrace{\left(\underbrace{\mathbf{r}_k + \gamma V^{\text{old}}(\mathbf{s}_{k+1})}_{\text{TD target estimates } R_\Sigma} - V^{\text{old}}(\mathbf{s}_k) \right)}^{\text{TD error}}. \tag{11.24}$$

For non-optimal policies $\boldsymbol{\pi}$, this same idea may be used to update the value function based on the value function one step in the future. Notice that this is very similar to optimization of the Bellman equation using dynamic programming but with retroactive updates based on sampled data rather than proactive updates based on a model prediction.

In the TD(0) update above, the expression $R_\Sigma = \mathbf{r}_k + \gamma V(\mathbf{s}_{k+1})$ is known as the *TD target*, as it is the estimate for the future reward, analogous to R_Σ in Monte Carlo learning of the Q function in (11.22). The difference between this target and the previous estimate of the value function is the TD error, and it is used to update the value function, just as in Monte Carlo learning, with a learning rate α.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

TD(n): n-Step Look Ahead

Other temporal difference algorithms can be developed, based on multi-step look-aheads into the future. For example, TD(1) uses a TD target based on two steps into the future,

$$R_\Sigma^{(1)} = \mathbf{r}_k + \gamma \mathbf{r}_{k+1} + \gamma^2 V(\mathbf{s}_{k+2}), \tag{11.25}$$

and TD(n) uses a TD target based on $n+1$ steps into the future,

$$R_\Sigma^{(n)} = \mathbf{r}_k + \gamma \mathbf{r}_{k+1} + \gamma^2 \mathbf{r}_{k+2} + \cdots + \gamma^n \mathbf{r}_{k+n} + \gamma^{n+1} V(\mathbf{s}_{k+n+1}) \tag{11.26a}$$

$$= \left(\sum_{j=0}^{n} \gamma^j \mathbf{r}_{k+j} \right) + \gamma^{n+1} V(\mathbf{s}_{k+n+1}). \tag{11.26b}$$

Again, there does not need to be a model for these future states, but, instead, the value function may be retroactively adjusted based on the actual sampled trajectory and rewards. Note that in the limit that an entire episode is used, TD(n) converges to the Monte Carlo learning approach.

TD-λ: Weighted Look Ahead

An important variant of the TD learning family is TD-λ, which was introduced by Sutton [682]. TD-λ creates a TD target R_Σ^λ that is a weighted average of the various TD(n) targets $R_\Sigma^{(n)}$. The weighting is given by

$$R_\Sigma^\lambda = (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} R_\Sigma^{(n)} \tag{11.27}$$

and the update equation is

$$V^{\text{new}}(\mathbf{s}_k) = V^{\text{old}}(\mathbf{s}_k) + \alpha (R_\Sigma^\lambda - V^{\text{old}}(\mathbf{s}_k)). \tag{11.28}$$

TD-λ was used for an impressive demonstration in the game of backgammon by Tesauro in 1995 [700].

TD learning provides one of the strongest connections between reinforcement learning and learning in biological systems. Neural circuits are believed to estimate the future reward, and feedback is based on the difference between the expected reward and the actual reward, which is closely related to the TD error. In fact, there are specific neurotransmitter feedback loops that strengthen connections based on proximity of their firing to a dopamine reward signal [645, 282]. The closer the proximity in time between an action and a reward, the stronger the feedback, which has implications for addiction.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Bias–Variance Tradeoff

Monte Carlo learning and TD learning exemplify the *bias–variance tradeoff* in machine learning. Monte Carlo learning typically has high variance but no bias, while TD learning has lower variance but introduces a bias because of the bootstrapping. Although the *true* TD target $\mathbf{r}_k + \gamma V(\mathbf{s}_{k+1})$ is an unbiased estimate of $V(\mathbf{s}_k)$ for an optimal policy $\boldsymbol{\pi}$, the sampled TD target is a biased estimate, because it uses sub-optimal actions and the current imperfect estimate of the value function.

SARSA: State–Action–Reward–State–Action Learning

SARSA is a popular TD algorithm that is used to learn the Q function *on-policy*. The Q update equation in SARSA(0) is nearly identical to the V update equation (11.24) in TD(0):

$$Q^{\text{new}}(\mathbf{s}_k, \mathbf{a}_k) = Q^{\text{old}}(\mathbf{s}_k, \mathbf{a}_k) + \alpha \left(\mathbf{r}_k + \gamma Q^{\text{old}}(\mathbf{s}_{k+1}, \mathbf{a}_{k+1}) - Q^{\text{old}}(\mathbf{s}_k, \mathbf{a}_k)\right). \quad (11.29)$$

There are SARSA variants for all of the TD(n) algorithms, based on the n-step TD target:

$$R_\Sigma^{(n)} = \mathbf{r}_k + \gamma \mathbf{r}_{k+1} + \gamma^2 \mathbf{r}_{k+2} + \cdots + \gamma^n \mathbf{r}_{k+n} + \gamma^{n+1} Q(\mathbf{s}_{k+n+1}, \mathbf{a}_{k+n+1}) \quad (11.30a)$$

$$= \sum_{j=0}^{n} \gamma^j \mathbf{r}_{k+j} + \gamma^{n+1} Q(\mathbf{s}_{k+n+1}, \mathbf{a}_{k+n+1}). \quad (11.30b)$$

In this case, the SARSA(n) update equation is given by

$$Q^{\text{new}}(\mathbf{s}_k, \mathbf{a}_k) = Q^{\text{old}}(\mathbf{s}_k, \mathbf{a}_k) + \alpha \left(R_\Sigma^{(n)} - Q^{\text{old}}(\mathbf{s}_k, \mathbf{a}_k)\right). \quad (11.31)$$

Note that this is on-policy because the actual action sequence $\mathbf{a}_k, \mathbf{a}_{k+1}, \ldots, \mathbf{a}_{k+n+1}$ has been used to receive the rewards $\mathbf{r}$ and evaluate the $(n+1)$-step Q function $Q(\mathbf{s}_{k+n+1}, \mathbf{a}_{k+n+1})$.

Q-Learning

We are now ready to discuss Q-learning [238, 722, 746], which is one of the most widely used approaches in model-free RL. Q-learning is essentially an *off-policy* TD learning scheme for the Q function. In Q-learning, the Q update equation is

$$Q^{\text{new}}(\mathbf{s}_k, \mathbf{a}_k) = Q^{\text{old}}(\mathbf{s}_k, \mathbf{a}_k) + \alpha \left(\mathbf{r}_k + \gamma \max_{\mathbf{a}} Q(\mathbf{s}_{k+1}, \mathbf{a}) - Q^{\text{old}}(\mathbf{s}_k, \mathbf{a}_k)\right). \quad (11.32)$$

Notice that the only difference between Q-learning and SARSA(0) is that SARSA(0) uses $Q(\mathbf{s}_{k+1}, \mathbf{a}_{k+1})$ for the TD target, while Q-learning uses $\max_{\mathbf{a}} Q(\mathbf{s}_{k+1}, \mathbf{a})$ for

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

the TD target. Thus, SARSA(0) is considered *on-policy* because it uses the action $\mathbf{a}_{k+1}$ based on the actual policy: $\mathbf{a}_{k+1} = \boldsymbol{\pi}(\mathbf{s}_{k+1})$. In contrast, Q-learning is *off-policy* because it uses the optimal $\mathbf{a}$ for the update based on the current estimate for Q, while taking a different action $\mathbf{a}_{k+1}$ based on a different behavior policy. Thus, Q-learning may take sub-optimal actions $\mathbf{a}_{k+1}$ to explore, while still using the optimal action $\mathbf{a}$ to update the Q function.

Generally, Q-learning will learn a more optimal solution faster than SARSA, but with more variance in the solution. However, SARSA will typically yield more cumulative rewards during the training process, since it is on-policy. In safety-critical applications, such as self-driving cars or other applications where there can be catastrophic failure, SARSA will typically learn less optimal solutions, but with a better safety margin, since it maximizes on-policy rewards.

Q-learning applies to discrete action spaces $\mathcal{A}$ and state spaces $\mathcal{S}$ governed by a finite MDP. The Q function is classically represented as a table of Q values that is updated through some iteration based on new information as a policy is tested and evaluated. However, this tabular approach does not scale well to large state spaces, and so typically function approximation is used to represent the Q function, such as a neural network in deep Q-learning. Even if the action and state spaces are continuous, as in the pendulum on a cart system, it is possible to discretize and then apply Q-learning.

In addition to being model-free, Q-learning is also referred to as *off-policy* RL, as it does not require that an optimal policy is enacted, as in policy iteration and value iteration. Off-policy learning is more realistic in real-world applications, enabling the RL agent to improve when its policy is sub-optimal and by watching and imitating other more skilled agents. Q-learning is especially good for games, such as backgammon, chess, and go. In particular, deep Q-learning, which approximates the Q function using a deep neural network, has been used to surpass the world champions in these challenging games.

Experience Replay and Imitation Learning

Because Q-learning is off-policy, it is possible to learn from action–state sequences that do not use the current optimal policy. For example, it is possible to store past experiences, such as previously played games, and *replay* these experiences to further improve the Q function.

In an on-policy strategy, such as SARSA, using actions that are sub-optimal, based on the current optimal policy, will degrade the Q function, since the TD target will be a flawed estimate of future rewards based on a sub-optimal action. However, in Q-learning, since the action is optimized over the current Q function in the update, it is possible to learn from experience resulting from sub-optimal actions. This also makes it possible to learn from watching other, more experienced agents, which is related to imitation learning [217, 330, 344,

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

625].

Experience replay is deeply intuitive, as it is closely related to how we learn, through recalling past experiences in the light of new knowledge (i.e., an updated Q function). Similarly, imitation learning is perhaps one of the most fundamental first steps in biological learning.

Exploration versus Exploitation: ϵ-Greedy Actions

It is important to introduce an element of random exploration into Q-learning, and there are several techniques. One approach is the ϵ-greedy algorithm to select the next action. In this approach, the agent takes the current optimal action $\mathbf{a}_k = \max_{\mathbf{a}} Q(\mathbf{s}_k, \mathbf{a})$, based on the current Q function, with probability $1-\epsilon$, where $\epsilon \in [0, 1]$. With probability ϵ, the agent takes a random action. Thus, the agent balances exploration with the random actions, and exploitation with the optimal actions. Larger ϵ promotes more random exploration.

Typically, the value of ϵ will be initialized to a large value, often $\epsilon = 1$. Throughout the course of training, ϵ decays so that, as the Q function improves, the agent increasingly takes the current optimal action. This is closely related to simulated annealing from optimization, which mimics the process of forging metal to find a low-energy state through a specific cooling schedule.

Policy Gradient Optimization

Policy gradient optimization [368, 660, 684] is a powerful technique to optimize a policy that is parameterized, as in (11.2). When the policy $\boldsymbol{\pi}$ is parameterized by $\boldsymbol{\theta}$, it is possible to use gradient optimization on the parameters to improve the policy much faster than through traditional iteration. The parameterization may be a multi-layer neural network, in which case this would be a *deep policy network*, although other representations and function approximations may be useful. In any case, instead of extracting the policy as the argument maximizing the value or quality functions, it is possible to directly optimize the parameters $\boldsymbol{\theta}$, for example through gradient descent or stochastic gradient descent. The value function $V_{\boldsymbol{\pi}}(\mathbf{s})$, depending on a policy $\boldsymbol{\pi}$, then becomes $V(\mathbf{s}, \boldsymbol{\theta})$, and a similar modification is possible for the quality function Q.

The total estimated reward is given by

$$R_{\Sigma,\boldsymbol{\theta}} = \mathbb{E}\left(Q(\mathbf{s}, \mathbf{a})\right) = \sum_{\mathbf{s}\in\mathcal{S}} \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{s}) \sum_{\mathbf{a}\in\mathcal{A}} \boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{s}, \mathbf{a}) Q(\mathbf{s}, \mathbf{a}), \tag{11.33}$$

where $\boldsymbol{\mu}_{\boldsymbol{\theta}}$ is the asymptotic steady state of the MDP given a policy $\boldsymbol{\pi}_{\boldsymbol{\theta}}$ parameterized by $\boldsymbol{\theta}$. It is then possible to compute the gradient of the total estimated

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

reward with respect to $\boldsymbol{\theta}$:

$$\nabla_{\boldsymbol{\theta}} R_{\Sigma,\boldsymbol{\theta}} = \sum_{\mathbf{s}\in\mathcal{S}} \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{s}) \sum_{\mathbf{a}\in\mathcal{A}} Q(\mathbf{s},\mathbf{a}) \nabla_{\boldsymbol{\theta}} \boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{s},\mathbf{a}) \tag{11.34a}$$

$$= \sum_{\mathbf{s}\in\mathcal{S}} \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{s}) \sum_{\mathbf{a}\in\mathcal{A}} \boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{s},\mathbf{a}) Q(\mathbf{s},\mathbf{a}) \frac{\nabla_{\boldsymbol{\theta}} \boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{s},\mathbf{a})}{\boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{s},\mathbf{a})} \tag{11.34b}$$

$$= \sum_{\mathbf{s}\in\mathcal{S}} \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{s}) \sum_{\mathbf{a}\in\mathcal{A}} \boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{s},\mathbf{a}) Q(\mathbf{s},\mathbf{a}) \nabla_{\boldsymbol{\theta}} \log(\boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{s},\mathbf{a})) \tag{11.34c}$$

$$= \mathbb{E}(Q(\mathbf{s},\mathbf{a}) \nabla_{\boldsymbol{\theta}} \log(\boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{s},\mathbf{a}))). \tag{11.34d}$$

Then the policy parameters may be updated as

$$\boldsymbol{\theta}^{\text{new}} = \boldsymbol{\theta}^{\text{old}} + \alpha \nabla_{\boldsymbol{\theta}} R_{\Sigma,\boldsymbol{\theta}}, \tag{11.35}$$

where α is the learning weight; note that α may be replaced with a vector of learning weights for each component of $\boldsymbol{\theta}$. There are several approaches to approximating this gradient, including through finite differences, the REINFORCE algorithm [759], and natural policy gradients [368].

11.4 Deep Reinforcement Learning

Deep reinforcement learning is one of the most exciting areas of machine learning and of control theory, and it is one of the most promising avenues of research towards generalized artificial intelligence. Deep learning has revolutionized our ability to represent complicated functions from data, providing a set of architectures for achieving human-level performance in complex tasks such as image recognition and natural language processing. Classic reinforcement learning suffers from a representation problem, as many of the relevant functions, such as the policy $\boldsymbol{\pi}$, the value function V, and the quality function Q, may be exceedingly complicated functions defined over a very high-dimensional state and action space. Indeed, even for simple games, such as the 1972 Atari game Pong, the black-and-white screen at standard resolution 336×240 has over $10^{24\,000}$ possible discrete states, making it infeasible to represent any of these functions exactly without approximation. Thus, deep learning provides a powerful tool for improving these representations.

It is possible to use deep learning in several different ways to approximate the various functions used in RL, or to model the environment more generally. Typically, the central challenge is in identifying and representing key features in a high-dimensional state space. For example, the policy $\boldsymbol{\pi}(\mathbf{s},\mathbf{a})$ may now be approximated by

$$\boldsymbol{\pi}(\mathbf{s},\mathbf{a}) \approx \boldsymbol{\pi}(\mathbf{s},\mathbf{a},\boldsymbol{\theta}), \tag{11.36}$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

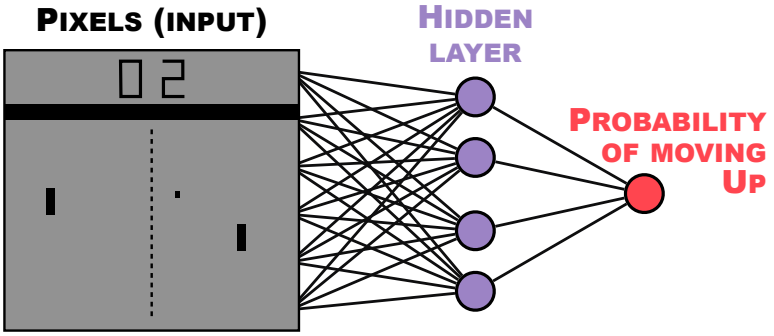


Figure 11.4: Deep policy network to encode the probability of moving up in the game of Pong. Inspired by Andrej Karpathy's Blog, "Deep Reinforcement Learning: Pong from Pixels" at `http://karpathy.github.io/2016/05/31/rl/`.

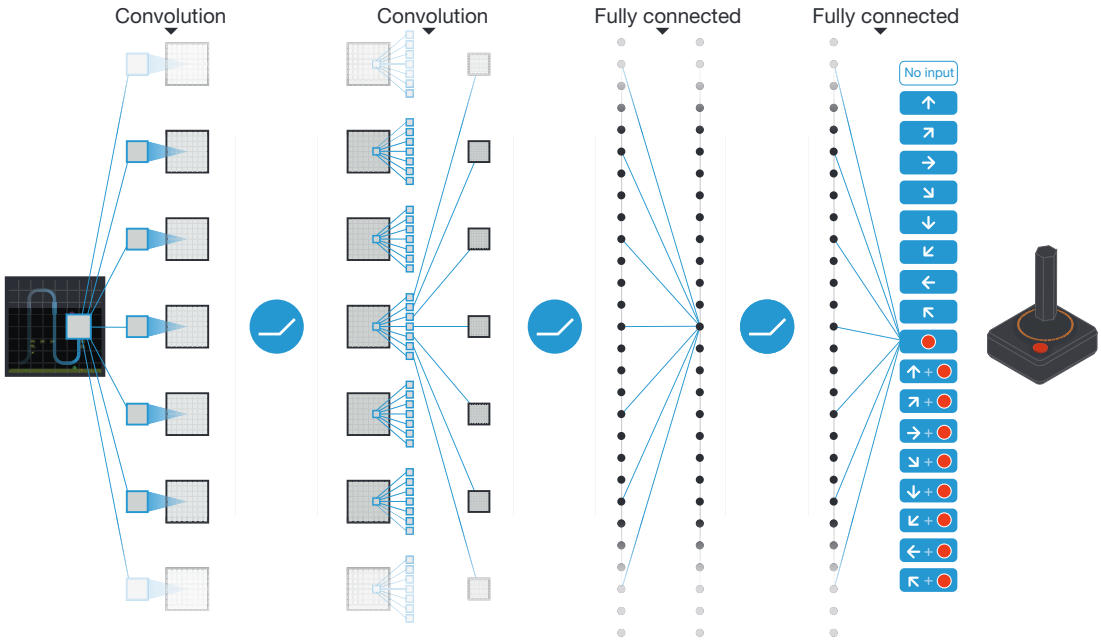


Figure 11.5: Convolutional structure of deep Q network used to play Atari games. Reproduced with permission from [506].

where $\boldsymbol{\theta}$ represent the weights of a neural network.

This pairing of deep learning for representations with reinforcement learning for decision making and control has resulted in dramatic improvements to the capabilities of reinforcement learning. For example, Fig. 11.4 shows a simple policy network designed to play Pong, and Fig. 11.5 shows a more general deep convolutional neural network architecture used to develop a deep Q net-

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

work to play Atari games at human levels of performance [506].

Much of what is discussed in this section is also relevant for other function approximation techniques besides deep learning. For example, policy gradients may be computed and used for gradient-based optimization using other representations of the form of (11.36), and there is a long history before deep learning [368, 684]. That said, many of the most exciting and impressive recent demonstrations of RL leverage the full power of deep learning, and so we present these innovations in this context.

Deep Q-Learning

Many of the most exciting advances in RL over the past decade have involved some variation of deep Q-learning, which uses deep neural networks to represent the quality function Q. As with the policy in (11.36), it is possible to approximate the Q function through some parameterization $\boldsymbol{\theta}$,

$$Q(\mathbf{s}, \mathbf{a}) \approx Q(\mathbf{s}, \mathbf{a}, \boldsymbol{\theta}), \tag{11.37}$$

where $\boldsymbol{\theta}$ represents the weights of a deep neural network. In this representation, the training loss function is directly related to the standard Q-learning update in (11.32):

$$\mathcal{L} = \mathbb{E}\left[\left(\mathbf{r}_k + \gamma \max_{\mathbf{a}} Q(\mathbf{s}_{k+1}, \mathbf{a}, \boldsymbol{\theta}) - Q(\mathbf{s}_k, \mathbf{a}_k, \boldsymbol{\theta})\right)^2\right]. \tag{11.38}$$

The first part of the loss function, $\mathbf{r}_k + \gamma \max_{\mathbf{a}} Q(\mathbf{s}_{k+1}, \mathbf{a}, \boldsymbol{\theta})$, is the temporal difference target from before, and the second part, $Q(\mathbf{s}_k, \mathbf{a}_k, \boldsymbol{\theta})$, is the prediction.

Deep reinforcement learning based on a deep Q network (DQN) was introduced by Mnih et al. [506] to play Atari games. Specifically, this network used a deep convolutional neural network to represent the Q function, where the inputs were pixels from the Atari screen and actions were joystick motions, as shown in Fig. 11.5. In this original paper, both Q functions in (11.38) were represented by the same network weights $\boldsymbol{\theta}$. However, in a double DQN [730], different networks are used to represent the target and prediction Q functions, which reduces bias due to inaccuracies early in training. In double DQN, it may be necessary to fix the target network for multiple training iterations of the prediction network before updating to improve stability and convergence [258].

Experience replay is a critical component of training a DQN, which is possible because it is an off-policy RL algorithm. Short segments of past experiences are used in batches for stochastic gradient descent during training. Moreover, to place more importance on experiences with large model mismatch, it is possible to weight past experiences by the magnitude of the TD error. This process is known as prioritized experience replay [630].

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Dueling deep Q networks (DDQNs) [745] are another important deep Q learning architecture that are used to improve training when actions have a marginal affect on the quality function. In particular, a DDQN splits the quality function into the sum of a value function and an *advantage* function $A(\mathbf{s}, \mathbf{a})$, which quantifies the additional benefit of a particular action over the value of being in that state:

$$Q(\mathbf{s}, \mathbf{a}, \boldsymbol{\theta}) = V(\mathbf{s}, \boldsymbol{\theta}_1) + A(\mathbf{s}, \mathbf{a}, \boldsymbol{\theta}_2). \tag{11.39}$$

Thus, separate value and advantage networks are combined to estimate the Q function.

There are a variety of other useful architectures for deep Q learning, with more introduced regularly. For example, deep recurrent Q networks are promising for dynamic problems [316]. Advantage actor–critic networks, discussed in the next section, combine the DDQN with deep policy networks.

Actor–Critic Networks

Actor–critic methods in reinforcement learning simultaneously learn a policy function and a value function, with the goal of taking the best of both value-based and policy-based learning. The basic idea is to have an actor, which is policy-based, and a critic, which is value-based, and to use the temporal difference signal from the critic to update the policy parameters. There are many actor–critic methods that pre-date deep learning. For example, a simple actor–critic approach would update the policy parameters $\boldsymbol{\theta}$ in (11.36) using the temporal difference error $\mathbf{r}_k + \gamma V(\mathbf{s}_{k+1}) - V(\mathbf{s}_k)$:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \alpha \nabla_{\boldsymbol{\theta}}((\log \boldsymbol{\pi}(\mathbf{s}_k, \mathbf{a}_k, \boldsymbol{\theta}))(\mathbf{r}_k + \gamma V(\mathbf{s}_{k+1}) - V(\mathbf{s}_k)). \tag{11.40}$$

It is rather straightforward to incorporate deep learning into an actor–critic framework. For example, in the advantage actor–critic (A2C) network, the actor is a deep policy network, and the critic is a DDQN. In this case, the update is given by

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \alpha \nabla_{\boldsymbol{\theta}}((\log \boldsymbol{\pi}(\mathbf{s}_k, \mathbf{a}_k, \boldsymbol{\theta}))Q(\mathbf{s}_k, \mathbf{a}_k, \boldsymbol{\theta}_2)). \tag{11.41}$$

Challenges and Additional Techniques

There are several important innovations that are necessary to make reinforcement learning tractable for even moderately challenging tasks. Two of the biggest challenges in RL are: (1) high-dimensional state and action spaces, and (2) sparse and delayed rewards.

Many games, such as chess and go, have exceedingly large state spaces. For example, Claude Shannon estimated the number of possible games of chess,

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

known as the Shannon number, at around 10^{120} in his famous paper "Programming a computer for playing chess" [654]; this paper was a major inspiration for modern dynamic programming and reinforcement learning. Representing a value or quality function, let alone sampling over these states, is beyond astronomically difficult. Thus, approximate representations of the value or quality functions using approximation theory, such as deep neural networks, are necessary.

Sparse and delayed rewards represent the central challenge of reinforcement learning, leading to the well-known credit assignment problem, which we have seen multiple times at this point. The following techniques, including reward shaping and hindsight experience replay, are leading techniques to overcome the credit assignment problem.

Reward Shaping

Perhaps the most standard approach for systems with sparse rewards is a technique called reward shaping. This involves designing customized proxy features that are indicative of a future reward and that may be used as an intermediate reward signal. For example, in the game of chess, the relative point count, where each piece is assigned a numeric value (e.g., a queen is worth 10 points, rooks are worth 5, knights and bishops are worth 3, and pawns are worth 1 point), is an example of a shaped reward that gives an intermediate reward signal each time a piece is taken.

Reward shaping is quite common and can be very effective. However, these rewards require expert human guidance to design, and this requires customized effort for each new task. Thus, reward shaping is not a viable strategy for a generalized artificial intelligence agent capable of learning multiple games or tasks. In addition, reward shaping generally limits the upper end of the agent's performance to that of the human expert.

Hindsight Experience Replay

In many tasks, such as robotic manipulation, the goal is to move the robot or an object from one location to another. For example, consider a robot arm that is required to slide an object on a table from point A to point B. Without a detailed physical model, or other prior knowledge, it is extremely unlikely that a random control policy will result in the object actually reaching the desired destination, so the rewards may be very sparse. It is possible to shape a reward based on the distance of the object to the goal state, although this is not a general strategy and suffers from the limitations discussed above.

Hindsight experience replay (HER) [22, 429] is a strategy that enriches the reward signal by taking failed trials and pretending that they were successful at a different task. This approach makes the reward structure much more dense,

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

and has the benefit of enabling the simultaneous learning of a whole family of motion tasks.

HER is quite intuitive in the context of human learning, for example in the case of tennis. Initially, it is difficult to aim the ball, and shots often go wild when learning. However, this provides valuable information about those muscle actions that might be useful for future tasks. After lots of practice, it then becomes possible to pick from different shots and place the ball more deliberately.

Curiosity-Driven Exploration

Another challenge with RL for large open-world environments is that the agent may easily get stuck in a local minimum, where it over-optimizes for a small region of state space. One approach to this problem is to augment the reward signal with a *novelty* reward that is large in regions of state space that are not well modeled. This is known as curiosity-driven exploration [549], and it involves an intrinsic curiosity module (ICM), which compares a forward model of the evolution of the state, or a latent representation of the state, with the actual observed evolution. The discrepancy between the model and the actual dynamics is the novelty reward. When this difference is large, the agent becomes *curious* and explores this region more. There are similarities between this approach and TD learning, and, in fact, many of the same variations may be implemented for curiosity-driven exploration. The main difference is that, in TD learning, the reward discrepancy is used as feedback to improve the value or quality function; while, in curiosity-driven exploration, the discrepancy is explicitly used as an additional reward signal. This is a clever approach to embedding this fundamental behavior of intelligent biological learning systems, to be curious and explore.

There are challenges when using this novelty reward for chaotic and stochastically driven systems, where there are aspects of the state evolution that are fundamentally unpredictable. A naive novelty reward would constantly provide positive incentive to explore these regions, since the forward model will not improve. Instead, the authors in [549] overcome this challenge by predicating novelty on the predictability of an outcome given the action, using latent features in an autoencoder, so only aspects of the future state that can be affected by the agent's actions are included in the novelty signal.

11.5 Applications and Environments

Here we provide a brief overview of some of the modern applications and success stories of RL, along with some common environments.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

OpenAI Gym

The OpenAI Gym is an incredible open-source resource to develop and test reinforcement learning algorithms in a wide range of environments. Figure 11.6 shows a small selection of these systems. Example environments include the following.

- Classic Atari video games: over 100 tasks on Atari 2600 games, including asteroids, breakout, space invaders, and many others.
- Classic control benchmarks: tasks include balancing an inverted pendulum on a cart, swing-up of a pendulum, swing-up of a double pendulum, and driving up a hill with an under-actuated system.
- Goal-based robotics [769]: tasks include pushing or fetching a block to a goal position with a robot arm, with and without sliding after loss of contact, and robotic hand manipulation for reaching a pose or orienting various objects.
- MuJoCo [706]: tasks include multi-legged locomotion, running, hopping, swimming, etc., within a fast physics simulator environment.

This wide range of environments and tasks provides an invaluable resource for RL researchers, dramatically lowering the barrier to entry and facilitating the benchmarking and comparison of new algorithms.

Classic Board Games

As discussed throughout this chapter, RL has developed tremendously over the past half-century, from a biologically inspired idea to a major field at the forefront of generalized artificial intelligence. This progress can be largely traced through the success of RL on increasingly challenging games, where RL has learned to interact with and mimic humans, and eventually to defeat our greatest Grandmasters.

Many of the most fundamental advances in RL were either developed for the purpose of playing games, or demonstrated on the most challenging games of the time. These simple board games also make the struggles of machine learning and artificial intelligence more relatable to humans,[2] as we can reflect on our own experiences learning first how to play tic-tac-toe, then checkers, and then eventually "real" games, such as backgammon, chess, and go. The progression of RL capabilities roughly follows this progression of complexity,

[2] "A strange game. The only winning move is not to play. How about a nice game of chess?" – WarGames, 1983.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

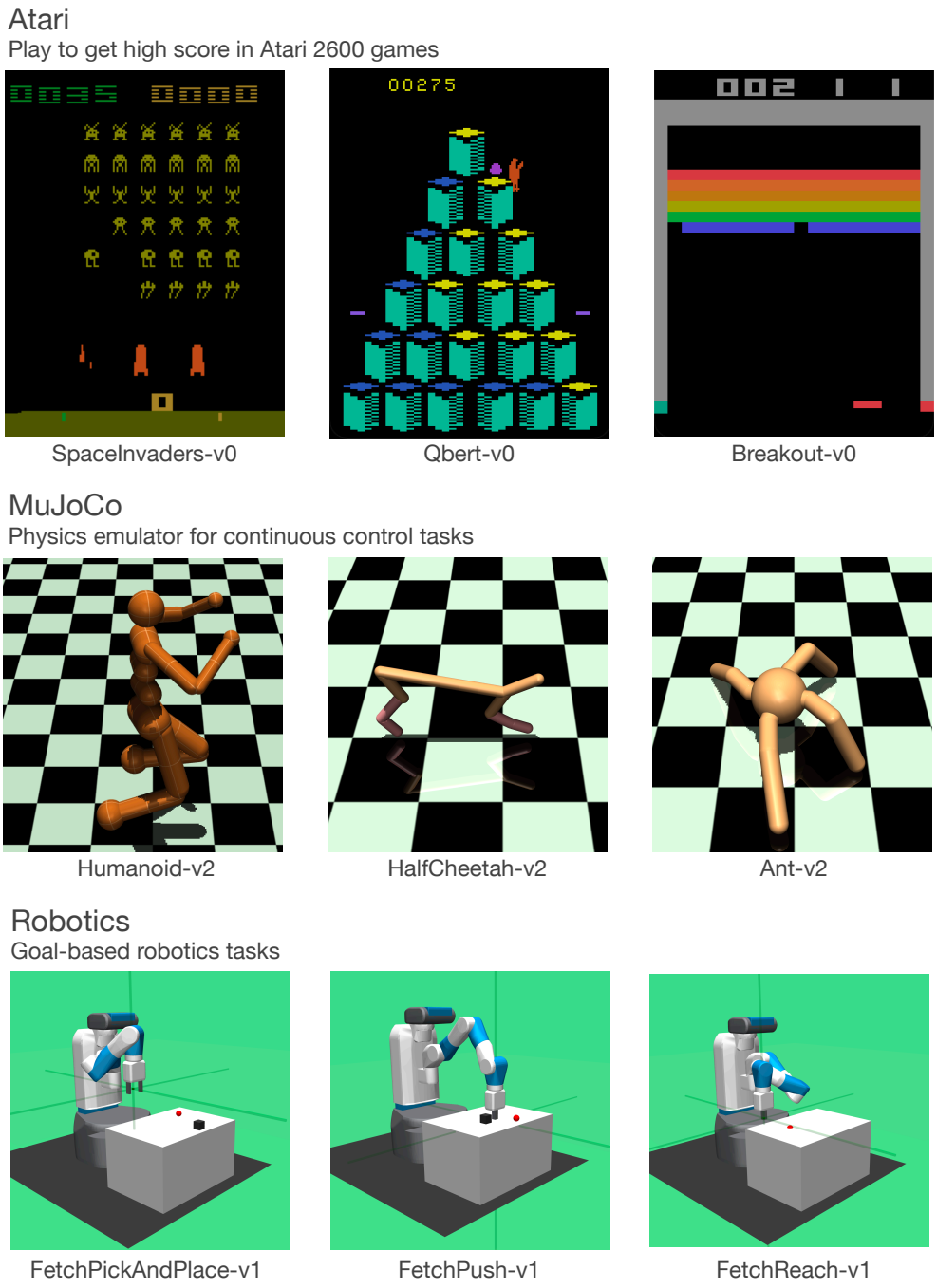


Figure 11.6: The OpenAI Gym [119] (`https://gym.openai.com`) provides a flexible simulation environment to test learning strategies. Examples include classic Atari 2600 video games and simulated rule-based control environments, including open-world physics [706] and robotics [769]. Other examples include classic control benchmarks.

with tic-tac-toe being essentially a homework exercise, checkers being the earliest real demonstration of RL by Arthur Samuel [616], and more complex games such as backgammon [700] and eventually chess and go [658, 661] following.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

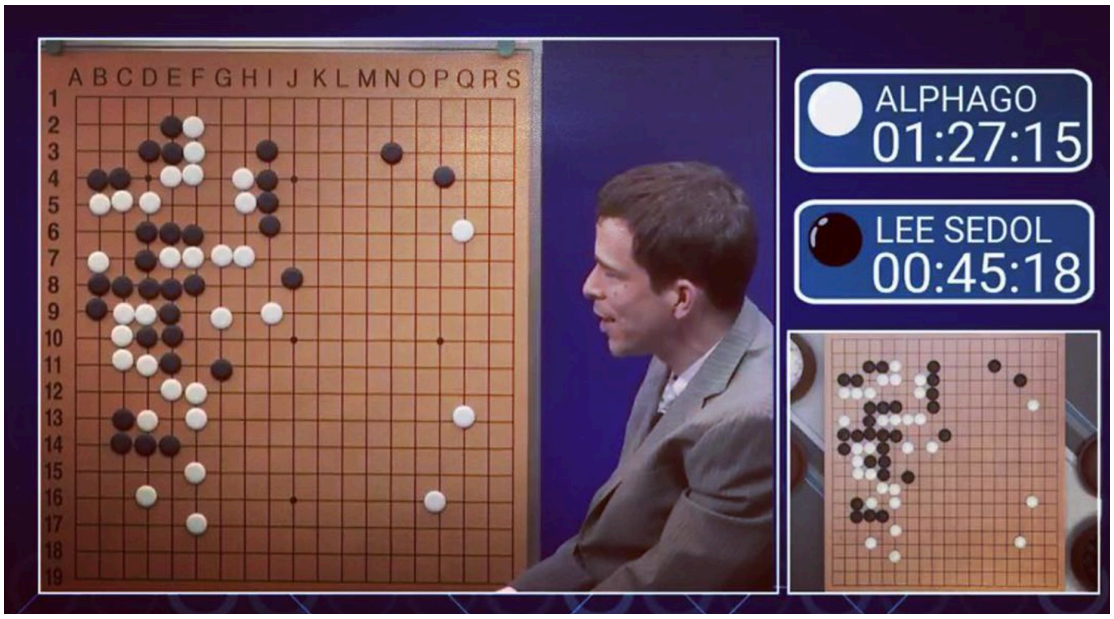


Figure 11.7: Reinforcement learning has demonstrated incredible performance in recent expert tasks, such as AlphaGo defeating world champion Lee Sedol in the game of go [659] on March 19, 2016. *Reproduced from https://www.flickr.com/photos/erikbenson/25717574115.*

Interestingly, about three decades passed between each of these definitive landmarks. One of the next major landmarks was a recent generalist RL agent that can learn to play multiple games [659], rather than specializing in only one task.

The success of DeepMind's AlphaGo and AlphaGo Zero demonstrates the remarkable power of modern RL. This system was a major breakthrough in RL research, learning to beat the Grandmaster Lee Sedol 4–1 in 2016, depicted in Fig. 11.7. However, AlphaGo relied heavily on reward shaping and expert guidance, making it a custom solution, rather than a generalized learner. Its successor, AlphaGo Zero, relied entirely on self-play, and was able to eventually defeat the original AlphaGo decisively. AlphaGo was based largely on CNNs, while AlphaGo Zero used a residual network (ResNet). ResNets are easier to train, and AlphaGo Zero was one of the first concrete success stories that cemented the ResNet as a competitive architecture. AlphaGo Zero was trained in 40 days on four tensor processing units, in contrast to many advanced ML algorithms that are trained for months on thousands of GPUs. Both AlphaGo and AlphaGo Zero are based on using deep learning to improve a Monte Carlo tree search.

Video Games

Some of the most impressive recent innovations in RL have involved scaling up to larger input spaces, which are well exemplified by the ability of RL to mas-

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

ter classic Atari video games [506]. In the case of Atari games, the pixel space is processed using a CNN architecture, with human-level performance being achieved mere years after the birth of modern deep learning for image classification [414]. More recently, RL has been demonstrated on more sophisticated games, such as StarCraft [735], which is a real-time strategy game; DeepMind's AlphaStar became a Grandmaster in 2019.

General artificial intelligence is one of the grand challenge problems in modern machine learning, whereby a learning agent is able to excel at multiple tasks, as in biological systems. What is perhaps most impressive about recent RL agents that learn video games is that the learning approach is *general*, so that the same RL framework can be used to learn multiple tasks. There is evidence that video games may improve performance in human surgeons [467, 607], and it may be that future RL agents will master both robotic manipulation and video games in a next stage of generalized AI.

Physical Systems

Although much of RL has been developed for board games and video games, it is increasingly being used for various advanced modeling and control tasks in physical systems. Physical systems, such as lasers [678] and fluids [581], often require additional considerations, such as continuous state and action spaces [589], and the need for certifiable solutions, such as trust regions [644], for safety-critical applications (e.g., transportation, autonomous flight, etc.).

There has been considerable work applying RL in the field of fluid dynamics [131] for fluid flow control [302, 565, 581, 582], for example for bluff-body control [241] and controlling Rayleigh–Bénard convection [66]. RL has also been applied to the related problem of navigation in a fluid environment [85, 180, 304], and more recently for turbulence modeling [530].

In addition to studying fluids, there is an extensive literature using RL to develop control policies for real and simulated robotic systems that operate primarily in a fluid environment, for example to learn how to fly and swim. For example, some of the earliest work has involved optimizing the flight of uninhabited aerial vehicles [1, 2, 385, 538, 591, 698, 773] with especially impressive helicopter aerobatics [2]. Controlling the motion of fish [268, 269, 532, 733] is another major area of development, including individual [268] and collective motion [269, 532, 733]. Gliding and perching is another large area of development [531, 590, 591].

Robotics and Autonomy

Robotics [300, 394] and autonomy [545, 592, 615, 652] are two of the largest areas of current research in RL. These both count as *physical systems*, as in the

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

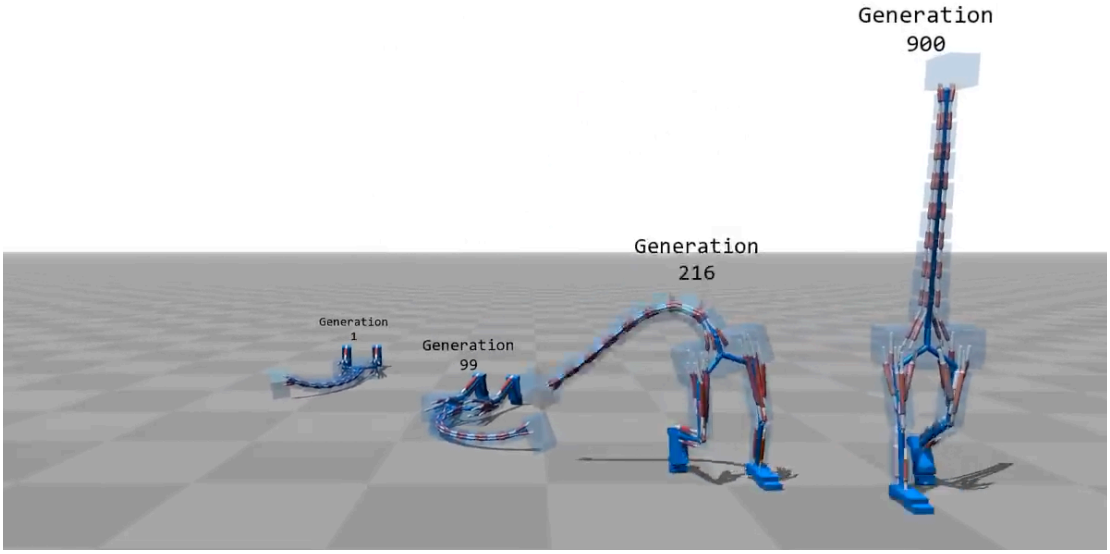


Figure 11.8: Illustration of improved bipedal locomotion performance with more generations of learning. Reproduced with permission from Geijtenbeek et al. [271].

section above, but deserve their own treatment, as these are major areas of innovation. In fact, both robotics and autonomy may be viewed as two of the most pressing societal applications of machine learning in general, and reinforcement learning in particular, with self-driving cars alone promising to reshape the modern transportation and energy landscape. As with the discussion of physical systems above, these are typically safety-critical applications with physical constraints [431, 696]. Figure 11.8 shows a virtual locomotion task that involves learning physics in a robot walker.

11.6 Optimal Nonlinear Control

Reinforcement learning has considerable overlap with optimal nonlinear control, and historically they were developed in parallel under the same optimization framework. Here we provide a brief overview of optimal nonlinear control theory, which will provide a connection between the classic linear control theory from Chapter 8 and dynamic programming to solve Bellman's equations used in this chapter. We have already seen optimal control in the context of linear dynamics and quadratic cost functions in Section 8.4, resulting in the linear–quadratic regulator (LQR). Similarly, we have used Bellman's equations to find optimal policies in RL for systems governed by MDPs. A major goal of this section is to provide a more general mathematical treatment of Bellman's equations, extending these approaches to fully nonlinear optimal control problems. However, this section is very technical and departs from the MDP notation

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

used throughout the rest of the chapter; it may be omitted on a first reading. For more details, see the excellent text by Stengel [675].

Hamilton–Jacobi–Bellman Equation

In optimal control, the goal is often to find a control input $\mathbf{u}(t)$ to drive a dynamical system,

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{x} = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t), t)\,, \tag{11.42}$$

to follow a trajectory $\mathbf{x}(t)$ that minimizes a cost function,

$$J(\mathbf{x}(t), \mathbf{u}(t), t_0, t_f) = Q(\mathbf{x}(t_f), t_f) + \int_{t_0}^{t_f} \mathcal{L}(\mathbf{x}(\tau), \mathbf{u}(\tau))\,\mathrm{d}\tau. \tag{11.43}$$

Note that this formulation in (11.43) generalizes the LQR cost function in (8.47); now the immediate cost function $\mathcal{L}(\mathbf{x}, \mathbf{u})$ and the terminal cost $Q(\mathbf{x}(t_f), t_f)$ may be non-quadratic functions. Often there are also constraints on the state $\mathbf{x}$ and control $\mathbf{u}$, which determine what solutions are admissible.

Given an initial state $\mathbf{x}_0 = \mathbf{x}(t_0)$ at t_0, an optimal control $\mathbf{u}(t)$ will result in an optimal cost function J. We may define a *value* function $V(\mathbf{x}, t_0, t_f)$ that describes the total integrated cost starting at this position $\mathbf{x}$ assuming the control law is optimal:

$$V(\mathbf{x}(t_0), t_0, t_f) = \min_{\mathbf{u}(t)} J(\mathbf{x}(t), \mathbf{u}(t), t_0, t_f), \tag{11.44}$$

where $\mathbf{x}(t)$ is the solution to (11.42) for the optimal $\mathbf{u}(t)$. Notice that the value function is no longer a function of the control $\mathbf{u}(t)$, as this has been optimized over, and it is also not a function of a trajectory $\mathbf{x}(t)$, but rather of an initial state $\mathbf{x}_0$, as the remainder of the trajectory is entirely specified by the dynamics and the optimal control law. The value function is often called the *cost-to-go* in control theory, as the value function evaluated at any point $\mathbf{x}(t)$ on an optimal trajectory will represent the remaining cost associated with continuing to enact this optimal policy until the final time t_f. In fact, this is a statement of Bellman's optimality principle, that the value function V remains optimal starting with any point on an optimal trajectory.

The Hamilton–Jacobi–Bellman[3] (HJB) equation establishes a partial differential equation that must be satisfied by the value function $V(\mathbf{x}(t), t, t_f)$ at every

[3]Kalman recognized that the Bellman optimal control formulation was a generalization of the Hamilton–Jacobi equation from classical mechanics to handle stochastic input–output systems. These formulations all involve the calculus of variations, which traces its roots back to the brachistochrone problem of Johann Bernoulli.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

intermediate time $t \in [t_0, t_f]$:

$$-\frac{\partial V}{\partial t} = \min_{\mathbf{u}(t)} \left(\left(\frac{\partial V}{\partial \mathbf{x}} \right)^T \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t)) + \mathcal{L}(\mathbf{x}(t), \mathbf{u}(t)) \right). \tag{11.45}$$

To derive the HJB equation, we may compute the total time derivative of the value function $V(\mathbf{x}(t), t, t_f)$ at some intermediate time t:

$$\frac{\mathrm{d}}{\mathrm{d}t} V(\mathbf{x}(t), t, t_f) = \frac{\partial V}{\partial t} + \left(\frac{\partial V}{\partial \mathbf{x}} \right)^T \frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} \tag{11.46a}$$

$$= \min_{\mathbf{u}(t)} \frac{\mathrm{d}}{\mathrm{d}t} \left(\int_t^{t_f} \mathcal{L}(\mathbf{x}(\tau), \mathbf{u}(\tau)) \, \mathrm{d}\tau + Q(\mathbf{x}(t_f), t_f) \right) \tag{11.46b}$$

$$= \min_{\mathbf{u}(t)} \left(\underbrace{\frac{\mathrm{d}}{\mathrm{d}t} \int_t^{t_f} \mathcal{L}(\mathbf{x}(\tau), \mathbf{u}(\tau)) \, \mathrm{d}\tau}_{-\mathcal{L}(\mathbf{x}(t), \mathbf{u}(t))} \right) \tag{11.46c}$$

$$\implies \quad -\frac{\partial V}{\partial t} = \min_{\mathbf{u}(t)} \left(\left(\frac{\partial V}{\partial \mathbf{x}} \right)^T \mathbf{f}(\mathbf{x}, \mathbf{u}) + \mathcal{L}(\mathbf{x}, \mathbf{u}) \right). \tag{11.46d}$$

Note that the terminal cost does not vary with t, so it has zero time derivative. The derivative of the integral of the instantaneous cost $\int_t^{t_f} \mathcal{L}(\mathbf{x}(\tau), \mathbf{u}(\tau)) \, \mathrm{d}\tau$ is equal to $-\mathcal{L}(\mathbf{x}(t), \mathbf{u}(t))$ by the first fundamental theorem of calculus. Finally, the term $(\partial V / \partial \mathbf{x})^T \mathbf{f}(\mathbf{x}, \mathbf{u})$ may be brought into the minimization argument, since V is already defined as the optimal cost over $\mathbf{u}$. The LQR optimal Riccati equation is a special case of the HJB equation, and the vector of partial derivatives in $(\partial J / \partial \mathbf{x})$ serves the same role as the Lagrange multiplier co-state $\boldsymbol{\lambda}$. The HJB equation may also be more intuitive in vector calculus notation

$$-\frac{\partial V}{\partial t} = \min_{\mathbf{u}(t)} (\nabla V \cdot \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t)) + \mathcal{L}(\mathbf{x}(t), \mathbf{u}(t))). \tag{11.47}$$

The HJB formulation above relies implicitly on Bellman's principle of optimality, namely that for any point on an optimal trajectory $\mathbf{x}(t)$, the value function V is still optimal for the remainder of the trajectory:

$$V(\mathbf{x}(t), t, t_f) = \min_{\mathbf{u}} \left(\int_t^{t_f} \mathcal{L}(\mathbf{x}(\tau), \mathbf{u}(\tau)) \, \mathrm{d}\tau + Q(\mathbf{x}(t_f), t_f) \right). \tag{11.48}$$

One outcome is that the value function can be decomposed as

$$V(\mathbf{x}(t_0), t_0, t_f) = V(\mathbf{x}(t_0), t_0, t) + V(\mathbf{x}(t), t, t_f). \tag{11.49}$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

This makes it possible to take the total time derivative above. A more rigorous derivation is possible using the calculus of variations.

The HJB equation is incredibly powerful, providing a PDE for the optimal solution of general nonlinear control problems. Typically, the HJB equation is solved numerically as a two-point boundary value problem, with boundary conditions $\mathbf{x}(0) = \mathbf{x}_0$ and $V(\mathbf{x}(t_f), t_f, t_f) = Q(\mathbf{x}(t_f), t_f)$, for example using a shooting method. However, a nonlinear control problem with a three-dimensional state vector $\mathbf{x} \in \mathbb{R}^3$ will result in a three-dimensional PDE. Thus, optimal nonlinear control based on the HJB equation typically suffers from the curse of dimensionality. Phase-space clustering techniques have shown great promise in reducing the effective state-space dimension for systems that evolve on a low-dimensional attractor [367].

Discrete-Time HJB and the Bellman Equation

Bellman's optimal control is especially intuitive for discrete-time systems, where, instead of optimizing over a function, we optimize over a discrete control sequence. Consider a discrete-time dynamical system

$$\mathbf{x}_{k+1} = \mathbf{F}(\mathbf{x}_k, \mathbf{u}_k). \tag{11.50}$$

The cost is now given by

$$J(\mathbf{x}_0, \{\mathbf{u}_k\}_{k=j}^n, j, n) = \sum_{k=j}^{n} \mathcal{L}(\mathbf{x}_k, \mathbf{u}_k) + Q(\mathbf{x}_n, t_n). \tag{11.51}$$

Similarly, the value function is defined as the value of the cumulative cost function, starting at a point $\mathbf{x}_0$ assuming an optimal control policy $\mathbf{u}$:

$$V(\mathbf{x}_0, 0, n) = \min_{\{\mathbf{u}_k\}_{k=0}^n} J(\mathbf{x}_0, \{\mathbf{u}_k\}_{k=0}^n, 0, n). \tag{11.52}$$

Again, Bellman's principle of optimality states that an optimal control policy has the property that, at any point along the optimal trajectory $\mathbf{x}(t)$, the remaining control policy is optimal with respect to this new initial state. Mathematically,

$$V(\mathbf{x}_0, 0, n) = V(\mathbf{x}_0, 0, k) + V(\mathbf{x}_k, k, n), \quad \forall\, k \in (0, n). \tag{11.53}$$

Thus, the value at an intermediate time-step k may be written as

$$V(\mathbf{x}_k, k, n) = \left(\min_{\mathbf{u}_k} \mathcal{L}(\mathbf{x}_k, \mathbf{u}_k) \right) + \underbrace{V(\mathbf{x}_{k+1}, k+1, n)}_{\text{s.t. } \mathbf{x}_{k+1} = \mathbf{F}(\mathbf{x}_k, \mathbf{u}_k)} \tag{11.54a}$$

$$= \min_{\mathbf{u}_k} \left(\mathcal{L}(\mathbf{x}_k, \mathbf{u}_k) + V(\mathbf{F}(\mathbf{x}_k, \mathbf{u}_k), k+1, n) \right). \tag{11.54b}$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

It is also possible, given a value function $V(\mathbf{x}_k, k, n)$, to determine the next optimal control action $\mathbf{u}_k$ by returning the $\mathbf{u}_k$ that minimizes the above expression. This defines an *optimal policy* $\mathbf{u} = \boldsymbol{\pi}(\mathbf{x})$. Dropping the functional dependence of V on the end time, we then have

$$V(\mathbf{x}) = \min_{\mathbf{u}}(\mathcal{L}(\mathbf{x}, \mathbf{u}) + V(\mathbf{F}(\mathbf{x}, \mathbf{u}))), \tag{11.55a}$$

$$\boldsymbol{\pi}(\mathbf{x}) = \operatorname*{argmin}_{\mathbf{u}}(\mathcal{L}(\mathbf{x}, \mathbf{u}) + V(\mathbf{F}(\mathbf{x}, \mathbf{u}))). \tag{11.55b}$$

These form the Bellman equations.

Note that we have explicitly included the terminal time t_f in the terminal cost $Q(\mathbf{x}_n, t_n)$ and $Q(\mathbf{x}(t_f), t_f)$, as there are situations when the arrival time should be minimized. However, it is also possible to include the time explicitly in the immediate cost $\mathcal{L}(\mathbf{x}, \mathbf{u}, t)$, for example to include a discount function $e^{-\gamma t}$ for future costs or rewards.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Suggested Reading

Texts

(1) Reinforcement learning: An introduction, by R. S. Sutton and A. G. Barto, 1998 [683].

Papers and reviews

(1) Q-learning, by C. Watkins and P. Dayan, *Machine Learning*, 1992 [746].

(2) TD(λ) converges with probability 1, by P. Dayan and T. J. Sejnowski, *Machine Learning*, 1994 [197].

(3) Human-level control through deep reinforcement learning, by V. Mnih et al., *Nature*, 2015 [506].

(4) Mastering the game of go without human knowledge, by D. Silver et al., *Nature*, 2017 [661].

(5) A tour of reinforcement learning: The view from continuous control, by B. Recht, *Annual Review of Control, Robotics, and Autonomous Systems*, 2019 [589].

Blogs and lectures

(1) Deep reinforcement learning: Pong from pixels, by A. Karpathy, `http://karpathy.github.io/2016/05/31/rl/`.

(2) Introduction to reinforcement learning with David Silver, by D. Silver, `www.youtube.com/playlist?list=PLqYmG7hTraZBiG_XpjnPrSNw-1XQaM_gB`

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Homework

Exercise 11-1. This example will explore reinforcement learning on the game of tic-tac-toe. First, describe the states, actions, and rewards.

Next, design a policy iteration algorithm to optimize the policy π. Begin with a randomly chosen policy. Plot the value function on the board and describe the optimal policy.

How many policy iterations are required before the policy and value function converge? How many games were played at each policy iteration? Is this consistent with what you would expect a human learning would do?

Is there any structure or symmetry in the game that could be used to improve the learning rate? Implement a policy iteration that exploits this structure, and determine how many policy iterations are required before converging and how many games played per policy iteration.

Exercise 11-2. Repeat the above example using value iteration instead of policy iteration. Compare the number of iterations in both methods, along with the total training time.

Exercise 11-3. This exercise will develop a reinforcement learning controller for the fluid flow past a cylinder. There are several open-source codes that can be used to simulate simple fluid flows, such as the IBPM code at `https://github.com/cwrowley/ibpm/`.

Use reinforcement learning to develop a control law to force the cylinder wake to be symmetric. Describe the reward structure and what learning framework you chose. Also plot your results, including learning rates, performance, etc. How long did it take to train this controller (i.e., how many computational iterations, how much CPU time, etc.)?

Now, assume that the RL agent only has access to the lift and drag coefficients, C_L and C_D. Design an RL scheme to track a given reference lift value, say $C_L = 1$ or $C_L = -1$. See if you can make your controller track a reference that switches between these values. What if the reference lift is much larger, say $C_L = 2$ or $C_L = 5$?

Exercise 11-4. Install the AI Gym API and develop an RL controller for the classic control example of a pendulum on a cart. Explore different RL strategies.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Chapter 12

Reduced-Order Models (ROMs)

The proper orthogonal decomposition (POD) is the SVD algorithm applied to partial differential equations (PDEs). As such, it is one of the most important dimensionality reduction techniques available to study complex, spatio-temporal systems. Such systems are typically exemplified by nonlinear PDEs that prescribe the evolution in time and space of the quantities of interest in a given physical, engineering, and/or biological system. The success of the POD is related to the seemingly ubiquitous observation that, in most complex systems, meaningful behaviors are encoded in low-dimensional patterns of dynamic activity. The POD technique seeks to take advantage of this fact in order to produce low-rank dynamical systems capable of accurately modeling the full spatio-temporal evolution of the governing complex system. Specifically, *reduced-order models* (ROMs) leverage POD modes for projecting PDE dynamics to low-rank subspaces where simulations of the governing PDE model can be more readily evaluated. Importantly, the low-rank models produced by the ROM allow for significant improvements in computational speed, potentially enabling prohibitively expensive Monte Carlo simulations of PDE systems, optimization over parameterized PDE systems, and/or real-time control of PDE-based systems. POD has been extensively used in the fluid dynamics community [335]. It has also found a wide variety of applications in structural mechanics and vibrational analysis [31, 311, 382, 437], optical and micro-electromechanical systems (MEMS) technologies [444, 657], atmospheric sciences (where it is called empirical orthogonal functions (EOFs)) [158, 159], wind engineering applications [667], acoustics [243], and neuroscience [45, 379, 703]. The success of the method relies on its ability to provide physically interpretable spatio-temporal decompositions of data [79, 170, 243, 381, 420, 444].

12.1 Proper Orthogonal Decomposition (POD) for Partial Differential Equations

Throughout the engineering, physical, and biological sciences, many systems are known to have prescribed relationships between time and space that drive patterns of dynamical activity. Even simple spatio-temporal relationships can lead to highly complex, yet coherent, dynamics that motivate the main thrust of analytic and computational studies. Modeling efforts seek to derive these spatio-temporal relationships either through first-principles laws or through well-reasoned conjectures about existing relationships, thus leading generally to an underlying partial differential equation (PDE) that constrains and governs the complex system. Typically, such PDEs are beyond our ability to solve analytically. As a result, two primary solution strategies are pursued: computation and/or asymptotic reduction. In the former, the complex system is discretized in space and time to artificially produce an extremely high-dimensional system of equations which can be solved to a desired level of accuracy, with higher accuracy requiring a larger dimension of the discretized system. In this technique, the high-dimensionality is artificial and simply a consequence of the underlying numerical solution scheme. In contrast, asymptotic reduction seeks to replace the complex system with a simpler set of equations, preferably that are *linear* so as to be amenable to analysis. Before the 1960s and the rise of computation, such asymptotic reductions formed the backbone of applied mathematics in fields such a fluid dynamics. Indeed, asymptotics form the basis of the earliest efforts of dimensionality reduction. Asymptotic methods are not covered in this book, but the computational methods that enable reduced-order models are.

To be more mathematically precise about our study of complex systems, we consider generically a system of nonlinear PDEs of a single spatial variable that can be modeled as

$$\mathbf{u}_t = \mathbf{N}(\mathbf{u}, \mathbf{u}_x, \mathbf{u}_{xx}, \ldots, x, t; \boldsymbol{\beta}) \tag{12.1}$$

where the subscripts denote partial differentiation and $\mathbf{N}(\cdot)$ prescribes the generically nonlinear evolution. The parameter $\boldsymbol{\beta}$ will represent a bifurcation parameter for our later considerations. Further, associated with (12.1) are a set of initial and boundary conditions on a domain $x \in [-L, L]$. Historically, a number of analytic solution techniques have been devised to study (12.1). Typically the aim of such methods is to reduce the PDE (12.1) to a set of ordinary differential equations (ODEs). The standard PDE methods of *separation of variables* and *similarity solutions* are constructed for this express purpose. Once in the form of an ODE, a broader variety of analytic methods can be applied along with a *qualitative theory* in the case of nonlinear behavior [334]. This again highlights the role that *asymptotics* can play in characterizing behavior.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Although a number of potential solution strategies have been mentioned, (12.1) does not admit a closed-form solution in general. Even the simplest nonlinearity or a spatially dependent coefficient can render the standard analytic solution strategies useless. However, computational strategies for solving (12.1) are abundant and have provided transformative insights across the physical, engineering, and biological sciences. The various computational techniques devised lead to an approximate numerical solution of (12.1), which is of high dimension. Consider, for instance, a standard spatial discretization of (12.1) whereby the spatial variable x is evaluated at $n \gg 1$ points,

$$\mathbf{u}(x_k, t) \quad \text{for} \quad k = 1, 2, \ldots, n, \tag{12.2}$$

with spacing $\Delta x = x_{k+1} - x_k = 2L/n$. Using standard finite-difference formulas, spatial derivatives can be evaluated using neighboring spatial points so that, for instance,

$$\mathbf{u}_x = \frac{\mathbf{u}(x_{k+1}, t) - \mathbf{u}(x_{k-1}, t)}{2\Delta x}, \tag{12.3a}$$

$$\mathbf{u}_{xx} = \frac{\mathbf{u}(x_{k+1}, t) - 2\mathbf{u}(x_k, t) + \mathbf{u}(x_{k-1}, t)}{\Delta x^2}. \tag{12.3b}$$

Such spatial discretization transforms the governing PDE (12.1) into a set of n ODEs:

$$\frac{\mathrm{d}\mathbf{u}_k}{\mathrm{d}t} = \mathbf{N}(\mathbf{u}(x_{k+1}, t), \mathbf{u}(x_k, t), \mathbf{u}(x_{k-1}, t), \ldots, x_k, t; \boldsymbol{\beta}), \quad k = 1, 2, \ldots, n. \tag{12.4}$$

This process of discretization produces a more manageable system of equations at the expense of rendering (12.1) high-dimensional. It should be noted that, as accuracy requirements become more stringent, the resulting dimension n of the system (12.4) also increases, since $\Delta x = 2L/n$. Thus, the dimension of the underlying computational scheme is artificially determined by the accuracy of the finite-difference differentiation schemes.

The spatial discretization of (12.1) illustrates how high-dimensional systems are rendered. The artificial production of high-dimensional systems is ubiquitous across computational schemes and presents significant challenges for scientific computing efforts. To further illustrate this phenomenon, we consider a second computational scheme for solving (12.1). In particular, we consider the most common technique for analytically solving PDEs: separation of variables. In this method, a solution is assumed, whereby space and time are independent, so that

$$\mathbf{u}(x, t) = \mathbf{a}(t)\psi(x), \tag{12.5}$$

where the variable $\mathbf{a}(t)$ subsumes all the time dependence of (12.1) and $\psi(x)$ characterizes the spatial dependence. Separation of variables is only guaranteed to work analytically if (12.1) is linear with constant coefficients. In that

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

restrictive case, two differential equations can be derived that separately characterize the spatial and temporal dependences of the complex system. The differential equations are related by a constant parameter that is present in each.

For the general form of (12.1), separation of variables can be used to yield a computational algorithm capable of producing accurate solutions. Since the spatial solutions are not known *a priori*, it is typical to assume a set of basis modes which are used to construct $\psi(x)$. Indeed, such assumptions on basis modes underlie the critical ideas of the method of *eigenfunction expansions*. This yields a separation-of-variables solution ansatz of the form

$$\mathbf{u}(x,t) = \sum_{k=1}^{n} \mathbf{a}_k(t)\psi_k(x), \tag{12.6}$$

where $\psi_k(x)$ form a set of $n \gg 1$ basis modes. As before, this expansion artificially renders a high-dimensional system of equations since n modes are required. This separation-of-variables solution approximates the true solution, provided n is large enough. Increasing the number of modes n is equivalent to increasing the spatial discretization in a finite-difference scheme.

The orthogonality properties of the basis functions $\psi_k(x)$ enable us to make use of (12.6). To illustrate this, consider a scalar version of (12.1) with the associated scalar separable solution $u(x,t) = \sum_{k=1}^{n} a_k(t)\psi_k(x)$. Inserting this solution into the governing equations gives

$$\sum \psi_k \frac{\mathrm{d}a_k}{\mathrm{d}t} = \mathbf{N}\left(\sum a_k\psi_k, \sum a_k(\psi_k)_x, \sum a_k(\psi_k)_{xx}, \ldots, x, t; \boldsymbol{\beta}\right), \tag{12.7}$$

where the sums are from $k = 1, 2, \ldots, n$. Orthogonality of our basis functions implies that

$$\langle \psi_k, \psi_j \rangle = \delta_{kj} = \begin{cases} 0 & j \neq k, \\ 1 & j = k, \end{cases} \tag{12.8}$$

where δ_{kj} is the Kronecker delta function and $\langle \psi_k, \psi_j \rangle$ is the inner product defined as

$$\langle \psi_k, \psi_j \rangle = \int_{-L}^{L} \psi_k \psi_j^* \, \mathrm{d}x, \tag{12.9}$$

where * denotes complex conjugation.

Once the modal basis is decided on, the governing equations for the $a_k(t)$ can be determined by multiplying (12.7) by $\psi_j(x)$ and integrating from $x \in [-L, L]$. Orthogonality then results in the temporal governing equations, or Galerkin projected dynamics, for each mode

$$\frac{\mathrm{d}a_k}{\mathrm{d}t} = \left\langle \mathbf{N}\left(\sum a_j\psi_j, \sum a_j(\psi_j)_x, \sum a_j(\psi_j)_{xx}, \ldots, x, t; \boldsymbol{\beta}\right), \psi_k \right\rangle \quad \text{for} \quad k = 1, 2, \ldots, n. \tag{12.10}$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

The given form of $\mathbf{N}(\cdot)$ determines the mode coupling that occurs between the various n modes. Indeed, the hallmark feature of nonlinearity is the production of modal mixing from (12.10).

Numerical schemes based on the Galerkin projection (12.10) are commonly used to perform simulations of the full governing system (12.1). Convergence to the true solution can be accomplished by judicious choice of both the modal basis elements ψ_k as well as the total number of modes n. Interestingly, the separation-of-variables strategy, which is rooted in *linear* PDEs, works for *nonlinear* and *non-constant-coefficient* PDEs, provided enough modal basis functions are chosen in order to accommodate all the nonlinear mode mixing that occurs in (12.10). A good choice of modal basis elements allows for a smaller set of n modes to be chosen to achieve a desired accuracy. The POD method is designed to specifically address the data-driven selection of a set of basis modes that are tailored to the particular dynamics, geometry, and parameters.

Fourier Mode Expansion

The most prolific basis used for the Galerkin projection technique is Fourier modes. More precisely, the fast Fourier transform (FFT) and its variants have dominated scientific computing applied to the engineering, physical, and biological sciences. There are two primary reasons for this: (1) there is a strong intuition developed around the meaning of Fourier modes as it directly relates to spatial wavelengths and frequencies, and, more importantly, (2) the algorithm necessary to compute the right-hand side of (12.10) can be executed in $O(n \log n)$ operations. The second fact has made the FFT one of the top 10 algorithms of the last century and a foundational cornerstone of scientific computing.

The Fourier mode basis elements are given by

$$\psi_k(x) = \frac{1}{L} \exp\left(i \frac{2\pi k x}{L}\right) \quad \text{for} \quad x \in [0, L] \quad \text{and} \quad k = -n/2, \ldots, -1, 0, 1, \ldots, n/2 - 1. \tag{12.11}$$

It should be noted that in most software packages, including MATLAB, the FFT command assumes that the spatial interval is $x \in [0, 2\pi]$. Thus one must rescale a domain of length L to 2π before using the FFT.

Obviously the Fourier modes (12.11) are complex periodic functions on the interval $x \in [0, L]$. However, they are applicable to a much broader class of functions that are not necessarily periodic. For instance, consider a localized Gaussian function

$$u(x, t) = \exp(-\sigma x^2) \tag{12.12}$$

whose Fourier transform is also a Gaussian. In representing such a function with Fourier modes, a large number of modes are often required since the func-

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

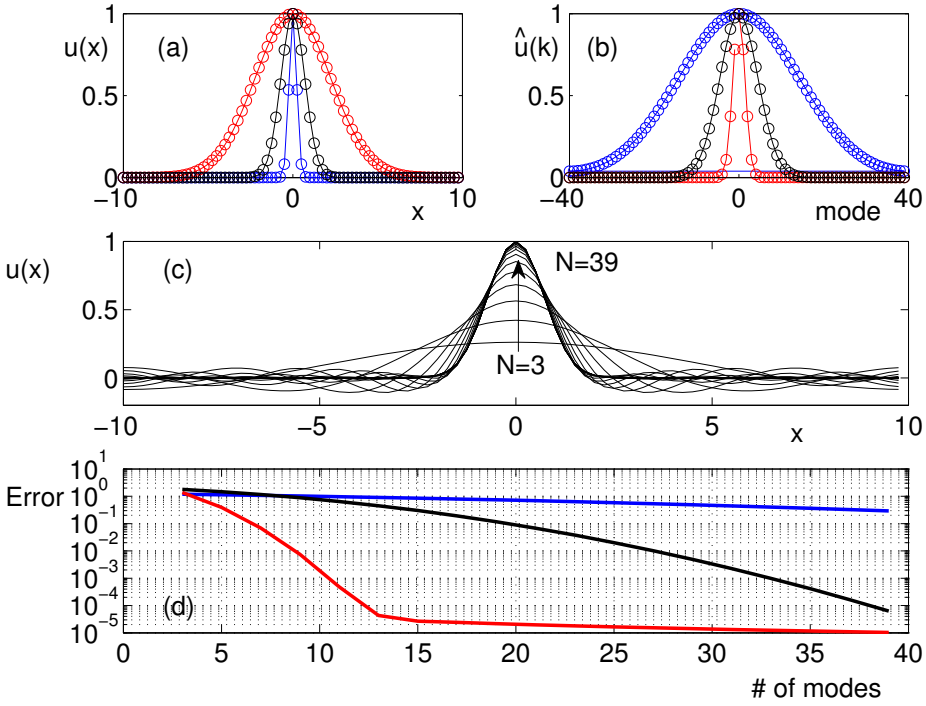


Figure 12.1: Illustration of Fourier modes for representing a localized Gaussian pulse. (a) Here $n = 80$ Fourier modes are used to represent the Gaussian $u(x) = \exp(-\sigma x^2)$ in the domain $x \in [-10, 10]$ for $\sigma = 0.1$ (red), $\sigma = 1$ (black), and $\sigma = 10$ (blue). (b) The Fourier mode representation of the Gaussian, showing the modes required for an accurate representation of the localized function. (c) The convergence of the n-mode solution to the actual Gaussian ($\sigma = 1$); with (d) the L^2 error from the true solution for the three values of σ.

tion itself is not periodic. Figure 12.1 shows the Fourier mode representation of the Gaussian for three values of σ. Of note is the fact that a large number of modes are required to represent this simple function, especially as the Gaussian width is decreased. Although the FFT algorithm is extremely fast and widely applied, one can see immediately that a large number of modes are generically required to represent simple functions of interest. Thus, solving problems using the FFT often requires high-dimensional representations (i.e., $n \gg 1$) to accommodate generic, localized spatial behaviors. Ultimately, our aim is to move away from artificially creating such high-dimensional problems.

Special Functions and Sturm–Liouville Theory

In the 1800s and early 1900s, mathematical physics developed many of the governing principles behind heat flow, electromagnetism, and quantum mechan-

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

ics, for instance. Many of the hallmark problems considered were driven by *linear* dynamics, allowing for analytically tractable solutions. And since these problems arose before the advent of computing, nonlinearities were typically treated as perturbations to an underlying linear equation. Thus one often considered complex systems of the form

$$\mathbf{u}_t = \mathbf{L}\mathbf{u} + \epsilon\mathbf{N}(\mathbf{u}, \mathbf{u}_x, \mathbf{u}_{xx}, \ldots, x, t; \boldsymbol{\beta}), \tag{12.13}$$

where $\mathbf{L}$ is a linear operator and $\epsilon \ll 1$ is a small parameter used for perturbation calculations. Often in mathematical physics, the operator $\mathbf{L}$ is a Sturm–Liouville operator, which guarantees many advantageous properties of the eigenvalues and eigenfunctions.

To solve equations of the form in (12.13), special modes are often used that are ideally suited for the problem. Such modes are eigenfunctions of the underlying linear operator $\mathbf{L}$ in (12.13):

$$\mathbf{L}\psi_k = \lambda_k\psi_k, \tag{12.14}$$

where $\psi_k(x)$ are orthonormal eigenfunctions of the operator $\mathbf{L}$. The eigenfunctions allow for an *eigenfunction expansion* solution whereby $\mathbf{u}(x,t) = \sum a_k(t)\psi_k(x)$. This leads to the following solution form:

$$\frac{\mathrm{d}a_k}{\mathrm{d}t} = \langle \mathbf{L}\mathbf{u}, \psi_k \rangle + \epsilon\langle \mathbf{N}, \psi_k \rangle. \tag{12.15}$$

The key idea in such an expansion is that the eigenfunctions presumably are ideal for modeling the spatial variations particular to the problem under consideration. Thus, they would seem to be ideal, or perfectly suited, modes for (12.13). This is in contrast to the Fourier mode expansion, as the sinusoidal modes may be unrelated to the particular physics or symmetries in the geometry. For example, the Gaussian example considered can be potentially represented more efficiently by Gauss–Hermite polynomials. Indeed, the wide variety of special functions, including the Sturm–Liouville operators of *Bessel*, *Laguerre*, *Hermite*, and *Legendre*, for instance, are aimed at making the representation of solutions more efficient and much more closely related to the underlying physics and geometry. Ultimately, one can think of using such functions as a way of doing *dimensionality reduction* by using an ideally suited set of basis functions.

Dimensionality Reduction

The examples above and solution methods for PDEs illustrate a common problem of scientific computing: the generation of n-degree, high-dimensional systems. For many complex PDEs with several spatial dimensions, it is not uncommon for discretization or modal expansion techniques to yield systems of differential equations with millions or billions of degrees of freedom. Such large

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

systems are extremely demanding for even the latest computational architectures, limiting accuracies and run-times in the modeling of many complex systems, such as high-Reynolds-number fluid flows.

To aid in computation, the selection of a set of optimal basis modes is critical, as it can greatly reduce the number of differential equations generated. Many solution techniques involve the solution of a linear system of size n, which generically involves $O(n^3)$ operations. Thus, reducing n is of paramount importance. One can already see that, even in the 1800s and early 1900s, the special functions developed for various problems of mathematical physics were an analytic attempt to generate an ideal set of modes for representing the dynamics of the complex system. However, for strongly nonlinear, complex systems (12.1), even such special functions rarely give the best set of modes. In the next section, we show how one might generate modes ψ_k that are tailored specifically for the dynamics and geometry in (12.1). Based on the SVD algorithm, the *proper orthogonal decomposition* (POD) generates a set of modes that are *optimal* for representing either simulation or measurement data, potentially allowing for significant reduction of the number of modes n required to model the behavior of (12.1) for a given accuracy [79, 737, 738].

12.2 Optimal Basis Elements: the POD Expansion

As illustrated in the previous section, the selection of a good modal basis for solving (12.1) using the Galerkin expansion in (12.6) is critical for efficient scientific computing strategies. Many algorithms for solving PDEs rely on choosing basis modes *a priori* based on (i) computational speed, (ii) accuracy, and/or (iii) constraints on boundary conditions. All these reasons are justified and form the basis of computationally sound methods. However, our primary concern in this chapter is in selecting a method that allows for maximal computational efficiency via *dimensionality reduction*. As already highlighted, many algorithms generate artificially large systems of size n. In what follows, we present a data-driven strategy, whereby optimal modes, also known as POD modes, are selected from numerical and/or experimental observations, thus allowing for a minimal number of modes $r \ll n$ to characterize the dynamics of (12.1).

Two options exist for extracting the optimal basis modes from a given complex system. Either one can collect data directly from an experiment, or one can simulate the complex system and sample the state of the system as it evolves according to the dynamics. In both cases, snapshots of the dynamics are taken and optimal modes identified. In the case when the system is simulated to extract modes, one can argue that no computational savings are achieved. However, much like the LU decomposition, which has an initial one-time computational cost of $O(n^3)$ before further $O(n^2)$ operations can be applied, the costly modal

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

extraction process is performed only once. The optimal modes can then be used in a computationally efficient manner thereafter.

To proceed with the construction of the optimal POD modes, the dynamics of (12.1) are sampled at some prescribed time interval. In particular, a snapshot $\mathbf{u}_k$ consists of samples of the complex system, with subscript k indicating sampling at time t_k, i.e., $\mathbf{u}_k := \begin{bmatrix}\mathbf{u}(x_1,t_k) & \mathbf{u}(x_2,t_k) & \cdots & \mathbf{u}(x_n,t_k)\end{bmatrix}^T$. Now, the continuous functions and modes will be evaluated at n discrete spatial locations, resulting in a high-dimensional vector representation; these will be denoted by bold symbols. We are generally interested in analyzing the computationally or experimentally generated large data set $\mathbf{X}$:

$$\mathbf{X} = \begin{bmatrix} | & | & & | \\ \mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_m \\ | & | & & | \end{bmatrix}, \tag{12.16}$$

where the columns $\mathbf{u}_k = \mathbf{u}(t_k) \in \mathbb{C}^n$ may be measurements from simulations or experiments. Matrix $\mathbf{X}$ consists of a *time series* of data, with m distinct measurement instants in time. Often the *state dimension* n is very large, on the order of millions or billions in the case of fluid systems. Typically $n \gg m$, resulting in a *tall-skinny* matrix, as opposed to a *short-fat* matrix when $n \ll m$.

As discussed previously, the singular value decomposition (SVD) provides a unique matrix decomposition for any complex-valued matrix $\mathbf{X} \in \mathbb{C}^{n\times m}$:

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^*, \tag{12.17}$$

where $\mathbf{U} \in \mathbb{C}^{n\times n}$ and $\mathbf{V} \in \mathbb{C}^{m\times m}$ are *unitary* matrices, and $\mathbf{\Sigma} \in \mathbb{C}^{n\times m}$ is a matrix with non-negative entries on the diagonal. Here * denotes the complex conjugate transpose. The columns of $\mathbf{U}$ are called *left singular vectors* of $\mathbf{X}$ and the columns of $\mathbf{V}$ are *right singular vectors*. The diagonal elements of $\mathbf{\Sigma}$ are called *singular values* and they are ordered from largest to smallest. The SVD provides critical insight into building an optimal basis set tailored to the specific problem. In particular, the matrix $\mathbf{U}$ is guaranteed to provide the best set of modes to approximate $\mathbf{X}$ in an ℓ_2 sense. Specifically, the columns of this matrix contain the orthogonal modes necessary to form the ideal basis. The matrix $\mathbf{V}$ gives the time history of each of the modal elements, and the diagonal matrix $\mathbf{\Sigma}$ is the weighting of each mode relative to the others. Recall that the modes are arranged with the most dominant first and the least dominant last.

The total number of modes generated is typically determined by the number of snapshots m taken in constructing $\mathbf{X}$ (where normally $n \gg m$). Our objective is to determine the minimal number of modes necessary to accurately represent the dynamics of (12.1) with a Galerkin projection (12.6). Thus we are interested in a rank-r approximation to the true dynamics where typically $r \ll m$. The quantity of interest is then the low-rank decomposition of the SVD

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

given by

$$\tilde{\mathbf{X}} = \tilde{\mathbf{U}}\tilde{\mathbf{\Sigma}}\tilde{\mathbf{V}}^*, \tag{12.18}$$

where $\|\mathbf{X}-\tilde{\mathbf{X}}\| < \epsilon$ for a given small value of ϵ. This low-rank truncation allows us to construct the modes of interest $\boldsymbol{\psi}_k$ from the columns of the truncated matrix $\tilde{\mathbf{U}}$. In particular, the optimal basis modes are given by

$$\tilde{\mathbf{U}} = \mathbf{\Psi} = \begin{bmatrix} | & | & & | \\ \boldsymbol{\psi}_1 & \boldsymbol{\psi}_2 & \cdots & \boldsymbol{\psi}_r \\ | & | & & | \end{bmatrix}, \tag{12.19}$$

where the truncation preserves the r most dominant modes used in (12.6). The truncated r modes $\{\boldsymbol{\psi}_1, \boldsymbol{\psi}_2, \ldots, \boldsymbol{\psi}_r\}$ are then used as the low-rank, orthogonal basis to represent the dynamics of (12.1).

The above snapshot-based method for extracting the low-rank, r-dimensional subspace of dynamic evolution associated with (12.1) is a data-driven computational architecture. Indeed, it provides an equation-free method, i.e., the governing equation (12.1) may actually be unknown. In the event that the underlying dynamics are unknown, then the extraction of the low-rank space allows one to build potential models in an r-dimensional subspace as opposed to remaining in a high-dimensional space where $n \gg r$. These ideas will be explored further in what follows. However, it suffices to highlight at this juncture that an optimal basis representation does not require an underlying knowledge of the complex system (12.1).

Galerkin Projection onto POD Modes

It is possible to approximate the state $\mathbf{u}$ of the PDE using a Galerkin expansion:

$$\mathbf{u}(t) \approx \mathbf{\Psi}\mathbf{a}(t), \tag{12.20}$$

where $\mathbf{a}(t) \in \mathbb{R}^r$ is the time-dependent coefficient vector and $r \ll n$. Plugging this modal expansion into the governing equation (12.13) and applying orthogonality (multiplying by $\mathbf{\Psi}^T$) gives the dimensionally reduced evolution

$$\frac{\mathrm{d}\mathbf{a}(t)}{\mathrm{d}t} = \mathbf{\Psi}^T\mathbf{L}\mathbf{\Psi}\mathbf{a}(t) + \mathbf{\Psi}^T\mathbf{N}(\mathbf{\Psi}\mathbf{a}(t), \boldsymbol{\beta}). \tag{12.21}$$

By solving this system of much smaller dimension, the solution of a high-dimensional nonlinear dynamical system can be approximated. Of critical importance is evaluating the nonlinear terms in an efficient way using the gappy POD or discrete empirical interpolation method (DEIM) mathematical architecture in Chapter 13. Otherwise, the evaluation of the nonlinear terms still requires calculation of functions and inner products with the original dimension n. In certain cases, such as the quadratic nonlinearity of Navier–Stokes, the

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

nonlinear terms can be computed once in an offline manner. However, parameterized systems generally require repeated evaluation of the nonlinear terms as the POD modes change with $\boldsymbol{\beta}$.

Example: the Harmonic Oscillator

To illustrate the POD method for selecting optimal basis elements, we will consider a classic problem of mathematical physics: the *quantum harmonic oscillator*. Although the ideal basis functions (Gauss–Hermite functions) for this problem are already known, we would like to infer these special functions in a purely data-driven way. In other words, can we deduce these special functions from snapshots of the dynamics alone? The standard harmonic oscillator arises in the study of spring–mass systems. In particular, one often assumes that the restoring force F of a spring is governed by the linear Hooke's law:

$$F(t) = -kx, \tag{12.22}$$

where k is the spring constant and $x(t)$ represents the displacement of the spring from its equilibrium position. Such a force gives rise to a potential energy for the spring of the form $V = kx^2/2$.

In considering quantum mechanical systems, such a restoring force (with $k = 1$ without loss of generality) and associated potential energy give rise to the Schrödinger equation with a parabolic potential,

$$iu_t + \frac{1}{2}u_{xx} - \frac{x^2}{2}u = 0, \tag{12.23}$$

where the second term in the partial differential equation represents the kinetic energy of a quantum particle while the last term is the parabolic potential associated with the linear restoring force.

The solution for the quantum harmonic oscillator can be easily computed in terms of special functions. In particular, by assuming a solution of the form

$$u(x,t) = a_k\psi_k(x)\exp[-i(k+\tfrac{1}{2})t], \tag{12.24}$$

with a_k determined from initial conditions, one finds the following boundary value problem for the eigenmodes of the system:

$$\frac{\mathrm{d}^2\psi_k}{\mathrm{d}x^2} + (2k+1-x^2)\psi_k, \tag{12.25}$$

with the boundary conditions $\psi_k \to 0$ as $x \to \pm\infty$. Normalized solutions to this equation can be expressed in terms of *Hermite polynomials*, $H_k(x)$, or the Gaussian–Hermite functions,

$$\psi_k = (2^k k!\sqrt{\pi})^{-1/2}\exp(-x^2/2)H_k(x) \tag{12.26a}$$

$$= (-1)^k(2^k k!\sqrt{\pi})^{-1/2}\exp(-x^2/2)\frac{\mathrm{d}^k}{\mathrm{d}x^k}\exp(-x^2). \tag{12.26b}$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

The Gauss–Hermite functions are typically thought of as the optimal basis functions for the harmonic oscillator, as they naturally represent the underlying dynamics driven by the Schrödinger equation with parabolic potential. Indeed, solutions of the complex system (12.23) can be represented as the sum

$$u(x,t) = \sum_{k=0}^{\infty} a_k (2^k k! \sqrt{\pi})^{-1/2} \exp(-x^2/2) H_k(x) \exp[-i(k+\tfrac{1}{2})t]. \tag{12.27}$$

Such a solution strategy is ubiquitous in mathematical physics, as is evidenced by the large number of special functions, often of Sturm–Liouville form, for different geometries and boundary conditions. These include Bessel functions, Laguerre polynomials, Legendre polynomials, parabolic cylinder functions, spherical harmonics, etc.

A numerical solution to the governing PDE (12.23) based on the fast Fourier transform is easy to implement [420]. The following code executes a full numerical solution with the initial conditions $u(x,0) = \exp(-0.2(x-x_0)^2)$, which is a Gaussian pulse centered at $x = x_0$. This initial condition generically excites a number of Gauss–Hermite functions. In particular, the initial projection onto the eigenmodes is computed from the orthogonality conditions so that

$$a_k = \langle u(x,0), \psi_k \rangle. \tag{12.28}$$

This inner product projects the initial condition onto each mode ψ_k.

Code 12.1: [MATLAB] Harmonic oscillator code.

```
L=30; n=512; x2=linspace(-L/2,L/2,n+1); x=x2(1:n); % spatial
    discretization
k=(2*pi/L)*[0:n/2-1 -n/2:-1].';     % wavenumbers for FFT
V=x.^2.';                 % potential
t=0:0.2:20;               % time domain collection points

u=exp(-0.2*(x-1).^2); % initial conditions
ut=fft(u);                % FFT initial data
[t,utsol]=ode45('pod_harm_rhs',t,ut,[],k,V); % integrate PDE
for j=1:length(t)
  usol(j,:)=ifft(utsol(j,:));       % transforming back
end
```

Code 12.1: [Python] Harmonic oscillator code.

```
u = np.exp(-0.2*np.power(x-1,2)) # initial conditions
ut = np.fft.fft(u) # FFT initial data
ut_split = np.concatenate((np.real(ut),np.imag(ut)))

utsol_split = integrate.odeint(harm_rhs,ut_split,t,mxstep
   =10**6)
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

```
utsol = utsol_split[:,:n] + (1j)*utsol_split[:,n:]

usol = np.zeros_like(utsol)
for jj in range(len(t)):
    usol[jj,:] = np.fft.ifft(utsol[jj,:])
```

The right-hand side function **pod_harm_rhs.m** associated with the above code contains the governing equation (12.23) in a three-line MATLAB code:

Code 12.2: [MATLAB] Harmonic oscillator right-hand side.

```
function rhs=pod_harm_rhs(t,ut,dummy,k,V)
u=ifft(ut);
rhs=-(i/2)*(k.^2).*ut - 0.5*i*fft(V.*u);
```

Code 12.2: [Python] Harmonic oscillator right-hand side.

```
def harm_rhs(ut_split,t,k=k,V=V,n=n):
    ut = ut_split[:n] + (1j)*ut_split[n:]
    u = np.fft.ifft(ut)
    rhs = -0.5*(1j)*np.power(k,2)*ut - 0.5*(1j)*np.fft.fft(V
        *u)
    rhs_split = np.concatenate((np.real(rhs),np.imag(rhs)))
    return rhs_split
```

The two codes together produce dynamics associated with the quantum harmonic oscillator. Figure 12.2 shows the dynamical evolution of an initial Gaussian $u(x,0) = \exp(-0.2(x-x_0)^2)$ with $x_0 = 0$ (top left) and $x_0 = 1$ (top right). From the simulation, one can see that there are a total of 101 snapshots (the initial condition and an additional 100 measurement times). These snapshots can be organized as in (12.16) and the singular value decomposition performed. The singular values of the decomposition are suggestive of the underlying dimensionality of the dynamics. For the dynamical evolution observed in the top panels of Fig. 12.2, the corresponding singular values of the snapshots are given in the bottom panels. For the symmetric initial condition (symmetric about $x = 0$), five modes dominate the dynamics. In contrast, for an asymmetric initial condition, twice as many modes are required to represent the dynamics with the same precision.

The singular value decomposition not only gives the distribution of energy within the first set of modes, but it also produces the optimal basis elements as columns of the matrix $\mathbf{U}$. The distribution of singular values is highly suggestive of how to truncate with a low-rank subspace of r modes, thus allowing us to construct the dimensionally reduced space (12.19) appropriate for a Galerkin–POD expansion.

The modes of the quantum harmonic oscillator are illustrated in Fig. 12.3. Specifically, the first five modes are shown for (i) the Gauss–Hermite functions

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

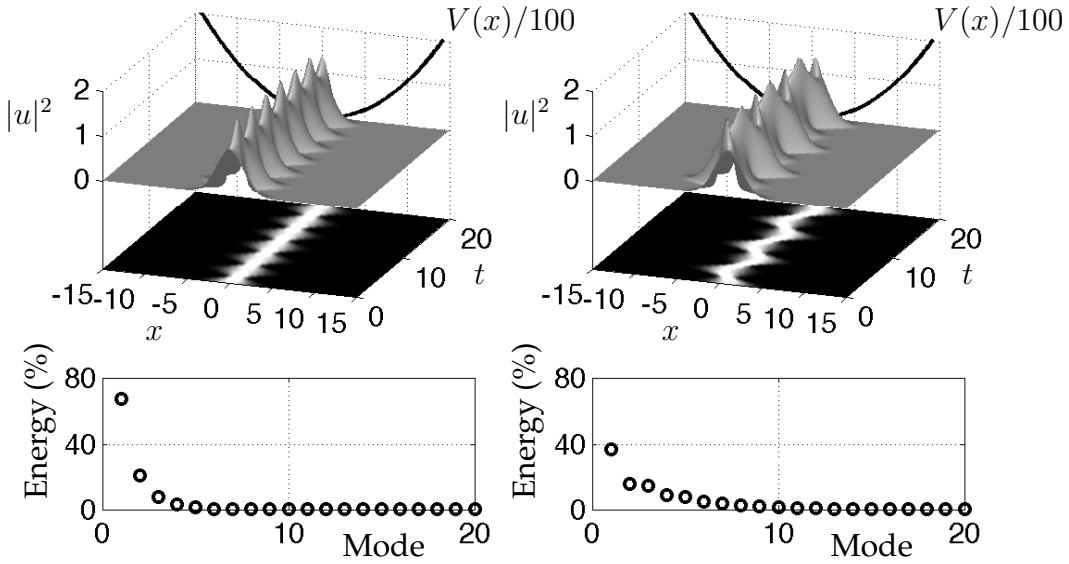


Figure 12.2: Dynamics of the quantum harmonic oscillator (12.23) given the initial condition $u(x,0) = \exp(-0.2(x-x_0)^2)$ for $x_0 = 0$ (top left) and $x_0 = 1$ (top right). The symmetric initial data elicits a dominant five-mode response while the initial condition with initial offset $x_0 = 1$ activates 10 modes. The bottom panels show the singular values of the SVD of the corresponding top panels, along with the percentage of energy (or L^2-norm) in each mode. The dynamics are clearly low-rank given the rapid decay of the singular values.

representing the special function solutions, (ii) the modes of the SVD for the symmetric ($x_0 = 0$) initial conditions, and (iii) the modes of the SVD for the offset (asymmetric, $x_0 = 1$) initial conditions. The Gauss–Hermite functions, by construction, are arranged from lowest eigenvalue of the Sturm–Liouville problem (12.25). The eigenmodes alternate between symmetric and asymmetric modes. For the symmetric (about $x = 0$) initial conditions given by $u(x,0) = \exp(-0.2x^2)$, the first five modes are all symmetric, as the snapshot-based method is incapable of producing asymmetric modes since they are actually not part of the dynamics, and thus they are not observable or manifested in the evolution. In contrast, with a slight offset, $u(x,0) = \exp(-0.2(x-1)^2)$, snapshots of the evolution produce asymmetric modes that closely resemble the asymmetric modes of the Gauss–Hermite expansion. Interestingly, in this case, the SVD arranges the modes by the amount of energy exhibited in each mode. Thus the first asymmetric mode (bottom panel in red – third mode) is equivalent to the second mode of the exact Gauss–Hermite polynomials (top panel in green – second mode). The key observation here is that the snapshot-based method is capable of generating, or nearly so, the known optimal Gauss–Hermite polynomials characteristic of this system. Importantly, the Galerkin–POD method generalizes to more complex physics and geometries where the solution is not

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

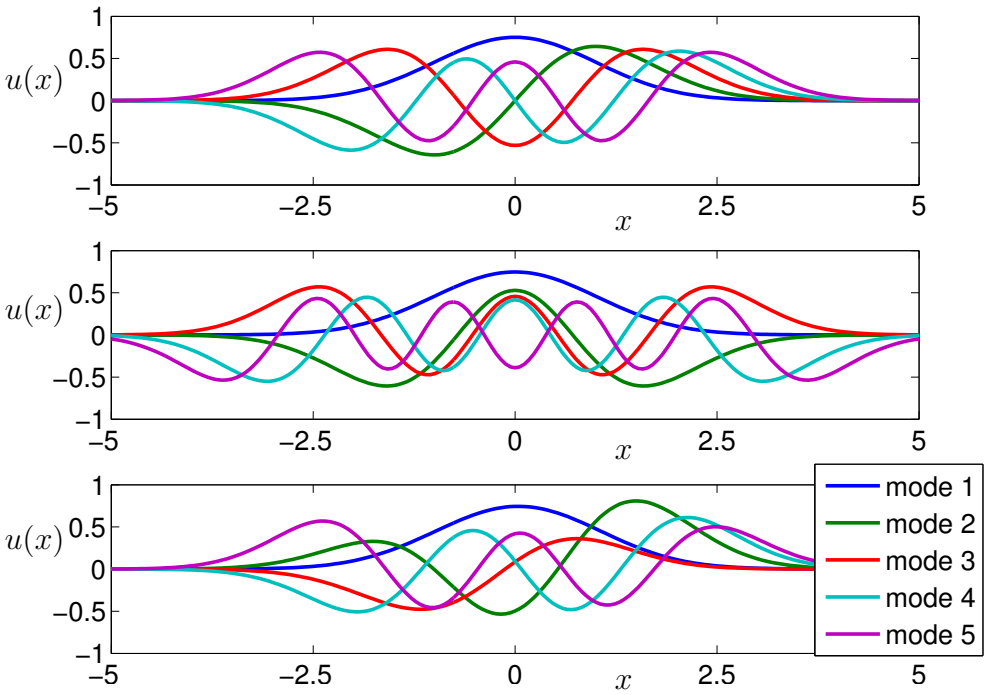


Figure 12.3: First five modes of the quantum harmonic oscillator. In the top panel, the first five Gauss–Hermite modes (12.26), arranged by their Sturm–Liouville eigenvalue, are illustrated. The second panel shows the dominant modes computed from the SVD of the dynamics of the harmonic oscillator with $u(x,0) = \exp(-0.2x^2)$, illustrated in Fig. 12.2 (left). Note that the modes are all symmetric, since no asymmetric dynamics was actually manifested. For the bottom panel, where the harmonic oscillator was simulated with the offset Gaussian $u(x,0) = \exp(-0.2(x-1)^2)$, asymmetry is certainly observed. This also produces modes that are very similar to the Gauss–Hermite functions. Thus a purely snapshot-based method is capable of reproducing the nearly ideal basis set for the harmonic oscillator.

known *a priori*.

12.3 POD and Soliton Dynamics

To illustrate a full implementation of the Galerkin–POD method, we will consider an illustrative complex system whose dynamics are strongly nonlinear. Thus, we consider the nonlinear Schrödinger (NLS) equation,

$$iu_t + \tfrac{1}{2}u_{xx} + |u|^2 u = 0, \tag{12.29}$$

with the boundary conditions $u \to 0$ as $x \to \pm\infty$. If not for the nonlinear term, this equation could be solved easily in closed form. However, the nonlinearity

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

mixes the eigenfunction components in the expansion (12.6), and it is impossible to derive a simple analytic solution.

To solve the NLS computationally, a Fourier mode expansion is used. Thus the standard fast Fourier transform may be leveraged. Rewriting (12.29) in the Fourier domain, i.e., taking the Fourier transform, gives the set of differential equations

$$\hat{u}_t = -\frac{i}{2}k^2\hat{u} + i\widehat{|u|^2u}, \tag{12.30}$$

where the Fourier mode mixing occurs due to the nonlinear mixing in the cubic term. This gives the system of differential equations to be solved in order to evaluate the NLS behavior.

It now remains to consider a specific spatial configuration for the initial condition. For the NLS, there are a set of special initial conditions called solitons where the initial conditions are given by

$$u(x,0) = N\mathrm{sech}(x), \tag{12.31}$$

where N is an integer. We will consider the soliton dynamics with $N = 1$ and $N = 2$. First, the initial condition is projected onto the Fourier modes with the fast Fourier transform.

The dynamics of the $N = 1$ and $N = 2$ solitons are demonstrated in Fig. 12.4. During evolution, the $N = 1$ soliton only undergoes phase changes while its amplitude remains stationary. In contrast, the $N = 2$ soliton undergoes periodic oscillations. In both cases, a large number of Fourier modes, about 50 and 200, respectively, are required to model the simple behaviors illustrated.

The obvious question to ask in light of our dimensionality reduction thinking is this: Is the soliton dynamics really a 50- or 200-degrees-of-freedom system as required by the Fourier mode solution technique? The answer is no. Indeed, with the appropriate basis, i.e., the POD modes generated from the SVD, it can be shown that the dynamics is a simple reduction to one or two modes, respectively. Indeed, it can easily be shown that the $N = 1$ and $N = 2$ solitons are truly low-dimensional, as shown by the evolutions in Fig. 12.4.

Figure 12.5 explicitly demonstrates the low-dimensional nature of the numerical solutions by computing the singular values, along with the modes to be used in our new eigenfunction expansion. For both of these cases, the dynamics are truly low-dimensional with the $N = 1$ soliton being modeled well by a single POD mode while the $N = 2$ dynamics are modeled quite well with two POD modes. Thus, in performing an eigenfunction expansion, the modes chosen should be the POD modes generated from the simulations themselves. In the next section, we will derive the dynamics of the modal interaction for these two cases, which are low-dimensional and amenable to analysis.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

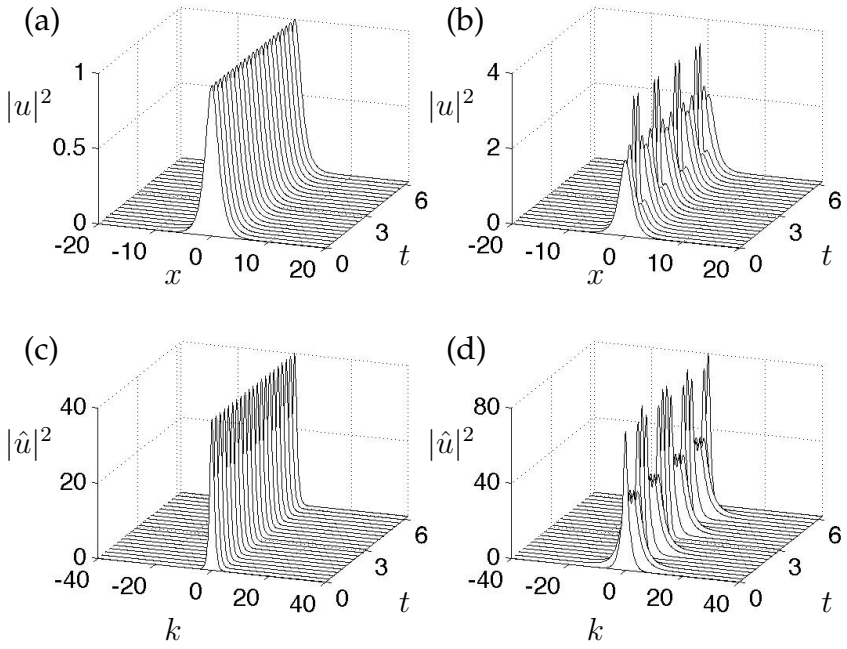


Figure 12.4: Evolution of the (a) $N = 1$ and (b) $N = 2$ solitons. Here steady-state ($N = 1$, panels (a) and (c)) and periodic ($N = 2$, panels (b) and (d)) dynamics are observed and approximately 50 and 200 Fourier modes, respectively, are required to model the behaviors.

Soliton Reduction ($N = 1$)

To take advantage of the low-dimensional structure, we first consider the $N = 1$ soliton dynamics. Figure 12.5 shows that a single mode in the SVD dominates the dynamics. This is the first column of the $\mathbf{U}$ matrix. Thus the dynamics are recast in a single mode so that

$$u(x,t) = a(t)\psi(x). \tag{12.32}$$

Plugging this into the NLS equation (12.29) yields the following:

$$ia_t\psi + \tfrac{1}{2}a\psi_{xx} + |a|^2 a|\psi|^2\psi = 0. \tag{12.33}$$

The inner product is now taken with respect to ψ, which gives

$$ia_t + \frac{\alpha}{2}a + \beta|a|^2 a = 0, \tag{12.34}$$

where

$$\alpha = \frac{\langle \psi_{xx}, \psi \rangle}{\langle \psi, \psi \rangle}, \tag{12.35a}$$

$$\beta = \frac{\langle |\psi|^2\psi, \psi \rangle}{\langle \psi, \psi \rangle}. \tag{12.35b}$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

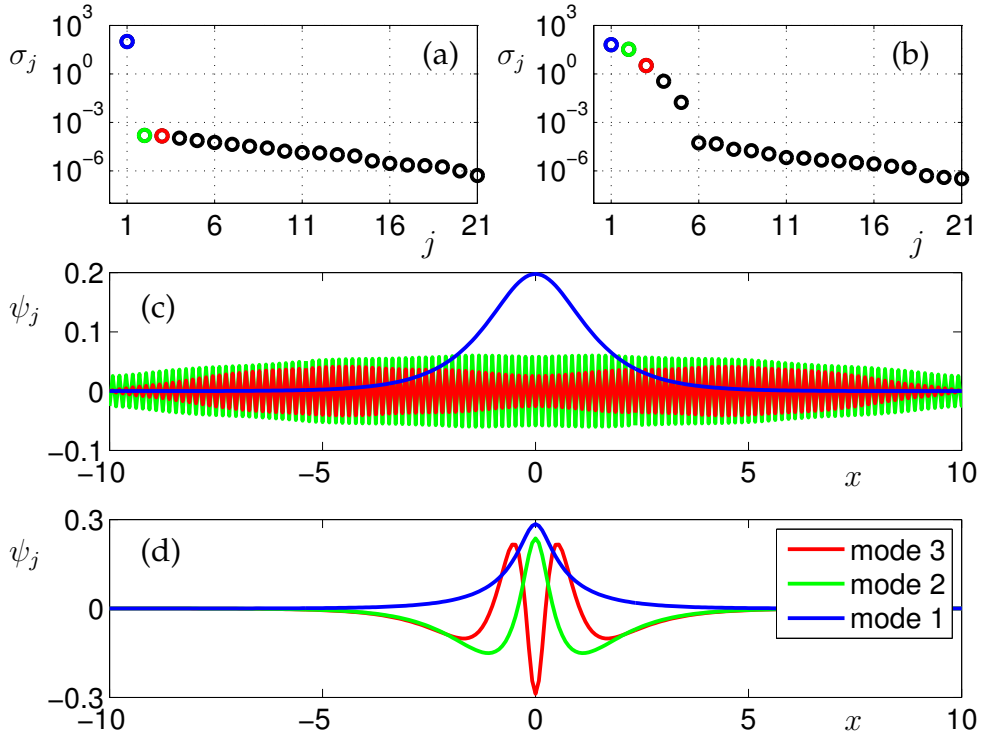


Figure 12.5: Projection of the $N = 1$ and $N = 2$ evolutions onto their POD modes. Panels (a) and (b) are the singular values σ_j on a logarithmic scale of the two evolutions demonstrated in Fig. 12.4. This demonstrates that the $N = 1$ and $N = 2$ soliton dynamics are primarily low-rank, with the $N = 1$ being a single-mode evolution and the $N = 2$ being dominated by two modes that contain approximately 95% of the evolution variance. The first three modes in both cases are shown in panels (c) and (d).

This is the low-rank approximation achieved by the Galerkin–POD method.

The differential equation (12.34) for $a(t)$ can be solved explicitly to yield

$$a(t) = a(0) \exp\left(i\frac{\alpha}{2}t + \beta|a(0)|^2 t\right), \tag{12.36}$$

where $a(0)$ is the initial condition for $a(t)$. To find the initial condition, recall that

$$u(x, 0) = \operatorname{sech}(x) = a(0)\psi(x). \tag{12.37}$$

Taking the inner product with respect to $\psi(x)$ gives

$$a(0) = \frac{\langle \operatorname{sech}(x), \psi \rangle}{\langle \psi, \psi \rangle}. \tag{12.38}$$

Thus the one-mode expansion gives the approximate PDE solution

$$u(x, t) = a(0) \exp\left(i\frac{\alpha}{2}t + \beta|a(0)|^2 t\right) \psi(x). \tag{12.39}$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

This solution is the low-dimensional POD approximation of the PDE expanded in the best basis possible, i.e., the SVD basis.

For the $N = 1$ soliton, the spatial profile remains constant while its phase undergoes a nonlinear rotation. The POD solution (12.39) can be solved exactly to characterize this phase rotation.

Soliton Reduction ($N = 2$)

The $N = 2$ soliton case is a bit more complicated and interesting. In this case, two modes clearly dominate the behavior of the system, as they contain 96% of the energy. These two modes, ψ_1 and ψ_2, are the first two columns of the matrix $\mathbf{U}$ and are now used to approximate the dynamics observed in Fig. 12.4. In this case, the two-mode expansion takes the form

$$u(x,t) = a_1(t)\psi_1(x) + a_2(t)\psi_2(x). \tag{12.40}$$

Inserting this approximation into the governing equation (12.29) gives

$$\begin{aligned} &i(a_{1t}\psi_1 + a_{2t}\psi_2) + \tfrac{1}{2}(a_1\psi_{1xx} + a_2\psi_{2xx}) \\ &\quad + (a_1\psi_1 + a_2\psi_2)^2(a_1^*\psi_1^* + a_2^*\psi_2^*) = 0. \end{aligned} \tag{12.41}$$

Multiplying out the cubic term gives

$$\begin{aligned} &i(a_{1t}\psi_1 + a_{2t}\psi_2) + \tfrac{1}{2}(a_1\psi_{1xx} + a_2\psi_{2xx}) \\ &\quad + |a_1|^2a_1|\psi_1|^2\psi_1 + |a_2|^2a_2|\psi_2|^2\psi_2 + 2|a_1|^2a_2|\psi_1|^2\psi_2 \\ &\quad + 2|a_2|^2a_1|\psi_2|^2\psi_1 + a_1^2a_2^*\psi_1^2\psi_2^* + a_2^2a_1^*\psi_2^2\psi_1^* = 0. \end{aligned} \tag{12.42}$$

All that remains is to take the inner product of this equation with respect to both $\psi_1(x)$ and $\psi_2(x)$. Recall that these two modes are orthogonal, resulting in the following 2×2 system of nonlinear equations:

$$\begin{aligned} &ia_{1t} + \alpha_{11}a_1 + \alpha_{12}a_2 + (\beta_{111}|a_1|^2 + 2\beta_{211}|a_2|^2)a_1 \\ &\quad + (\beta_{121}|a_1|^2 + 2\beta_{221}|a_2|^2)a_2 + \sigma_{121}a_1^2a_2^* + \sigma_{211}a_2^2a_1^* = 0, \end{aligned} \tag{12.43a}$$

$$\begin{aligned} &ia_{2t} + \alpha_{21}a_1 + \alpha_{22}a_2 + (\beta_{112}|a_1|^2 + 2\beta_{212}|a_2|^2)a_1 \\ &\quad + (\beta_{122}|a_1|^2 + 2\beta_{222}|a_2|^2)a_2 + \sigma_{122}a_1^2a_2^* + \sigma_{212}a_2^2a_1^* = 0, \end{aligned} \tag{12.43b}$$

where

$$\alpha_{jk} = \langle \psi_{j_{xx}}, \psi_k \rangle / 2, \tag{12.44a}$$

$$\beta_{jkl} = \langle |\psi_j|^2\psi_k, \psi_l \rangle, \tag{12.44b}$$

$$\sigma_{jkl} = \langle \psi_j^2\psi_k^*, \psi_l \rangle, \tag{12.44c}$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

and the initial values of the two components are given by

$$a_1(0) = \frac{\langle 2\,\mathrm{sech}(x), \psi_1 \rangle}{\langle \psi_1, \psi_1 \rangle}, \tag{12.45a}$$

$$a_2(0) = \frac{\langle 2\,\mathrm{sech}(x), \psi_2 \rangle}{\langle \psi_2, \psi_2 \rangle}. \tag{12.45b}$$

This gives a complete description of the two-mode dynamics predicted from the SVD analysis.

The two-mode dynamics accurately approximates the solution. However, there is a phase drift that occurs in the dynamics that would require both higher precision in the time series of the full PDE and more accurate integration of the inner products for the coefficients. Indeed, the most simple trapezoidal rule has been used to compute the inner products, and its accuracy is somewhat suspect; this issue will be addressed in the following section. Higher-order schemes could certainly help improve the accuracy. Additionally, incorporating the third or higher modes could also help. In either case, this demonstrates how one would use the low-dimensional structures to approximate PDE dynamics in practice.

12.4 Continuous Formulation of POD

Thus far, the POD reduction has been constructed to accommodate discrete data measurement snapshots $\mathbf{X}$ as given by (12.16). The POD reduction generates a set of low-rank basis modes $\mathbf{\Psi}$ so that the following least-squares error is minimized:

$$\underset{\mathbf{\Psi}\text{ s.t. rank}(\mathbf{\Psi})=r}{\operatorname{argmin}} \|\mathbf{X} - \mathbf{\Psi}\mathbf{\Psi}^T\mathbf{X}\|_F. \tag{12.46}$$

Recall that $\mathbf{X} \in \mathbb{C}^{n\times m}$ and $\mathbf{\Psi} \in \mathbb{C}^{n\times r}$, where r is the rank of the truncation.

In many cases, measurements are performed on a continuous-time process over a prescribed spatial domain; thus the data we consider are constructed from trajectories

$$u(x,t) \quad \text{with} \quad t \in [0,T],\ x \in [-L,L]. \tag{12.47}$$

Such data require a *continuous*-time formulation of the POD reduction. In particular, an equivalent of (12.46) must be constructed for these continuous-time trajectories. Note that, instead of a spatially dependent function $u(x,t)$, one can also consider a vector of trajectories $\mathbf{u}(t) \in \mathbb{C}^n$. This may arise when a PDE is discretized so that the infinite-dimensional spatial variable x is finite-dimensional. Wolkwein [737, 738] gives an excellent, technical overview of the POD method and its continuous formulation.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

To define the continuous formulation, we prescribe the inner product

$$\langle f(x), g(x) \rangle = \int_{-L}^{L} f(x) g^*(x)\, \mathrm{d}x. \tag{12.48}$$

To find the best-fit function through the entire temporal trajectory $u(x,t)$ in (12.47), the following minimization problem must be solved:

$$\min_{\psi} \frac{1}{T} \int_0^T \|u(x,t) - \langle u(x,t), \psi(x) \rangle \psi\|^2 \, \mathrm{d}t \quad \text{subject to} \quad \|\psi\|^2 = 1, \tag{12.49}$$

where the normalization of the temporal integral by $1/T$ averages the difference between the data and its low-rank approximation using the function ψ over the time $t \in [0, T]$. Equation (12.49) is equivalent to maximizing the inner product between the data $u(x,t)$ and the function $\psi(x)$, i.e., they are maximally parallel in function space. Thus the minimization problem can be restated as

$$\max_{\psi} \frac{1}{T} \int_0^T |\langle u(x,t), \psi(x) \rangle|^2 \, \mathrm{d}t \quad \text{subject to} \quad \|\psi\|^2 = 1. \tag{12.50}$$

The constrained optimization problem in (12.50) can be reformulated as a Lagrangian functional,

$$\mathcal{L}(\psi, \lambda) = \frac{1}{T} \int_0^T |\langle u(x,t), \psi(x) \rangle|^2 \mathrm{d}t + \lambda(1 - \|\psi\|^2), \tag{12.51}$$

where λ is the Lagrange multiplier that enforces the constraint $\|\psi\|^2 = 1$. This can be rewritten as

$$\begin{aligned} \mathcal{L}(\psi, \lambda) = \frac{1}{T} \int_0^T \left(\int_{-L}^{L} u(\xi, t) \psi^*(\xi) \, \mathrm{d}\xi \int_{-L}^{L} u^*(x,t) \psi(x) \, \mathrm{d}x \right) \mathrm{d}t \\ + \lambda(1 - \|\psi\|^2) + \lambda \left(1 - \int_{-L}^{L} \psi(x) \psi^*(x) \, \mathrm{d}x \right). \end{aligned} \tag{12.52}$$

The Lagrange multiplier problem requires that the functional derivative be zero:

$$\frac{\partial \mathcal{L}}{\partial \psi^*} = 0. \tag{12.53}$$

Applying this derivative constraint to (12.52) and interchanging integrals yields

$$\frac{\partial \mathcal{L}}{\partial \psi^*} = \int_{-L}^{L} \mathrm{d}\xi \left[\frac{1}{T} \int_0^T \left(u(\xi, t) \int_{-L}^{L} u^*(x,t) \psi(x) \, \mathrm{d}x \right) \mathrm{d}t - \lambda \psi(x) \right] = 0. \tag{12.54}$$

Setting the integrand to zero, the following eigenvalue problem is derived:

$$\langle R(\xi, x), \psi \rangle = \lambda \psi, \tag{12.55}$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

where $R(\xi, x)$ is a two-point correlation tensor of the continuous data $u(x,t)$, which is averaged over the time interval where the data is sampled:

$$R(\xi, x) = \frac{1}{T}\int_0^T u(\xi, t)u^*(x,t)\,\mathrm{d}t. \tag{12.56}$$

If the spatial direction x is discretized, resulting in a high-dimensional vector $\mathbf{u}(t) = \begin{bmatrix} u(x_1,t) & u(x_2,t) & \cdots & u(x_n,t)\end{bmatrix}^T$, then $R(\xi, x)$ becomes

$$\mathbf{R} = \frac{1}{T}\int_0^T \mathbf{u}(t)\mathbf{u}^*(t)\,\mathrm{d}t. \tag{12.57}$$

In practice, the function $\mathbf{R}$ is evaluated using a quadrature rule for integration. This will allow us to connect the method to the snapshot-based method discussed thus far.

Quadrature Rules for R: Trapezoidal Rule

The evaluation of the integral (12.57) can be performed by numerical quadrature [420]. The simplest quadrature rule is the trapezoidal rule, which evaluates the integral via summation of approximating rectangles. Figure 12.6 illustrates a version of the trapezoidal rule where the integral is approximated by a summation over a number of rectangles. This gives the approximation of the two-point correlation tensor:

$$\begin{aligned}\mathbf{R} &= \frac{1}{T}\int_0^T \mathbf{u}(t)\mathbf{u}^*(t)\,\mathrm{d}t \\ &\approx \frac{\Delta t}{T}[\mathbf{u}^*(t_1)\mathbf{u}(t_1) + \mathbf{u}^*(t_2)\mathbf{u}(t_2) + \cdots + \mathbf{u}^*(t_m)\mathbf{u}(t_m)] \\ &= \frac{\Delta t}{T}[\mathbf{u}_1^*\mathbf{u}_1 + \mathbf{u}_2^*\mathbf{u}_2 + \cdots + \mathbf{u}_m^*\mathbf{u}_m].\end{aligned} \tag{12.58}$$

Here we have assumed $u(x,t)$ is discretized into a vector $\mathbf{u}_j = \mathbf{u}(t_j)$, and there are m rectangular bins of width Δt so that $(m)\Delta t = T$. Defining a data matrix

$$\mathbf{X} = [\mathbf{u}_1 \quad \mathbf{u}_2 \quad \cdots \quad \mathbf{u}_m], \tag{12.59}$$

we can then rewrite the two-point correlation tensor as

$$\mathbf{R} \approx \frac{1}{m}\mathbf{X}^*\mathbf{X}, \tag{12.60}$$

which is exactly the definition of the covariance matrix in (1.39), i.e., $\mathbf{C} \approx \mathbf{R}$. Note that the role of $1/T$ is to average over the various trajectories so that the average is subtracted out, giving rise to a definition consistent with the covariance.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

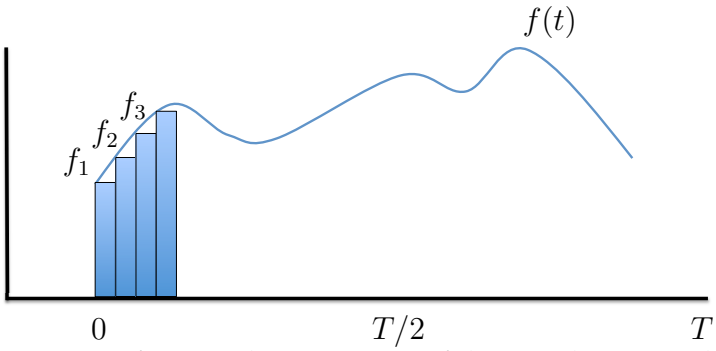


Figure 12.6: Illustration of an implementation of the quadrature rule to evaluate the integrals $\int_0^T f(t)\,\mathrm{d}t$. The rectangles of height $f(t_j) = f_j$ and width δt are summed to approximate the integral.

Higher-Order Quadrature Rules

Numerical integration simply calculates the area under a given curve. The basic ideas for performing such an operation come from the definition of integration,

$$\int_a^b f(t)\,\mathrm{d}t = \lim_{\Delta t \to 0} \sum_{j=0}^{m-1} f(t_j)\Delta t, \tag{12.61}$$

where $b - a = (m-1)\Delta t$. The area under the curve is a limiting process of summing up an ever-increasing number of rectangles. This process is known as numerical quadrature. Specifically, any sum can be represented as follows:

$$Q[f] = \sum_{j=0}^{m-1} w_j f(t_j) = w_0 f(t_0) + w_1 f(t_1) + \cdots + w_{m-1} f(t_{m-1}), \tag{12.62}$$

where $a = t_0 < t_1 < t_2 < \cdots < t_{m-1} = b$. Thus the integral is evaluated as

$$\int_a^b f(t)\,\mathrm{d}t = Q[f] + E[f], \tag{12.63}$$

where the term $E[f]$ is the error in approximating the integral by the quadrature sum (12.62). Typically, the error $E[f]$ is due to truncation error. To integrate, we will use polynomial fits to the y-values $f(t_j)$. Thus we assume the function $f(t)$ can be approximated by a polynomial,

$$P_n(t) = a_n t^n + a_{n-1} t^{n-1} + \cdots + a_1 t + a_0, \tag{12.64}$$

where the truncation error in this case is proportional to the $(n+1)$th derivative $E[f] = A f^{(n+1)}(c)$ and A is a constant. This process of polynomial fitting the data gives the *Newton–Cotes formulas*.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

The following integration approximations result from using a polynomial fit through the data to be integrated. It is assumed that

$$t_k = t_0 + \Delta t k \quad \text{and} \quad f_k = f(t_k). \tag{12.65}$$

This gives the following integration algorithms:

trapezoid rule

$$\int_{t_0}^{t_1} f(t)\,\mathrm{d}t = \frac{\Delta t}{2}(f_0 + f_1) - \frac{\Delta t^3}{12} f''(c), \tag{12.66a}$$

Simpson's rule

$$\int_{t_0}^{t_2} f(t)\,\mathrm{d}t = \frac{\Delta t}{3}(f_0 + 4f_1 + f_2) - \frac{\Delta t^5}{90} f^{(4)}(c), \tag{12.66b}$$

Simpson's 3/8 rule

$$\int_{t_0}^{t_3} f(t)\,\mathrm{d}t = \frac{3\Delta t}{8}(f_0 + 3f_1 + 3f_2 + f_3) - \frac{3\Delta t^5}{80} f^{(4)}(c), \tag{12.66c}$$

Boole's rule

$$\int_{t_0}^{t_4} f(t)\,\mathrm{d}t = \frac{2\Delta t}{45}(7f_0 + 32f_1 + 12f_2 + 32f_3 + 7f_4) - \frac{8\Delta t^7}{945} f^{(6)}(c). \tag{12.66d}$$

These algorithms have varying degrees of accuracy. Specifically, they are $O(\Delta t^2)$, $O(\Delta t^4)$, $O(\Delta t^4)$, and $O(\Delta t^6)$ accurate schemes, respectively. The accuracy condition is determined from the truncation terms of the polynomial fit. Note that the *trapezoidal rule* uses a sum of simple trapezoids to approximate the integral. *Simpson's rule* fits a quadratic curve through three points and calculates the area under the quadratic curve. *Simpson's 3/8 rule* uses four points and a cubic polynomial to evaluate the area, while *Boole's rule* uses five points and a quartic polynomial fit to generate an evaluation of the integral.

The integration methods (12.66) give values for the integrals over only a small part of the integration domain. The trapezoidal rule, for instance, only gives a value for $t \in [t_0, t_1]$. However, our fundamental aim is to evaluate the integral over the entire domain $t \in [a, b]$. Assuming once again that our interval is divided as $a = t_0 < t_1 < t_2 < \cdots < t_{m-1} = b$, then the trapezoidal rule applied over the interval gives the total integral

$$\int_a^b f(t)\,\mathrm{d}t \approx Q[f] = \sum_{j=1}^{m} \frac{\Delta t}{2}(f_j + f_{j+1}). \tag{12.67}$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Writing out this sum gives

$$\begin{aligned}\sum_{j=1}^{m}\frac{\Delta t}{2}(f_j+f_{j+1}) &= \frac{\Delta t}{2}(f_0+f_1)+\frac{\Delta t}{2}(f_1+f_2)+\cdots+\frac{\Delta t}{2}(f_m+f_{m-1})\\ &= \frac{\Delta t}{2}(f_0+2f_1+2f_2+\cdots+2f_m+f_{m-1}) \qquad (12.68)\\ &= \frac{\Delta t}{2}\left(f_0+f_{m-1}+2\sum_{j=1}^{m}f_j\right).\end{aligned}$$

The final expression no longer double-counts the values of the points between f_0 and f_{m-1}. Instead, the final sum only counts the intermediate values once, thus making the algorithm about twice as fast as the previous sum expression. These are computational savings which should always be exploited if possible.

POD Modes from Quadrature Rules

Any of these algorithms could be used to approximate the two-point correlation tensor $\mathbf{R}(\xi, x)$. The method of snapshots implicitly uses the trapezoidal rule to produce the snapshot matrix $\mathbf{X}$. Specifically, recall that

$$\mathbf{X} = \begin{bmatrix} | & | & & | \\ \mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_m \\ | & | & & | \end{bmatrix}, \qquad (12.69)$$

where the columns $\mathbf{u}_k \in \mathbb{C}^n$ may be measurements from simulations or experiments. The SVD of this matrix produces the modes used to produce a low-rank embedding $\mathbf{\Psi}$ of the data.

One could alternatively use a higher-order quadrature rule to produce a low-rank decomposition. Thus the matrix (12.69) would be modified to

$$\mathbf{X} = \begin{bmatrix} | & | & | & | & | & & | & | \\ \mathbf{u}_1 & 4\mathbf{u}_2 & 2\mathbf{u}_3 & 4\mathbf{u}_4 & 2\mathbf{u}_5 & \cdots & 4\mathbf{u}_{m-1} & \mathbf{u}_m \\ | & | & | & | & | & & | & | \end{bmatrix}, \qquad (12.70)$$

where the Simpson's rule quadrature formula is used. Simpson's rule is commonly used in practice, as it is simple to execute and provides significant improvement in accuracy over the trapezoidal rule. Producing this matrix simply involves multiplying the data matrix on the right by $\begin{bmatrix}1 & 4 & 2 & 4 & \cdots & 2 & 4 & 1\end{bmatrix}^T$. The SVD can then be used to construct a low-rank embedding $\mathbf{\Psi}$. Before approximating the low-rank solution, the quadrature weighting matrix must be undone. To our knowledge, very little work has been done in quantifying the merits of various quadrature rules. However, the interested reader should consider the optimal snapshot sampling strategy developed by Kunisch and Volkwein [419].

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

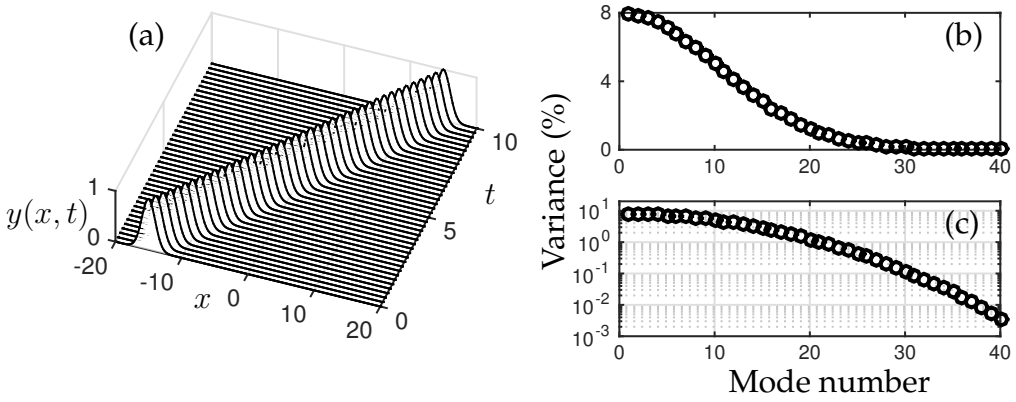


Figure 12.7: (a) Translating Gaussian with speed $c = 3$. The singular value decomposition produces a slow decay of the singular values, which is shown on a (b) normal and (c) logarithmic plot.

12.5 POD with Symmetries: Rotations and Translations

The POD method is not without its shortcomings. It is well known in the POD community that the underlying SVD algorithm does handle invariances in the data in an optimal way. The most common invariances arise from translational or rotational invariances in the data. Translational invariance is observed in the simple phenomenon of wave propagation, making it difficult for correlation to be computed, since critical features in the data are no longer aligned snapshot to snapshot.

In what follows, we will consider the effects of both translation and rotation. The examples are motivated from physical problems of practical interest. The important observation is that, unless the invariance structure is accounted for, the POD reduction will give an artificially inflated dimension for the underlying dynamics. This challenges our ability to use the POD as a diagnostic tool or as the platform for reduced-order models.

Translation: Wave Propagation

To illustrate the impact of translation on a POD analysis, consider a simple translating Gaussian propagating with velocity c:

$$u(x,t) = \exp[-(x - ct + 15)^2]. \tag{12.71}$$

We consider this solution on the space and time intervals $x \in [-20, 20]$ and $t \in [0, 10]$.

Figure 12.7(a) demonstrates the simple evolution to be considered. As is clear from the figure, the translation of the pulse will clearly affect the correla-

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

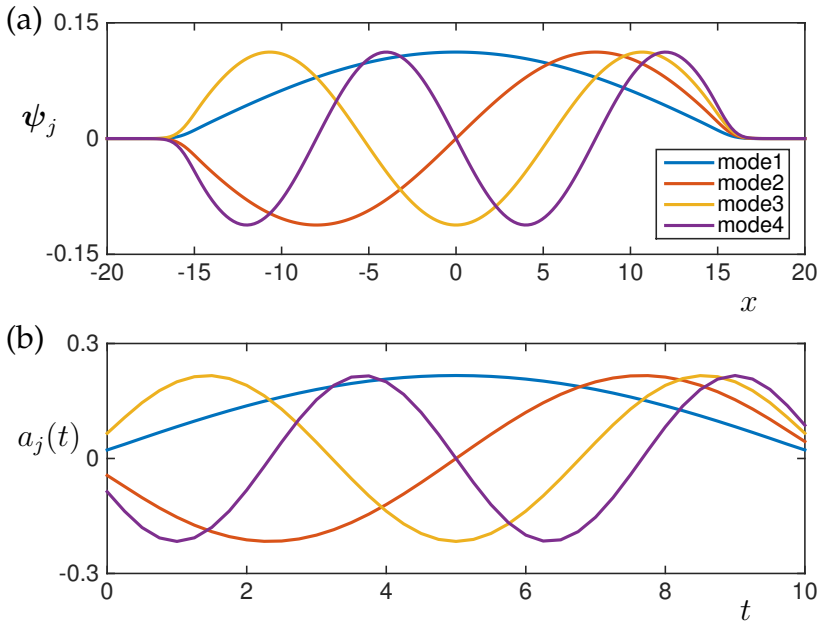


Figure 12.8: First four (a) spatial modes (first four columns of the $\mathbf{U}$ matrix) and (b) temporal modes (first four columns of the $\mathbf{V}$ matrix). A wave translating at a constant speed produces Fourier mode structures in both space and time.

tion at a given spatial location. Naive application of the SVD does not account for the translating nature of the data. As a result, the singular values produced by the SVD decay slowly, as shown in Figs. 12.7(b) and (c). In fact, the first few modes each contain approximately 8% of the variance.

The slow decay of singular values suggests that a low-rank embedding is not easily constructed. Moreover, there are interesting issues interpreting the POD modes and their time dynamics. Figure 12.8 shows the first four spatial ($\mathbf{U}$) and temporal ($\mathbf{V}$) modes generated by the SVD. The spatial modes are global in that they span the entire region where the pulse propagation occurred. Interestingly, they appear to be Fourier modes over the region where the pulse propagated. The temporal modes illustrate a similar Fourier mode basis for this specific example of a translating wave propagating at a constant velocity.

The failure of POD in this case is due simply to the translational invariance. If the invariance is *removed*, or factored out [609], before a data reduction is attempted, then the POD method can once again be used to produce a low-rank approximation. In order to remove the invariance, the invariance must first be identified and an auxiliary variable defined. Thus we consider the dynamics rewritten as

$$u(x,t) \to u(x - c(t)), \tag{12.72}$$

where $c(t)$ corresponds to the translational invariance in the system responsible

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.



Figure 12.9: Spiral waves (a) $u(x, y)$, (b) $|u(x, y)|$ and (c) $u(x, y)^5$ on the domain $x \in [-20, 20]$ and $y \in [-20, 20]$. The spirals are made to spin clockwise with angular velocity ω.

for limiting the POD method. The parameter c can be found by a number of methods. Rowley and Marsden [609] propose a template-based technique for factoring out the invariance. Alternatively, a simple center-of-mass calculation can be used to compute the location of the wave and the variable $c(t)$ [420].

Rotation: Spiral Waves

A second invariance commonly observed in simulations and data is associated with rotation. Much like translation, rotation moves a coherent, low-rank structure in such a way that correlations, which are produced at specific spatial locations, are no longer produced. To illustrate the effects of rotational invariance, a localized spiral wave with rotation will be considered.

A spiral wave centered at the origin can be defined as follows:

$$u(x, y) = \tanh[\sqrt{x^2 + y^2}\cos(A\angle(x + iy) - \sqrt{x^2 + y^2})], \tag{12.73}$$

where A is the number of arms of the spiral, and the $\angle$ denotes the phase angle of the quantity $(x+iy)$. To localize the spiral on a spatial domain, it is multiplied by a Gaussian centered at the origin so that our function of interest is given by

$$f(x, y) = u(x, y)\exp[-0.01(x^2 + y^2)]. \tag{12.74}$$

This function creates the *rotation* structure we wish to consider. The rate of spin can be made faster or slower by lowering or raising the value of the denominator, respectively.

In addition to considering the function $u(x, y)$, we will also consider the closely related functions $|u(x, y)|$ and $u(x, y)^5$ as shown in Fig. 12.9. Although these three functions clearly have the same underlying function that rotates, the change in functional form is shown to produce quite different low-rank approximations for the rotating waves.

To begin our analysis, consider the function $u(x, y)$ illustrated in Fig. 12.9(a). The SVD of this matrix can be computed and its low-rank structure evaluated. Two figures are produced (Figs. 12.10 and 12.11). The first assesses the rank

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

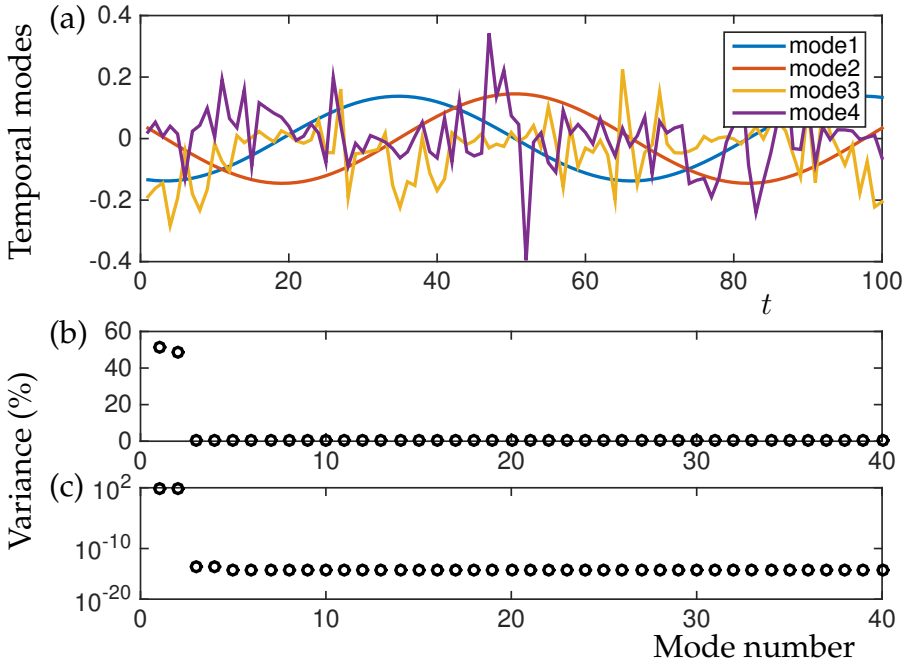


Figure 12.10: (a) First four temporal modes of the matrix $\mathbf{V}$. To numerical precision, all the variance is in the first two modes, as shown by the singular value decay on a (b) normal and (c) logarithmic plot. Remarkably, the POD extracts exactly two modes (see Fig. 12.11) to represent the rotating spiral wave.

of the observed dynamics and the temporal behavior of the first four modes in $\mathbf{V}$. Figures 12.10(b) and (c) show the decay of singular values on a regular and logarithmic scale, respectively. Remarkably, the first two modes capture *all* the variance of the data to numerical precision. This is further illustrated in the time dynamics of the first four modes. Specifically, the first two modes of Fig. 12.10(a) have a clear oscillatory signature associated with the rotation of modes one and two of Fig. 12.11. Modes 3 and 4 resemble noise in both time and space as a result of numerical round-off.

The spiral wave (12.74) allows for a two-mode truncation that is accurate to numerical precision. This is in part due to the sinusoidal nature of the solution when circumnavigating the solution at a fixed radius. Simply changing the data from $u(x,t)$ to either $|u(x,t)|$ or $u(x,t)^5$ reveals that the low-rank modes and their time dynamics are significantly different (see Figs. 12.12 and 12.13). Figures 12.12(a) and (b) show the decay of the singular values for these two new functions and demonstrate the significant difference from the two-mode evolution previously considered. The dominant time dynamics computed from the matrix $\mathbf{V}$ are also demonstrated. In the case of the absolute value of the function $|u(x,t)|$, the decay of the singular values is slow and never approaches numerical precision. The quintic function suggests a rank $r = 6$ truncation is

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

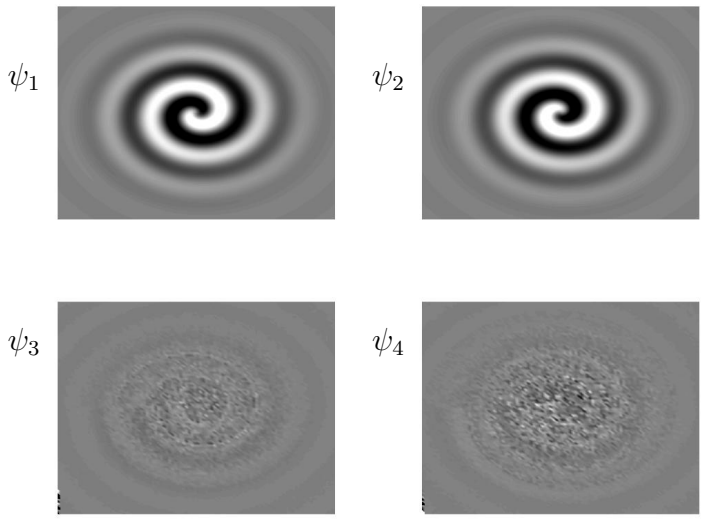


Figure 12.11: First four POD modes associated with the rotating spiral wave $u(x, y)$. The first two modes capture all the variance to numerical precision, while the third and fourth mode are noisy due to numerical round-off. The domain considered is $x \in [-20, 20]$ and $y \in [-20, 20]$.

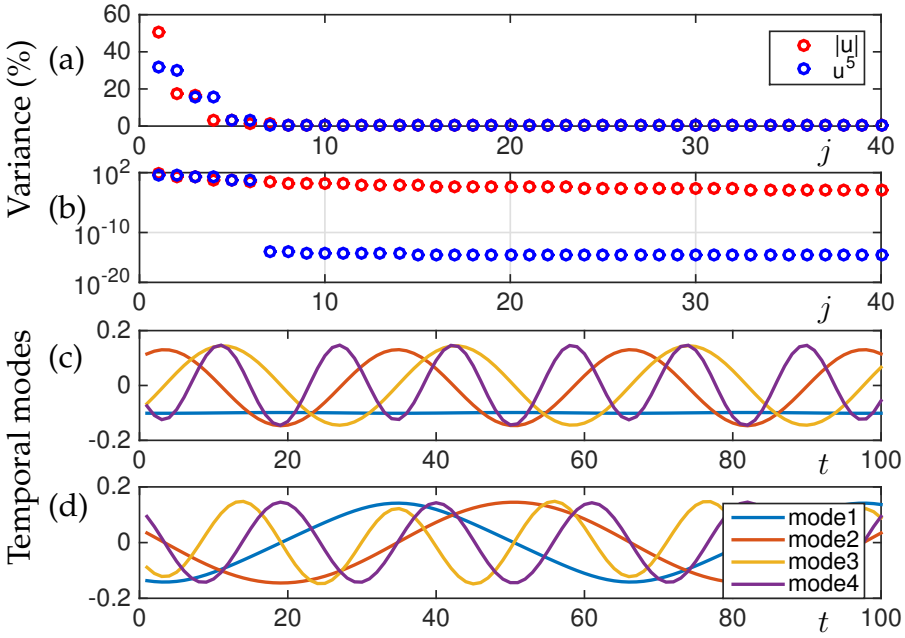


Figure 12.12: Decay of the singular values on a (a) normal and (b) logarithmic scale showing that the function $|u(x, t)|$ produces a slow decay while $u(x, t)^5$ produces an $r = 6$ approximation to numerical accuracy. The first four temporal modes of the matrix $\mathbf{V}$ are shown for these two functions in panels (c) and (d), respectively. The spatial modes are shown in Fig. 12.13.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

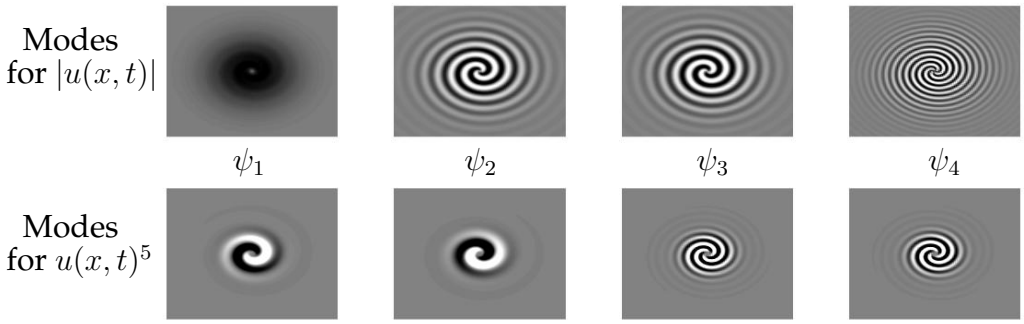


Figure 12.13: First four POD modes associated with the rotating spiral wave $|u(x,y)|$ (top row) and $u(x,t)^5$ (bottom row). Unlike our previous example, the first four modes do not capture all the variance to numerical precision, thus requiring more modes for accurate approximation. The domain considered is $x \in [-20, 20]$ and $y \in [-20, 20]$.

capable of producing an approximation to numerical precision. This highlights the fact that rotational invariance complicates the POD reduction procedure. After all, the only difference between the three rotating solutions is the actual shape of the rotating function, as they are all rotating with the same speed.

To conclude, invariance can severely limit the POD method. Most notably, it can artificially inflate the dimension of the system and lead to compromised interpretability. Expert knowledge of a given system and its potential invariances can help frame mathematical strategies to remove the invariances, i.e., re-aligning the data [420, 609]. But this strategy also has limitations, especially if two or more invariant structures are present. For instance, if two waves of different speeds are observed in the data, then the methods proposed for removing invariances will fail to capture both wave speeds simultaneously. Ultimately, dealing with invariances remains an open research question.

12.6 Neural Networks for Time-Stepping with POD

The emergence of machine learning is expanding the mathematical possibilities for the construction of accurate ROMs. As shown in the previous sections, the focus of traditional projection-based ROMs is on computing the low-dimensional subspace $\mathbf{\Psi}$ on which to project the governing equations. Recall that, in constructing the low-dimensional subspace, the SVD is used on snapshots of high-fidelity simulation (or experimental) data $\mathbf{X} \approx \mathbf{\Psi}\tilde{\mathbf{\Sigma}}\tilde{\mathbf{V}}^*$. The POD reduction technique uses only the single matrix $\mathbf{\Psi}$ in the reduction process. The temporal evolution in the reduced space $\mathbf{\Psi}$ is quantified by $\tilde{\mathbf{\Sigma}}\tilde{\mathbf{V}}^*$. This gives explicitly the evolution of each mode over the snapshots of $\mathbf{X}$, information that is

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

not used in projection-based ROMs. Neural networks can then be used directly on the time-series data encoded in $\mathbf{V}$ to build a time-stepping algorithm for marching the solution forward in time.

The motivation for using deep learning algorithms for time-stepping is the recognition that projection-based model reduction often can produce unstable iteration schemes [160]. A second important fact is that valuable temporal information in the low-dimensional space is summarily dismissed by the projection schemes, i.e., only the POD modes are retained for ROM construction. Neural networks aim to leverage the temporal information and in the process build efficient and stable time-stepping proxies. Recall that model reduction proceeds by projecting into the low-dimensional subspace spanned by $\mathbf{\Psi}$ so that

$$\mathbf{u}(t) \approx \mathbf{\Psi}\mathbf{a}(t). \tag{12.75}$$

In the projection-based ROMs of previous sections, the amplitude dynamics $\mathbf{a}(t)$ are constructed by Galerkin projection of the governing equations onto $\mathbf{\Psi}$. With neural networks, the dynamics $\mathbf{a}(t)$ are approximated from the discrete time-series data encoded in $\mathbf{V}$. Specifically, this gives

$$\mathbf{a}(t) \quad \Longrightarrow \quad \tilde{\mathbf{\Sigma}}\tilde{\mathbf{V}}^* = \begin{bmatrix} | & | & & | \\ \mathbf{a}_1 & \mathbf{a}_2 & \cdots & \mathbf{a}_m \\ | & | & & | \end{bmatrix} \tag{12.76}$$

over the m time snapshots of the original data matrix on which the ROM is to be constructed.

Deep learning algorithms provide a flexible framework for constructing a mapping between successive time-steps. As shown in Fig. 12.14, the typical ROM architecture constrains the dynamics to a subspace spanned by the POD modes $\mathbf{\Psi}$. Thus in the original coordinate system, the high-fidelity simulations of the governing equations for $\mathbf{u}$ are solved with a given numerical discretization scheme to produce a snapshot matrix $\mathbf{X}$ containing $\mathbf{u}_k$. In the new coordinate system, which is generated by projection to the subspace $\mathbf{\Psi}$, the snapshot matrix is now constructed from $\mathbf{a}_k$ as shown in (12.76). In traditional ROMs, the snapshot matrix (12.76) is not used. Instead snapshots of $\mathbf{a}_k$ are achieved by solving the Galerkin projected model (12.21). However, the snapshot matrix (12.76) can be used to construct a time-stepping model using neural networks. Neural networks allow one to use the high-fidelity simulation data to train a mapping

$$\mathbf{a}_{k+1} = \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{a}_k), \tag{12.77}$$

where $\mathbf{f}_{\boldsymbol{\theta}}$ is a generic representation of a neural network which is characterized by its structure, weights, and biases. Note that deep learning can also be used to learn nonlinear coordinates that generalize the SVD embedding.

Recently, Parish and Carlberg [548] and Regazzoni et al. [595] developed a suite of neural-network-based methods for learning time-stepping models

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

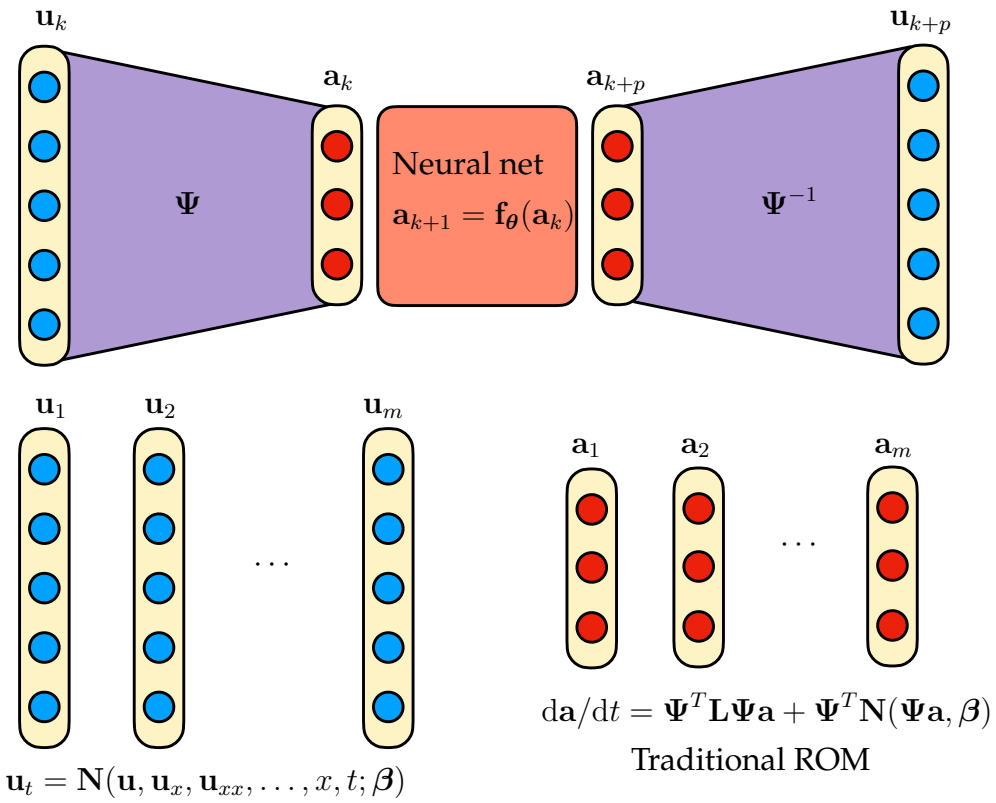


Figure 12.14: Illustration of neural network integration with POD subspaces. The autoencoder structure projects the original high-dimensional state-space data into a low-dimensional space via $\mathbf{u}(t) \approx \boldsymbol{\Psi}\mathbf{a}(t)$. As shown in the bottom left, the snapshots $\mathbf{u}_k$ are generated by high-fidelity numerical solutions of the governing equations $\mathbf{u}_t = \mathbf{N}(\mathbf{u}, \mathbf{u}_x, \mathbf{u}_{xx}, \ldots, x, t; \boldsymbol{\beta})$. In traditional ROMs, the snapshots $\mathbf{a}_k$ are constructed from Galerkin projection as shown in the bottom right. Neural networks instead learn a mapping $\mathbf{a}_{k+1} = \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{a}_k)$ from the original, low-dimensional snapshot data. It should be noted that time-stepping Runge–Kutta schemes, for instance, are a form of feedforward neural networks, which are used to produce the original high-fidelity data snapshots $\mathbf{u}_k$ [289].

for (12.77). Moreover, they provide extensive comparisons between different neural network architectures along with traditional techniques for time-series modeling. In such models the neural networks (or time-series analysis methods) simply map an input ($\mathbf{a}_k$) to an output ($\mathbf{a}_{k+1}$) as in Section 6.6. Autoencoders can also replace the POD embedding above.

In its simplest form, the neural network training requires input–output pairs

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

that can be generated from snapshots $\mathbf{a}_k$. Thus two matrices can be constructed:

$$\mathbf{A} = \begin{bmatrix} | & | & & | \\ \mathbf{a}_1 & \mathbf{a}_2 & \cdots & \mathbf{a}_{m-1} \\ | & | & & | \end{bmatrix} \quad \text{and} \quad \mathbf{A}' = \begin{bmatrix} | & | & & | \\ \mathbf{a}_2 & \mathbf{a}_3 & \cdots & \mathbf{a}_m \\ | & | & & | \end{bmatrix}, \tag{12.78}$$

where $\mathbf{A}$ denotes the input and $\mathbf{A}'$ denotes the output. This gives the training data necessary for learning (optimizing) a neural network map:

$$\mathbf{A}' = \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{A}). \tag{12.79}$$

There are numerous neural network architectures that can learn the mapping $\mathbf{f}_{\boldsymbol{\theta}}$. In Section 6.6, a simple feedforward network was already shown to be quite accurate in learning such a model. Further sophistication can improve accuracy and reduce data requirements for training.

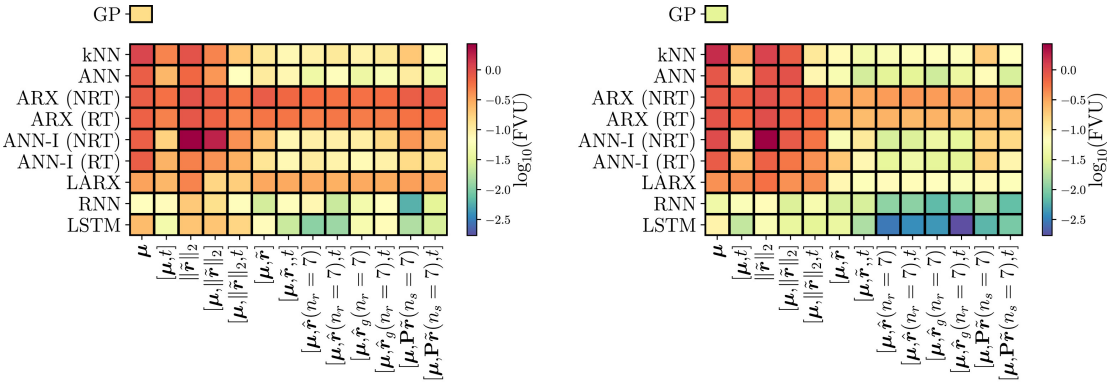
Regazzoni et al. [595] formulated the optimization of (12.79) in terms of maximum likelihood. Specifically, they considered the most suitable representation of the high-fidelity model in terms of simpler neural network models. They show that such neural network models can approximate the solution to within any accuracy required (limited by the accuracy of the training data, or course) simply by constructing them from the input–output pairs given by (12.79). Parish and Carlberg [548] provide an in-depth study of different neural network architectures that can be used for learning the time-steppers. They are especially focused on *recurrent neural network* (RNN) architectures that have proven to be so effective in temporal sequences associated with language [290]. Their extensive comparisons show that *long short-term memory* (LSTM) [331] neural networks outperform other methods and provide substantial improvements over traditional time-series approaches such as autoregressive models. In addition to a baseline *Gaussian process* (GP) regression, they specifically compare time-stepping models that include the following: k-nearest neighbors (kNN), artificial neural networks (ANN), autoregressive with exogenous inputs (ARX), integrated ANN (ANN-I), latent ARX (LARX), RNN, LSTM, and standard GP. Some models include recursive training (RT) and others do not (NRT). Their comparisons on a diversity of PDE models, which will not be detailed here, are evaluated on the *fraction of variance unexplained* (FVU). Figure 12.15 gives a representation of the extensive comparisons made on these methods for an advection–diffusion PDE model.

The success of neural networks for learning time-stepping representations fits more broadly under the aegis of *flow maps* [753], which were introduced in Section 7.1:

$$\mathbf{u}_{k+1} = \mathbf{F}(\mathbf{u}_k). \tag{12.80}$$

For neural networks, the flow map is approximated by the learned model (12.77) so that $\mathbf{F} = \mathbf{f}_{\boldsymbol{\theta}}$. Qin et al. [575] and Liu et al. [449] have explored the construction of flow maps from neural networks as yet another modeling paradigm for

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.



(a) Normed state error $\delta_{\boldsymbol{x}}$ **(b)** QoI error δ_s

Figure 12.15: Comparison of a diversity of error metrics and methods for constructing the mapping (12.77) for advection–diffusion equations. In all models considered in the paper, the LSTM and RNN structures proved to be the most accurate models for time-stepping. The reader is encouraged to consult the original paper for the details of the underlying models, the error metrics displayed, and the training data used. Python codes are available in the appendix of the original paper. From Parish and Carlberg [548].

advancing the solution in time without recourse to high-fidelity simulations. Such methods offer a broader framework for fast time-stepping algorithms, as no initial dimensionality reduction needs to be computed. In Qin et al. [575], the neural network model $\mathbf{f}_{\boldsymbol{\theta}}$ is constructed with a *residual network* (ResNet) as the basic architecture for approximation. In addition to a one-step method, which is shown to be exact in temporal integration, a recurrent ResNet and recursive ResNet are also constructed for multiple time-steps. Their formulation is also in the weak form where no derivative information is required in order to produce the time-stepping approximations. Several numerical examples are presented to demonstrate the performance of the methods. Like Parish and Carlberg [548] and Regazzoni et al. [595], the method is shown to be exceptionally accurate even in comparison with direct numerical integration, highlighting the qualities of the universal approximation properties of $\mathbf{f}_{\boldsymbol{\theta}}$.

Liu et al. [449] leveraged the flow map approximation scheme to learn a multi-scale time-stepping scheme. Specifically, one can learn flow maps for different characteristic timescales. Thus a given model

$$\mathbf{a}_{k+\tau} = \mathbf{f}_{\boldsymbol{\theta}_\tau}(\mathbf{a}_k) \tag{12.81}$$

can learn a flow map over a prescribed timescale τ. If there exist distinct timescales in the data, for instance denoted by t_1, t_2, and t_3 with $t_1 \gg t_2 \gg t_3$ (slow, medium, and fast times), then three models can be learned: $\mathbf{f}_{\boldsymbol{\theta}_1}$, $\mathbf{f}_{\boldsymbol{\theta}_2}$, and $\mathbf{f}_{\boldsymbol{\theta}_3}$ for

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

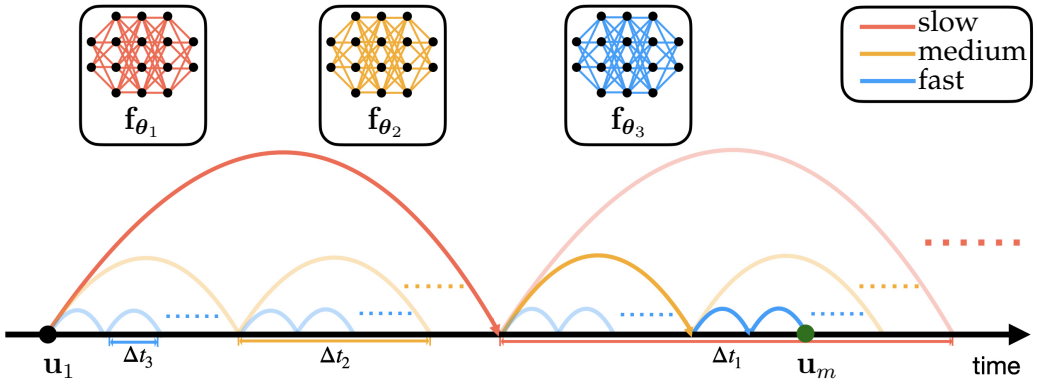


Figure 12.16: Multi-scale hierarchical time-stepping scheme. Neural network representations of the time-steppers are constructed over three distinct timescales. The red model takes large steps (slow timescale $\mathbf{f}_{\boldsymbol{\theta}_1}$), leaving the finer time-stepping to the yellow (medium timescale $\mathbf{f}_{\boldsymbol{\theta}_2}$) and blue (fast timescale $\mathbf{f}_{\boldsymbol{\theta}_3}$) models. The dark path shows the sequence of maps from $\boldsymbol{u}_1$ to $\boldsymbol{u}_m$. Modified from Liu et al. [449].

the slow, medium, and fast times, respectively. Figure 12.16 shows the *hierarchical time-stepping* (HiTS) scheme with three distinct timescales. The training data of a high-fidelity simulation, or collection of experimental data, allow for the construction of flow maps, which can then be used to efficiently forecast long times into the future. Specifically, one can use the flow map constructed on the slowest scale $\mathbf{f}_{\boldsymbol{\theta}_1}$ to march far into the future, while the medium and fast scales are then used to advance to the specific point in time. Thus a minimal number of steps is taken on the fast scale, and the work of forecasting long into the future is done by the slow and medium scales. The method is highly efficient and accurate.

Figure 12.17 compares the HiTS scheme across a number of example problems, some of which are videos and music frames. Thus HiTS does not require governing equations, simply time-series data arranged into input–output pairs. The performance of such flow maps is remarkably robust, stable, and accurate, even when compared to leading time-series neural networks such as LSTMs, *echo state networks* (ESNs), and *clockwork recurrent neural networks* (CW-RNNs). This is especially true for long forecasts, in contrast to the small time-steps evaluated in the work of Parish and Carlberg [548].

Overall, the works of Parish and Carlberg [548], Regazzoni et al. [595], Qin et al. [575], and Liu et al. [449] exploit very simple training paradigms related to input–output pairings of temporal snapshot data as structured in (12.78). This provides a significant potential improvement for learning time-stepping proxies to the Galerkin projected models such as (12.21).

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

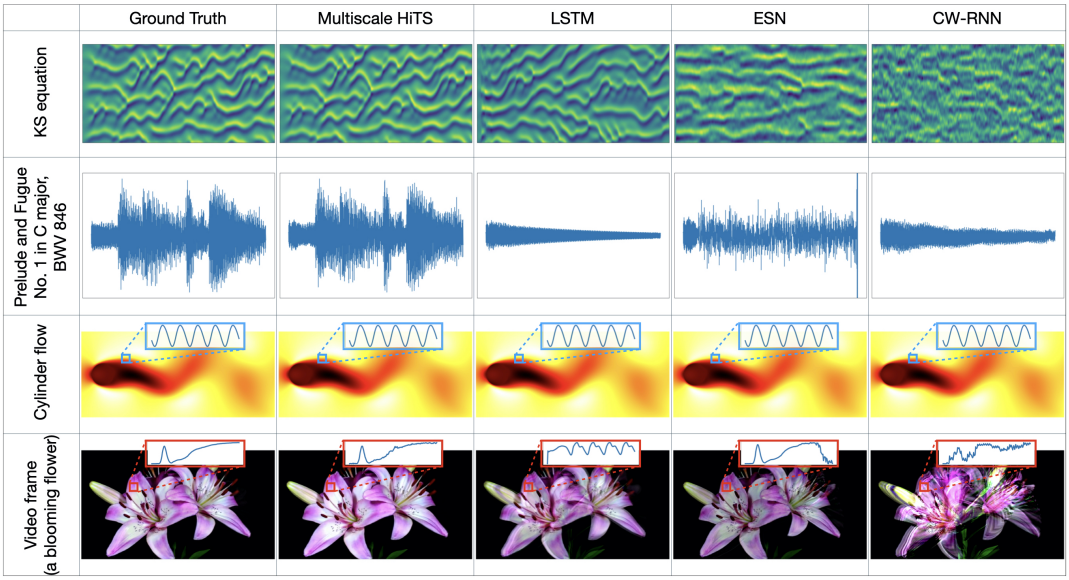


Figure 12.17: Evaluation of different neural network architectures (columns) on each training sequence (rows). Key diagnostics are visualized from a diversity of examples, including music files and videos. The last frame of the reconstruction is visualized for the first, third, and fourth examples, while the entire music score is visualized in the second example. Note the superior performance of the hierarchical time-stepping scheme in comparison with other modern neural network models such as LSTMs, *echo state networks* (ESNs), and *clockwork recurrent neural networks* (CW-RNNs). From Liu et al. [449]. The code is publicly available at `https://github.com/luckystarufo/multiscale_HiTS`.

12.7 Leveraging DMD and SINDy for POD-Galerkin

The construction of a traditional ROM that is accurate and efficient is centered on the reduction (12.21). Thus, once a low-rank subspace is computed from the SVD, the POD modes $\boldsymbol{\Psi}$ are used for projecting the dynamics. In the last section, projection of the governing evolution equations was circumvented by simply learning a neural network for the temporal (time-stepping) evolution. In this section, we use data-driven, non-intrusive methods in order to regress to a model for the temporal dynamics. Consider the evolution dynamics in (12.13):

$$\mathbf{u}_t = \mathbf{L}\mathbf{u} + \mathbf{N}(\mathbf{u}, \mathbf{u}_x, \mathbf{u}_{xx}, \ldots, x, t; \boldsymbol{\beta}) \tag{12.82}$$

where the linear and nonlinear parts of the evolution, denoted by $\mathbf{L}$ and $\mathbf{N}(\cdot)$, respectively, have been explicitly separated. The solution ansatz $\mathbf{u} = \boldsymbol{\Psi}\mathbf{a}$ yields the ROM

$$\frac{\mathrm{d}\mathbf{a}}{\mathrm{d}t} = \boldsymbol{\Psi}^T\mathbf{L}\boldsymbol{\Psi}\mathbf{a} + \boldsymbol{\Psi}^T\mathbf{N}(\boldsymbol{\Psi}\mathbf{a}, \boldsymbol{\beta}). \tag{12.83}$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Note that the linear operator in the reduced space $\boldsymbol{\Psi}^T\mathbf{L}\boldsymbol{\Psi}$ is an $r \times r$ matrix, which is easily computed. The nonlinear portion of the operator $\boldsymbol{\Psi}^T\mathbf{N}(\boldsymbol{\Psi}\mathbf{a}, \boldsymbol{\beta})$ is more complicated since it involves repeated computation of the operator as the solution $\mathbf{a}$, and consequently the high-dimensional state $\mathbf{u}$, is updated in time. Efficient interpolation methods for computing the nonlinear contribution in the ROM model are explored extensively in the next chapter of this book.

Simplifying POD-Galerkin with DMD

One method for overcoming the difficulties introduced in evaluating the nonlinear term on the right-hand side is to introduce the DMD algorithm. DMD approximates a set of snapshots by a best-fit linear model. Thus the nonlinearity can be evaluated over snapshots and a linear model constructed to approximate the dynamics. Thus two matrices can be constructed:

$$\mathbf{N} = \begin{bmatrix} | & | & & | \\ \mathbf{N}_1 & \mathbf{N}_2 & \cdots & \mathbf{N}_{m-1} \\ | & | & & | \end{bmatrix} \quad \text{and} \quad \mathbf{N}' = \begin{bmatrix} | & | & & | \\ \mathbf{N}_2 & \mathbf{N}_3 & \cdots & \mathbf{N}_m \\ | & | & & | \end{bmatrix} \tag{12.84}$$

where $\mathbf{N}_k$ is the evaluation of the nonlinear term $\mathbf{N}(\mathbf{u}, \mathbf{u}_x, \mathbf{u}_{xx}, \ldots, x, t; \boldsymbol{\beta})$ at $t = t_k$. Here $\mathbf{N}$ denotes the input and $\mathbf{N}'$ denotes the output. This gives the training data necessary for regressing to a DMD model,

$$\mathbf{N}' = \mathbf{A}_\mathbf{N}\mathbf{N}. \tag{12.85}$$

The governing equation (12.86) can then be approximated by

$$\mathbf{u}_t \approx \mathbf{L}\mathbf{u} + \mathbf{A}_\mathbf{N}\mathbf{u} = (\mathbf{A} + \mathbf{A}_\mathbf{N})\mathbf{u}, \tag{12.86}$$

where the operator $\mathbf{L}$ has been replaced by $\mathbf{A}$. The dynamics is now completely linear and solutions can be easily constructed from the eigenvalues and eigenvectors of the linear operator $\mathbf{A} + \mathbf{A}_\mathbf{N}$.

In practice, the DMD algorithm highlighted in Section 7.2 also exploits low-dimensional structure in building a ROM model. Thus instead of the approximate linear model (12.86), we instead wish to build a low-dimensional version. From snapshots (12.84) of the nonlinearity, the DMD algorithm can be used to approximate the dominant rank-r nonlinear contribution to the dynamics as

$$\mathbf{N}(\mathbf{u}, \mathbf{u}_x, \mathbf{u}_{xx}, \ldots, x, t; \boldsymbol{\beta}) \approx \sum_{j=1}^{r} b_j \boldsymbol{\phi}_j \exp(\omega_j t) = \boldsymbol{\Phi} \exp(\boldsymbol{\Omega} t)\mathbf{b}, \tag{12.87}$$

where b_j determines the weighting of each mode. Here $\boldsymbol{\phi}_j$ is the DMD mode and ω_j is the DMD eigenvalue. This approximation can be used in (12.88) to produce the POD–DMD approximation:

$$\frac{\mathrm{d}\mathbf{a}}{\mathrm{d}t} = \boldsymbol{\Psi}^T\mathbf{L}\boldsymbol{\Psi}\mathbf{a} + \boldsymbol{\Psi}^T\boldsymbol{\Phi} \exp(\boldsymbol{\Omega} t)\mathbf{b}. \tag{12.88}$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

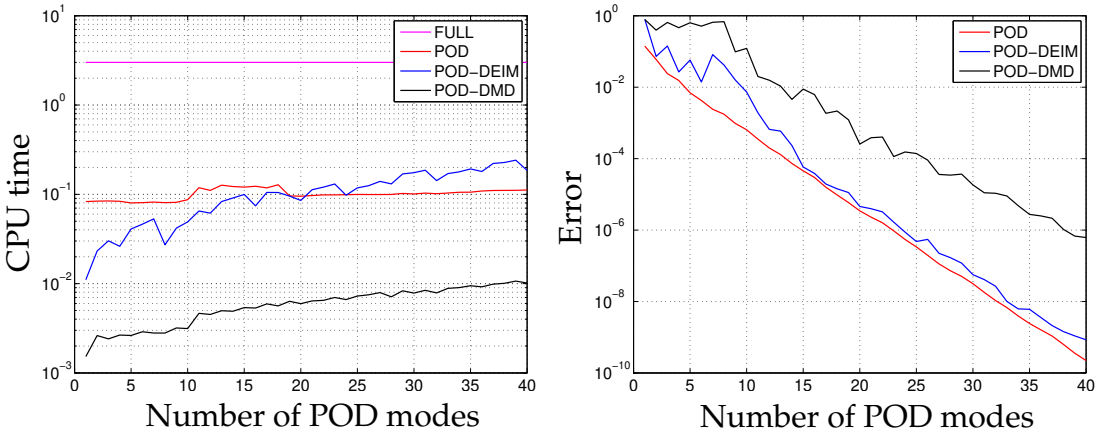


Figure 12.18: Computation time and accuracy on a semi-linear parabolic equation. Four methods are compared: the high-fidelity simulation of the governing equations (FULL); a Galerkin–POD reduction as given in (12.88) (POD); a Galerkin–POD reduction with the *discrete empirical interpolation* (DEIM) algorithm for evaluation of the nonlinearity (POD–DEIM); and the POD–DMD approximation (12.88). The left panel shows the computation times, which are an order of magnitude faster than for traditional POD–DEIM algorithms. The right panel shows the accuracy of the different methods for reproducing the high-fidelity simulations. POD–DMD loses some accuracy in comparison to Galerkin–POD methods due to the fact that DMD modes are not orthogonal, and thus the error does not decrease as quickly as in the POD-based methods. Modified from Alla and Kutz [10].

In this formulation, there are a number of advantageous features: (i) The nonlinearity is only evaluated once with the DMD algorithm (12.87). (ii) The products $\boldsymbol{\Psi}^T\mathbf{L}\boldsymbol{\Psi}$ and $\boldsymbol{\Psi}^T\boldsymbol{\Phi}$ are also only evaluated once and both produce matrices that are low-rank, i.e., they are independent of the original high-rank system. Thus with a one-time, up-front evaluation of two snapshot matrices to produce $\boldsymbol{\Psi}$ and $\boldsymbol{\Phi}$, the DMD produces a computationally efficient ROM that requires no recourse to the original high-dimensional system.

Alla and Kutz [10] integrated the DMD algorithm into the traditional ROM formalism to produce the POD–DMD model (12.88). The comparison of this computationally efficient ROM with traditional model reduction is shown in Fig. 12.18. Specifically, both the computational time and error are evaluated using this technique. Once the DMD algorithm is used to produce an approximation of the nonlinear term, it can be used for producing future state predictions and a computationally efficient ROM. Indeed, its computational acceleration is quite remarkable in comparison to traditional methods. Moreover, the method is non-intrusive and does not require additional evaluation of the nonlinear term. The entire method can be used with randomized algorithms to speed up

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

the low-rank evaluations even further [11]. Note that the computational performance boost comes at the expense of accuracy, as shown in Fig. 12.18. This is primarily due to the fact that additional POD modes used for standard ROMs, which are orthogonal by construction and guaranteed to be a best fit in ℓ_2, are now replaced by DMD modes which are no-longer orthogonal [10].

SINDy for POD-Galerkin Regression

The SINDy regression framework also allows one to build a parsimonious model for the evolution of the temporal dynamics in the low-rank subspace. Section 7.3 highlights the SINDy algorithm for model discovery. In the context of ROMs, the goal is now to discover a model of the evolution dynamics of a high-fidelity model embedded in a low-rank subspace. Recall that $\mathbf{u}(t) \approx \boldsymbol{\Psi}\mathbf{a}(t)$, $\boldsymbol{\Psi}$ can be computed with the SVD. The evolution of $\mathbf{a}(t)$ ultimately determines the temporal behavior of the system. Thus far, the temporal evolution has been computed via Galerkin projection and DMD. SINDy gives yet another alternative to model

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{a} = \mathbf{f}(\mathbf{a}), \tag{12.89}$$

where the right-hand side function prescribing the evolution dynamics $\mathbf{f}(\cdot)$ is unknown. SINDy provides a sparse regression framework to determine this dynamics. The snapshots of $\mathbf{a}(t)$ are collected into the matrix

$$\mathbf{A} = \begin{bmatrix} | & | & & | \\ \mathbf{a}_1 & \mathbf{a}_2 & \cdots & \mathbf{a}_m \\ | & | & & | \end{bmatrix}, \tag{12.90}$$

and the SINDy regression framework is then formulated as

$$\dot{\mathbf{A}} = \boldsymbol{\Theta}(\mathbf{A})\boldsymbol{\Xi}, \tag{12.91}$$

where each column $\boldsymbol{\xi}_k$ in $\boldsymbol{\Xi}$ is a vector of coefficients determining the active terms in the kth row in (12.89). As in Section 7.3, leveraging parsimony provides a dynamical model using as few terms as possible in $\boldsymbol{\Xi}$. Such a model may be identified using a convex ℓ_1-regularized sparse regression:

$$\boldsymbol{\xi}_k = \mathrm{argmin}_{\boldsymbol{\xi}'_k} \|\dot{\mathbf{A}}_k - \boldsymbol{\Theta}(\mathbf{A})\boldsymbol{\xi}'_k\|_2 + \lambda\|\boldsymbol{\xi}'_k\|_1. \tag{12.92}$$

Note that $\dot{\mathbf{a}}_k$ is the kth column of $\dot{\mathbf{A}}$, and λ is a sparsity-promoting hyperparameter. Section 7.3 discusses the many variants for sparsity promotion that can be used [53, 151, 153, 156, 157, 204, 702, 717], including the advocated sequential least-squares thresholding to select active terms.

Applying SINDy to POD mode coefficients provides a simple regression framework for discovering a parsimonious, and generally nonlinear, model for

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

the evolution dynamics of the high-dimensional system in a low-dimensional subspace. This approach, called sparse Galerkin regression [455], was illustrated in Fig. 7.6 in Section 7.3. For that example, the canonical example of flow past a circular cylinder was considered. This is modeled by the two-dimensional, incompressible Navier–Stokes equations:

$$\nabla \cdot \boldsymbol{u} = 0, \quad \partial_t \boldsymbol{u} + (\mathbf{u} \cdot \nabla)\mathbf{u} = -\nabla p + \frac{1}{Re}\Delta \boldsymbol{u}, \tag{12.93}$$

where $\boldsymbol{u}$ is the two-component flow velocity field in 2D and p is the pressure term. For Reynolds number $Re = Re_c \approx 47$, the fluid flow past a cylinder undergoes a supercritical Hopf bifurcation, where the steady flow for $Re < Re_c$ transitions to unsteady vortex shedding [60]. The unfolding gives the celebrated Stuart–Landau ODE, which is essentially the Hopf normal form in complex coordinates. This has resulted in accurate and efficient reduced-order models for this system [524, 526].

In Fig. 7.6, simulations at $Re = 100$ were considered. The snapshots of the evolution dynamics can be collected as in (12.16). Noack et al. [524] showed that the first two SVD modes and a third, orthogonal shift mode capture the essential dynamics of this flow. In these coordinates, the discovered dynamical model is given by

$$\dot{a}_1 = \mu a_1 - \omega a_2 + A a_1 a_3, \tag{12.94a}$$
$$\dot{a}_2 = \omega a_1 + \mu a_2 + A a_2 a_3, \tag{12.94b}$$
$$\dot{a}_3 = -\lambda(a_3 - a_1^2 - a_2^2), \tag{12.94c}$$

which is the same as was found by Noack et al. [524] through a detailed asymptotic reduction of the flow dynamics. Thus the ROM evolution dynamics (12.94) provide a non-intrusive, purely data-driven path to discover models similar to those achieved via Galerkin–POD projection. Not only is this model stable, but it also captures the correct supercritical Hopf bifurcation dynamics as a function of Reynolds number. Loiseau et al. [455, 456] also showed that it is possible to incorporate partially known physics, such as the energy preserving skew-symmetry of the quadratic terms, as constraints in the SINDy regression.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Suggested Reading

Texts

(1) Certified reduced basis methods for parametrized partial differential equations, by J. Hesthaven, G. Rozza, and B. Stamm, 2015 [325].

(2) Reduced basis methods for partial differential equations: An introduction, by A. Quarteroni, A. Manzoni, and N. Federico, 2015 [578].

(3) Model reduction and approximation: Theory and algorithms, by P. Benner, A. Cohen, M. Ohlberger, and K. Willcox, 2017 [74].

(4) Turbulence, coherent structures, dynamical systems and symmetry, by P. Holmes, J. L. Lumley, G. Berkooz, and C. W. Rowley, 2012 [335].

Papers and reviews

(1) A survey of model reduction methods for parametric systems, by P. Benner, S. Gugercin, and K. Willcox, *SIAM Review*, 2015 [75].

(2) Model reduction using proper orthogonal decomposition, by S. Volkwein, *Lecture Notes, Institute of Mathematics and Scientific Computing, University of Graz*, 2011 [737].

(3) The proper orthogonal decomposition in the analysis of turbulent flows, by G. Berkooz, P. Holmes, and J. L. Lumley, *Annual Review of Fluid Mechanics*, 1993 [79].

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Homework

Exercise 12-1. Using the flow around a cylinder data, compute the singular value spectrum and POD modes using the standard method of snapshots (using the trapezoidal rule when discretizing the continuous formulation) and compare it against the modes and spectrum when using Simpson's rule and Boole's rule for integration. Quantify the difference in modes using least-squares error.

Exercise 12-2. Generate high-fidelity, well-resolved solutions for the Kuramoto–Sivashinsky (KS) equation in a parameter regime where spatio-temporal chaos is exhibited. Build a number of reduced-order models using the high-fidelity data and test the models for future state prediction.

(a) Compute the leading POD modes and produce a rank-r Galerkin–POD approximation of the PDE evolution.

(b) Compute a ROM by using the snapshots to produce a rank-r DMD model to characterize the evolution.

(c) Compute a POD–DMD model where the nonlinear terms are approximated using a DMD model.

(d) Learn both a feedforward and LSTM network for advancing the solution in time in a rank-r POD basis.

Compare the various architectures in terms of their stability and future state prediction capabilities. Investigate the dynamics as a function of the rank reduction parameter r. Repeat the experiments by adding noise to the high-fidelity simulation data. Repeat the experiments yet again with both noise and added outliers (corruption) to the high-fidelity simulation data.

Exercise 12-3. Learn a deep neural network autoencoder to build a linear model for flow around a cylinder. Use high-fidelity flow around a cylinder data to learn a coordinate (autoencoder) transformation to an ($r = 3$)-dimensional subspace where the dynamics is linear and a Koopman operator can be constructed [465]. In the new linear coordinates, compute the eigenvalues and eigenvectors of the latent state representation. Use the model to forecast the future state and compare with the high-fidelity simulations.

Exercise 12-4. Learn a deep neural network autoencoder to build a parsimonious, but nonlinear, model for flow around a cylinder. Use high-fidelity flow around a cylinder data to learn a coordinate (autoencoder) transformation to an

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

($r = 3$)-dimensional subspace where the dynamics is given by a parsimonious dynamical system [168]. In the new linear coordinates, compute the eigenvalues and eigenvectors of the latent state representation. Use the model to forecast the future state and compare with the high-fidelity simulations.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Chapter 13

Interpolation for Parametric Reduced-Order Models

In the last chapter, the mathematical framework of ROMs was outlined. Specifically, Chapter 12 has already highlighted the POD method for projecting PDE dynamics to low-rank subspaces where simulations of the governing PDE model can be more readily evaluated. However, the complexity of projecting into the low-rank approximation subspace remains challenging due to the nonlinearity. Interpolation in combination with POD overcomes this difficulty by providing a computationally efficient method for discretely (sparsely) sampling and evaluating the nonlinearity. This chapter leverages the ideas of the sparse and compressive sampling algorithms of Chapter 3 where a small number of samples are capable of reconstructing the low-rank dynamics of PDEs. Ultimately, these methods ensure that the computational complexity of ROMs scale favorably with the rank of the approximation, even for complex nonlinearities. The primary focus of this chapter is to highlight sparse interpolation methods that enable a rapid and low-dimensional construction of the ROMs. In practice, these techniques dominate the ROM community since they are critically enabling for evaluating parametrically dependent PDEs where frequent ROM model updates are required.

13.1 Gappy POD

The success of nonlinear model order reduction is largely dependent upon two key innovations: (i) the well-known Galerkin–POD method [79, 335, 737, 738], which is used to project the high-dimensional nonlinear dynamics onto a low-dimensional subspace in a principled way; and (ii) sparse sampling of the state space for interpolating the nonlinear terms required for the subspace projection. Thus sparsity is already established as a critically enabling mathematical framework for model reduction through methods such as gappy POD and its variants [162, 215, 239, 754, 767]. Indeed, efficiently managing the compu-

tation of the nonlinearity was recognized early on in the ROMs community, and a variety of techniques were proposed to accomplish this task. Perhaps the first innovation in sparse sampling with POD modes was the technique proposed by Everson and Sirovich for which the gappy POD moniker was derived [239]. In their sparse sampling scheme, random measurements were used to approximate inner products. Principled selection of the interpolation points, through the gappy POD infrastructure [162, 215, 239, 754, 767] or missing point (best points) estimation (MPE) [29, 522], was quickly incorporated into ROMs to improve performance. More recently, the empirical interpolation method (EIM) [55] and its most successful variant, the POD-tailored discrete empirical interpolation method (DEIM) [171], have provided a greedy algorithm that allows for nearly optimal reconstructions of nonlinear terms of the original high-dimensional system. The DEIM approach combines projection with interpolation. Specifically, DEIM uses selected interpolation indices to specify an interpolation-based projection for a nearly optimal ℓ_2 subspace approximating the nonlinearity.

The low-rank approximation provided by POD allows for a reconstruction of the solution $\mathbf{u}(x,t)$ in (13.9) with r measurements of the n-dimensional state. This viewpoint has profound consequences on how we might consider measuring our dynamical system [239]. In particular, only $r \ll n$ measurements are required for reconstruction, allowing us to define the sparse representation variable $\tilde{\mathbf{u}} \in \mathbb{C}^r$,

$$\tilde{\mathbf{u}} = \mathbf{P}\mathbf{u}, \tag{13.1}$$

where the measurement matrix $\mathbf{P} \in \mathbb{R}^{r\times n}$ specifies r measurement locations of the full state $\mathbf{u} \in \mathbb{C}^n$. As an example, the measurement matrix might take the form

$$\mathbf{P} = \begin{bmatrix} 1 & 0 & \cdots & & & & & & \cdots & 0 \\ 0 & \cdots & 0 & 1 & 0 & \cdots & & & \cdots & 0 \\ 0 & \cdots & & & \cdots & 0 & 1 & 0 & \cdots & 0 \\ \vdots & 0 & & & \cdots & 0 & 0 & 1 & \cdots & \vdots \\ 0 & \cdots & & & \cdots & 0 & 0 & 0 & \cdots & 1 \end{bmatrix}, \tag{13.2}$$

where measurement locations take on the value of unity and the matrix elements are zero elsewhere. The matrix $\mathbf{P}$ defines a projection onto an r-dimensional space $\tilde{\mathbf{u}}$ that can be used to approximate solutions of a PDE.

The insight and observation of (13.1) forms the basis of the *gappy POD* method introduced by Everson and Sirovich [239]. In particular, one can use a small number of measurements, or gappy data, to reconstruct the full state of the system. In doing so, we can overcome the complexity of evaluating higher-order nonlinear terms in the POD reduction.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Sparse Measurements and Reconstruction

The measurement matrix $\mathbf{P}$ allows for an approximation of the state vector $\mathbf{u}$ from r measurements. The approximation is obtained by using (13.1) with the standard POD projection:

$$\tilde{\mathbf{u}} \approx \mathbf{P} \sum_{k=1}^{r} \tilde{a}_k \boldsymbol{\psi}_k, \tag{13.3}$$

where the coefficients $\tilde{a}_k$ minimize the error in approximation: $\|\tilde{\mathbf{u}} - \mathbf{P}\mathbf{u}\|$. The challenge now is how to determine the $\tilde{a}_k$ given that taking inner products of (13.3) can no longer be performed. Specifically, the vector $\tilde{\mathbf{u}}$ has dimension r whereas the POD modes have dimension n, i.e., the inner product requires information from the full range of $\mathbf{x}$, the underlying discretized spatial variable, which is of length n. Thus, the modes $\boldsymbol{\psi}_k(x)$ are in general not orthogonal over the r-dimensional support of $\tilde{\mathbf{u}}$. The support will be denoted as $s[\tilde{\mathbf{u}}]$. More precisely, orthogonality must be considered on the full range versus the support space. Thus the following two relationships hold:

$$M_{kj} = \langle \boldsymbol{\psi}_k, \boldsymbol{\psi}_j \rangle = \delta_{kj}, \tag{13.4a}$$

$$M_{kj} = \langle \boldsymbol{\psi}_k, \boldsymbol{\psi}_j \rangle_{s[\tilde{\mathbf{u}}]} \neq 0 \quad \text{for all } k, j, \tag{13.4b}$$

where M_{kj} are the entries of the Hermitian matrix $\mathbf{M}$ and δ_{kj} is the Kronecker delta function. The fact that the POD modes are not orthogonal on the support $s[\tilde{\mathbf{u}}]$ leads us to consider alternatives for evaluating the vector $\tilde{\mathbf{a}}$.

To determine the $\tilde{a}_k$, a least-squares algorithm can be used to minimize the error,

$$E = \int_{s[\tilde{\mathbf{u}}]} \left[\tilde{\mathbf{u}} - \sum_{k=1}^{r} \tilde{a}_k \boldsymbol{\psi}_k \right]^2 \mathrm{d}\mathbf{x}, \tag{13.5}$$

where the inner product is evaluated on the support $s[\tilde{\mathbf{u}}]$, thus making the two terms in the integral of the same size r. The minimizing solution to (13.5) requires the residual to be orthogonal to each mode $\boldsymbol{\psi}_k$, so that

$$\left\langle \tilde{\mathbf{u}} - \sum_{k=1}^{r} \tilde{a}_k \boldsymbol{\psi}_k, \boldsymbol{\psi}_j \right\rangle_{s[\tilde{\mathbf{u}}]} = 0 \quad \text{for } j \neq k,\ j = 1, 2, \ldots, r. \tag{13.6}$$

In practice, we can project the full-state vector $\mathbf{u}$ onto the support space and determine the vector $\tilde{\mathbf{a}}$:

$$\mathbf{M}\tilde{\mathbf{a}} = \mathbf{f}, \tag{13.7}$$

where the elements of $\mathbf{M}$ are given by (13.4b) and the components of the vector $\mathbf{f}$ are given by

$$f_k = \langle \mathbf{u}, \boldsymbol{\psi}_k \rangle_{s[\tilde{\mathbf{u}}]}. \tag{13.8}$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Note that if the measurement space is sufficiently dense, or if the support space is the entire space, then $\mathbf{M} = \mathbf{I}$, implying that the eigenvalues of $\mathbf{M}$ approach unity as the number of measurements becomes dense. Once the vector $\tilde{\mathbf{a}}$ is determined, a reconstruction of the solution can be performed as

$$\mathbf{u}(x,t) \approx \mathbf{\Psi}\tilde{\mathbf{a}}. \tag{13.9}$$

As the measurements become dense, not only does the matrix $\mathbf{M}$ converge to the identity, but also $\tilde{\mathbf{a}} \to \mathbf{a}$. Interestingly, these observations lead us to consider the efficacy of the method and/or approximation by considering the condition number of the matrix $\mathbf{M}$ [711]:

$$\kappa(\mathbf{M}) = \|\mathbf{M}\|\,\|\mathbf{M}^{-1}\| = \frac{\sigma_1}{\sigma_m}. \tag{13.10}$$

Here the 2-norm has been used. If $\kappa(\mathbf{M})$ is small, then the matrix is said to be well conditioned. A minimal value of $\kappa(\mathbf{M})$ is achieved with the identity matrix $\mathbf{M} = \mathbf{I}$. Thus, as the sampling space becomes dense, the condition number also approaches unity. This can be used as a metric for determining how well the sparse sampling is performing. Large condition numbers suggest poor reconstruction, while values tending toward unity should perform well.

Harmonic Oscillator Modes

To demonstrate the gappy sampling method and its reconstruction efficacy, we apply the technique using the first 10 modes of the Gauss–Hermite functions defined by (12.25) and (12.26). To compute the second derivative, we use the fact that the Fourier transform $\mathcal{F}$ can produce a spectrally accurate approximation, i.e., $u_{xx} = \mathcal{F}^{-1}[(ik)^2\mathcal{F}u]$. For the sake of producing accurate derivatives, we consider the domain $x \in [-10, 10]$ but then work with the smaller domain of interest $x \in [-4, 4]$. Recall further that the Fourier transform assumes a 2π-periodic domain. This is handled by a scaling factor in the k wavevectors. The first five modes have been demonstrated in Fig. 12.3.

The mode construction is shown in the top panel of Fig. 13.1. Each colored cell represents the discrete value of the mode in the interval $x \in [-4, 4]$ with $\Delta x = 0.1$. Thus there are 81 discrete values for each of the modes $\boldsymbol{\psi}_k$. Our objective is to reconstruct a function outside of the basis modes of the harmonic oscillator. In particular, consider the function

$$f(x) = \exp[-(x-0.5)^2] + 3\exp[-2(x+3/2)^2], \tag{13.11}$$

which will be discretized and defined over the same domain as the modal basis of the harmonic oscillator. We construct this function and further numerically construct the projection of the function onto the basis functions $\boldsymbol{\psi}_n$. The original

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

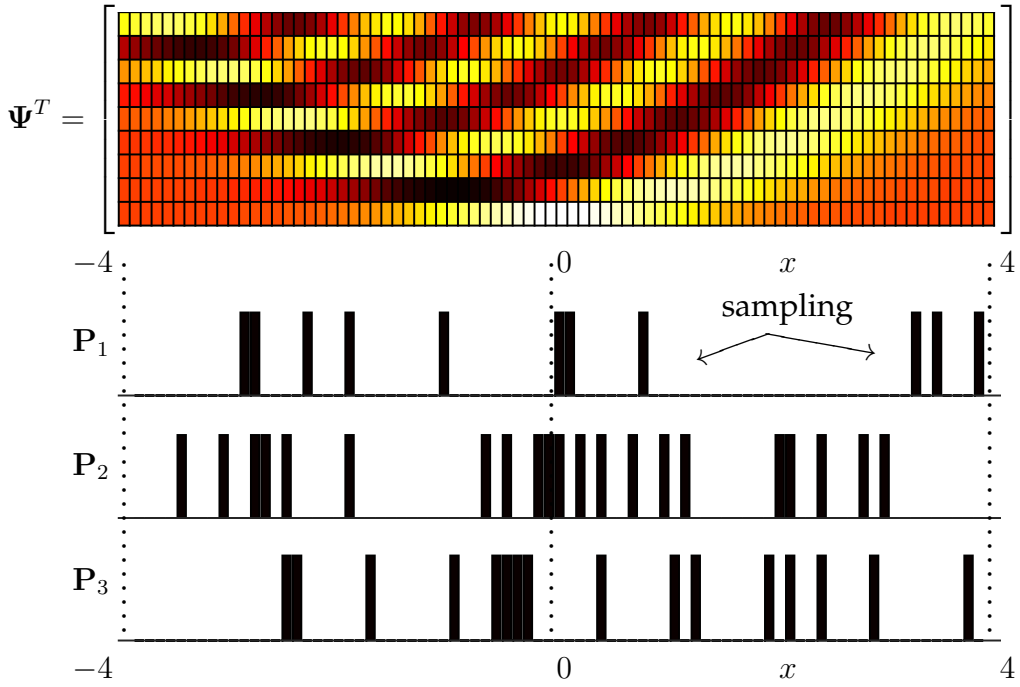


Figure 13.1: The top panel shows the first nine modes of the quantum harmonic oscillator considered in (12.25) and (12.26). Three randomly generated measurement matrices, $\mathbf{P}_j$, with $j = 1$, 2, and 3, are depicted. There is a 20% chance of performing a measurement at a given spatial location x_j in the interval $x \in [-4, 4]$ with a spacing of $\Delta x = 0.1$.

function is plotted in the top panel of Fig. 13.2. Note that the goal now is to reconstruct this function both with a low-rank projection onto the harmonic oscillator modes, and with a gappy reconstruction whereby only a sampling of the data is used, via the measurements $\mathbf{P}_j$. A test function is reconstructed in the 10-mode harmonic oscillator basis. Further, it builds the matrix $\mathbf{M}$ for the full-state measurements and computes its condition number.

Results of the low-rank and gappy reconstruction are shown in Fig. 13.2. The low-rank reconstruction is performed using the full measurements projected to the 10 leading harmonic oscillator modes. In this case, the inner product of the measurement matrix is given by (13.4a) and is approximately the identity. The fact that we are working on a limited domain $x \in [-4, 4]$ with a discretization step of $\Delta x = 0.1$ is what makes $\mathbf{M} \approx \mathbf{I}$ versus being exactly the identity. For the three different sparse measurement scenarios $\mathbf{P}_j$ of Fig. 13.1, the reconstruction is also shown along with the least-squares error and the logarithm of the condition number $\log(\kappa(\mathbf{M}_j))$. We also visualize the three matrices $\mathbf{M}_j$ in Fig. 13.3. The condition number of each of these matrices helps determine its reconstruction accuracy.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

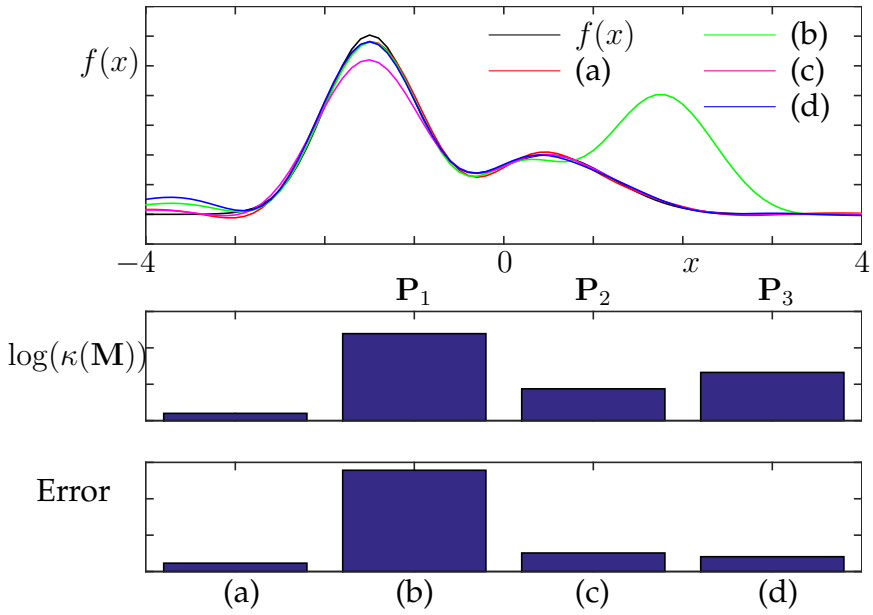


Figure 13.2: (top) The original function (black) along with a 10-mode reconstruction of the test function $f(x) = \exp[-(x-0.5)^2] + 3\exp[-2(x+3/2)^2]$ sampled in the full space ((a), red) and three representative support spaces $s[\tilde{\mathbf{u}}]$ of Fig. 13.1, specifically (b) $\mathbf{P}_1$, (c) $\mathbf{P}_2$, and (d) $\mathbf{P}_3$. Note that the error measurement is specific to the function being considered, whereas the condition number metric is independent of the specific function. Although both can serve as proxies for performance, the condition number serves for any function, which is advantageous.

13.2 Error and Convergence of Gappy POD

As was shown in the previous section, the ability of the gappy sampling strategy to accurately reconstruct a given function depends critically on the placement of the measurement (sensor) locations. Given the importance of this issue, we will discuss a variety of principled methods for placing a limited number of sensors in detail in subsequent sections. Our goal in this section is to investigate the convergence properties and error associated with the gappy method as a function of the percentage of sampling of the full system. Random sampling locations will be used.

Given our random sampling strategy, the results that follow will be statistical in nature, computing averages and variances for batches of randomly selected sampling. The modal basis for our numerical experiments are again the Gauss–Hermite functions defined by (12.25) and (12.26), and shown in the top panel of Fig. 13.1.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

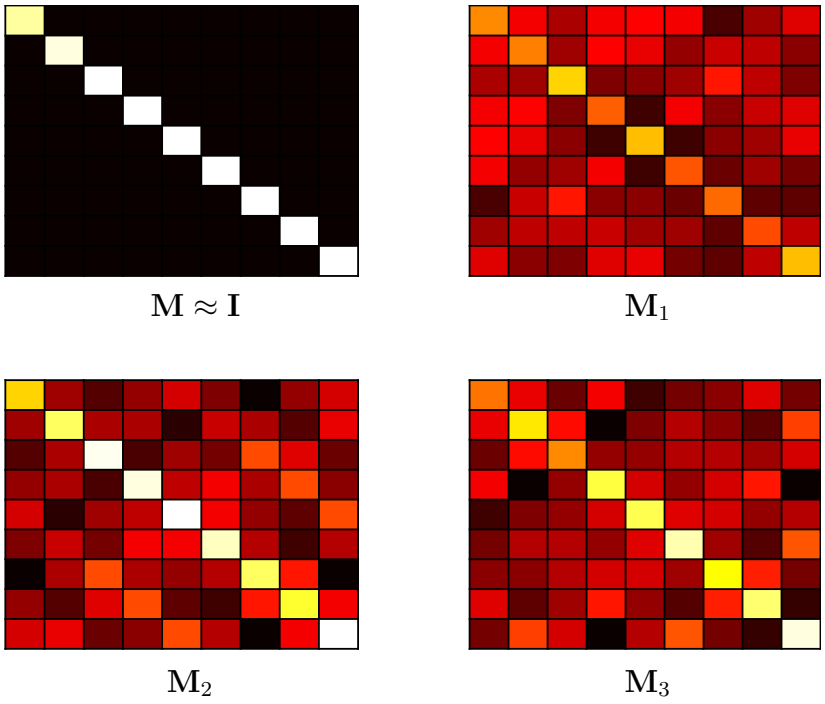


Figure 13.3: Demonstration of the deterioration of the orthogonality of the modal basis in the support space $s[\tilde{\mathbf{u}}]$ as given by the matrix $\mathbf{M}$ defined in (13.4). The top left shows that the identity matrix is produced for full measurements, or nearly so but with errors due to truncation of the domain over $x \in [-4, 4]$. The matrices $\mathbf{M}_j$, which no longer look diagonal, correspond to the sparse sampling matrices $\mathbf{P}_j$ in Fig. 13.1. Thus it is clear that the modes are not orthogonal in the support space of the measurements.

Random Sampling and Convergence

Our study begins with random sampling of the modes at a level of 10%, 20%, 30%, 40%, 50%, and 100%, respectively. The latter case represents the idealized full sampling of the system. As one would expect, the error and reconstruction are improved as more samples are taken. To show the convergence of the gappy sampling, we consider two error metrics: (i) the ℓ_2 error between our randomly subsampled reconstruction, and (ii) the condition number of the matrix $\mathbf{M}$ for a given measurement matrix $\mathbf{P}_j$. Recall that the condition number provides a way to measure the error without knowing the truth, i.e., (13.11).

Figure 13.4 depicts the average over 1000 trials of the logarithm of the least-squares error, $\log(E+1)$ (unity is added to avoid negative numbers), and the log of the condition number, $\log(\kappa(\mathbf{M}))$, as a function of percentage of random measurements. Also depicted is the variance σ, with the red bars denoting $\mu \pm \sigma$, where μ is the average value. The error and condition number both perform better as the number of samples increases. Note that the error does not ap-

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

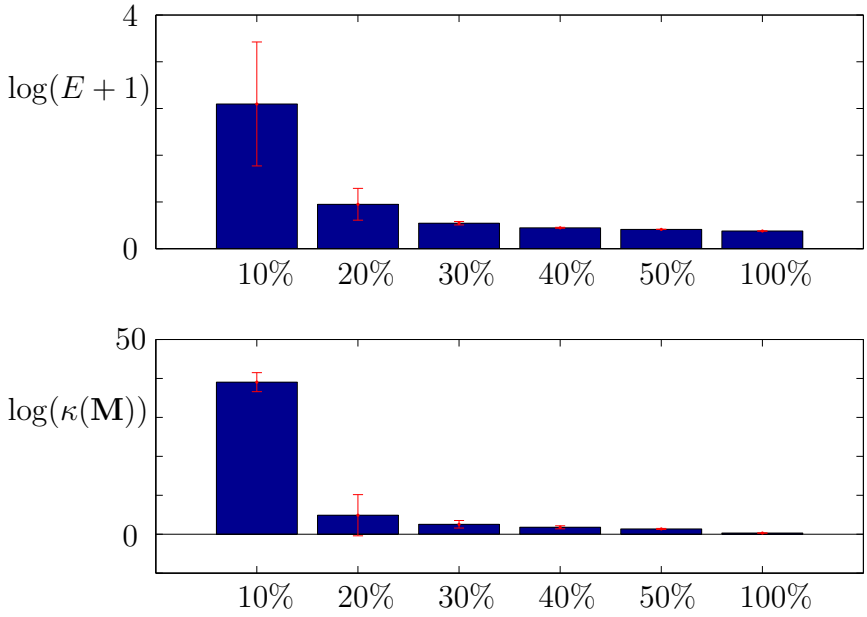


Figure 13.4: Logarithm of the least-squares error, $\log(E+1)$ (unity is added to avoid negative numbers), and the log of the condition number, $\log(\kappa(\mathbf{M}))$, as a function of percentage of random measurements. For 10% measurements, the error and condition number are largest, as expected. However, the variance of the results, depicted by the red bars, is also quite large, suggesting that the performance for a small number of sensors is highly sensitive to their placement.

proach zero since only a 10-mode basis expansion is used, thus limiting the accuracy of the POD expansion and reconstruction even with full measurements.

We draw over 1000 random sensor configurations (see Fig. 13.4) using 10%, 20%, 30%, 40%, and 50% sampling. The full reconstruction (100% sampling) is used to make the final graphic for Fig. 13.4. Note that, as expected, the error and condition number trends are similar, thus supporting the hypothesis that the condition number can be used to evaluate the efficacy of the sparse measurements. Indeed, this clearly shows that the condition number provides an evaluation that does not require knowledge of the function in (13.11).

Gappy Measurements and Performance

We can continue this statistical analysis of the gappy reconstruction method by looking more carefully at 200 random trials of 20% measurements. Figure 13.5 shows three key features of the 200 random trials. In particular, as shown in the top panel of this figure, there is a large variance in the distribution of the condition number $\kappa(\mathbf{M})$ for 20% sampling. Specifically, the condition number can change by orders of magnitude with the same number of sensors, but simply

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

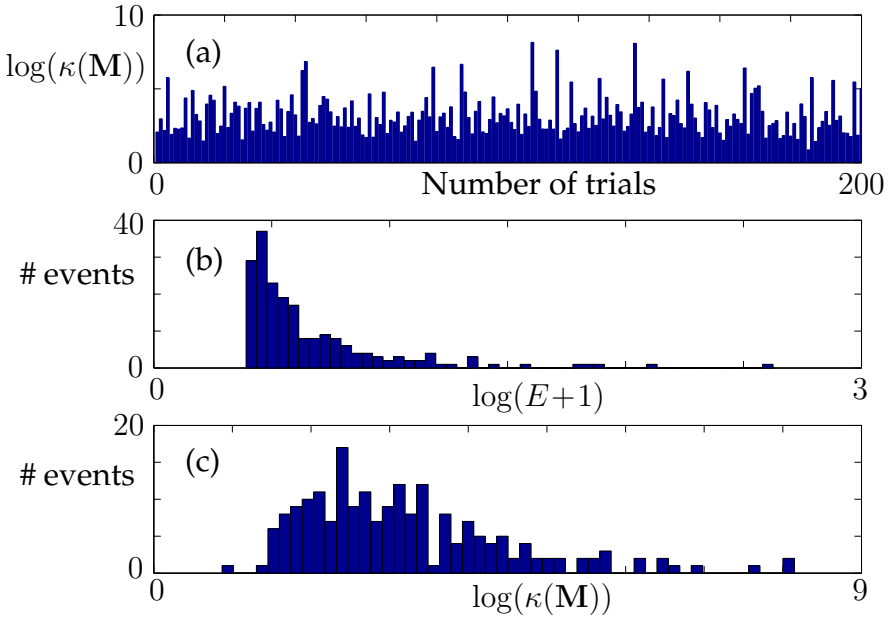


Figure 13.5: Statistics of 20% random measurements considered in Fig. 13.4. Panel (a) depicts 200 random trials and the condition number $\log(\kappa(\mathbf{M}))$ of each trial. Histograms of (b) the logarithm of the least-squares error, $\log(E+1)$, and (c) the condition number, $\log(\kappa(\mathbf{M}))$, are also depicted for the 200 trials. The panels illustrate the extremely high variability generated from the random, sparse measurements. In particular, 20% measurements can produce both exceptional results and extremely poor performance depending upon the measurement locations. The measurement vectors $\mathbf{P}$ that generate these statistics are depicted in Fig. 13.6.

placed in different locations. A histogram of the distribution of the log error $\log(E+1)$ and the log of the condition number are shown in the bottom two panels. The error appears to be distributed in an exponentially decaying fashion whereas the condition number distribution is closer to a Gaussian. There are distinct outliers whose errors and condition numbers are exceptionally high, suggesting sensor configurations to be avoided.

In order to visualize the random, gappy measurements of the 200 samples used in the statistical analysis of Fig. 13.5, we plot the $\mathbf{P}_j$ measurement masks in each row of the matrix in Fig. 13.6. The white regions represent regions where no measurements occur. The black regions are where the measurements are taken. These are the measurements that generate the orders-of-magnitude variance in the error and condition number.

As a final analysis, we can sift through the 200 random measurements of Fig. 13.6 and pick out both the 10 best and 10 worst measurement vectors $\mathbf{P}_j$. Figure 13.7 shows the results of this sifting process. The top two panels depict

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

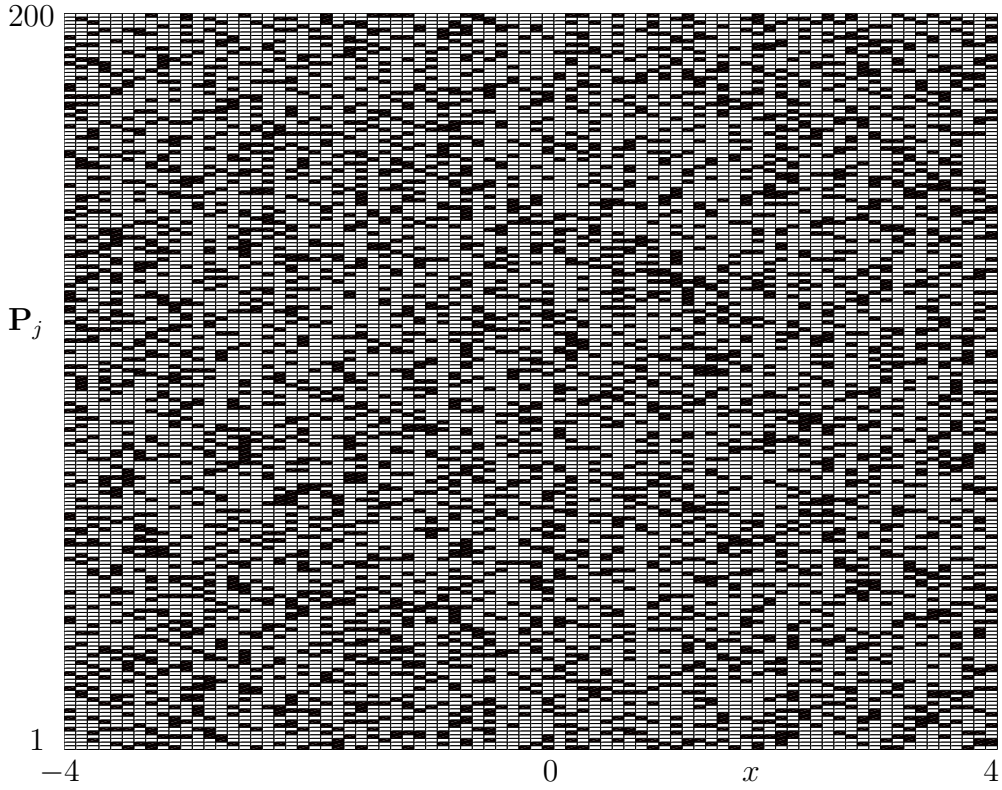


Figure 13.6: Depiction of the 200 random 20% measurement vectors $\mathbf{P}_j$ considered in Fig. 13.5. Each row is a randomly generated measurement trial (from 1 to 200) while the columns represent their spatial location on the domain $x \in [-4, 4]$ with $\Delta x = 0.1$.

the best and worst measurement configurations. Interestingly, the worst measurements have long stretches of missing measurements near the center of the domain where much of the modal variance occurs. In contrast, the best measurements have well-sampled domains with few long gaps between measurement locations. The bottom panel shows that the best measurements (on the left) offer an improvement of two orders of magnitude in the condition number over the poor-performing counterparts (on the right).

13.3 Gappy Measurements: Minimize Condition Number

The preceding section illustrates that the placement of gappy measurements is critical for accurately reconstructing the POD solution. This suggests that a principled way to determine measurement locations is of great importance. In

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

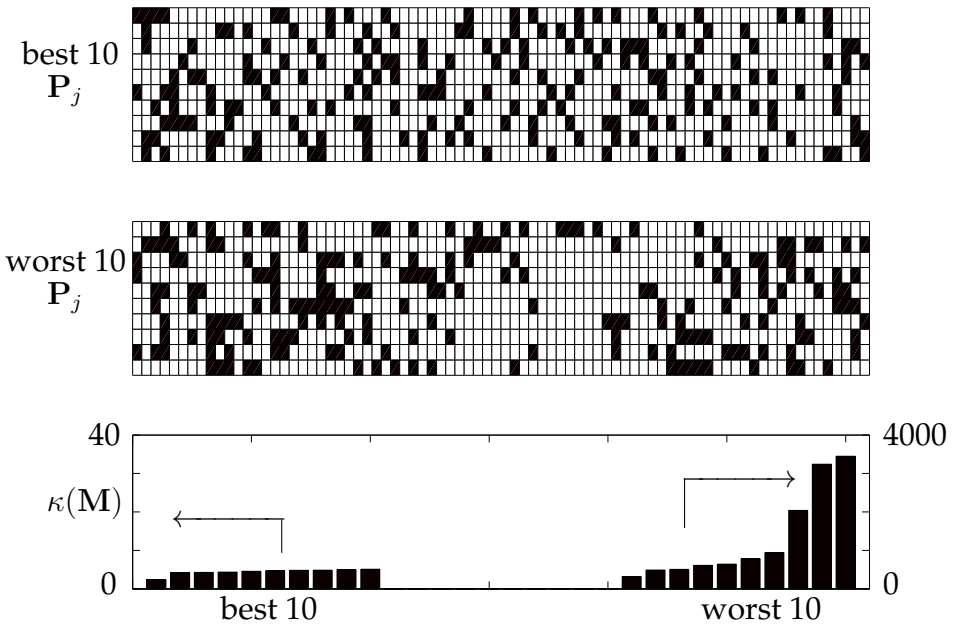


Figure 13.7: Depiction of the 10 best and 10 worst random 20% measurement vectors $\mathbf{P}_j$ considered in Figs. 13.5 and 13.6. The top panel shows that the best measurement vectors sample fairly uniformly across the domain $x \in [-4, 4]$ with $\Delta x = 0.1$. In contrast, the worst randomly generated measurements (middle panel) have large sampling gaps near the center of the domain, leading to a large condition number $\kappa(\mathbf{M})$. The bottom panel shows a bar chart of the best and worst values of the condition number. Note that with 20% sampling, there can be two orders of magnitude difference in the condition number, thus suggesting the importance of prescribing good measurement locations.

what follows, we outline a method originally proposed by Willcox [754] for assessing the gappy measurement locations. The method is based on minimizing the condition number $\kappa(\mathbf{M})$ in the placement process. As already shown, the condition number is a good proxy for evaluating the efficacy of the reconstruction. Moreover, it is a measure that is independent of any specific function.

The algorithm proposed [754] is computationally costly, but it can be performed in an offline training stage. Once the sensor locations are determined, they can be used for online reconstruction. The algorithm is as follows:

1. Place sensor k at each spatial location possible and evaluate the condition number $\kappa(\mathbf{M})$. Only points not already containing a sensor are considered.

2. Determine the spatial location that minimizes the condition number $\kappa(\mathbf{M})$. This spatial location is now the kth sensor location.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

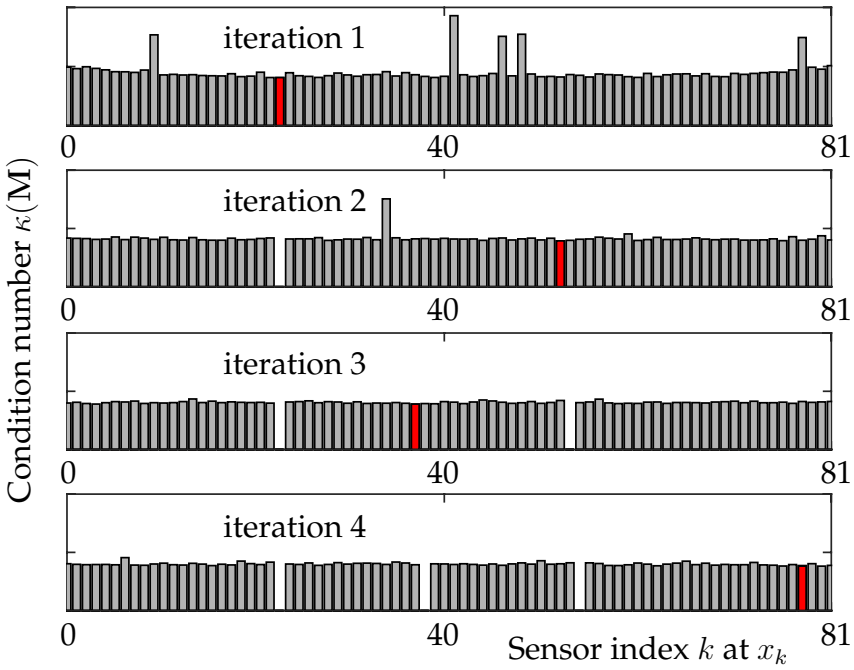


Figure 13.8: Depiction of the first four iterations of the gappy measurement location algorithm of Willcox [754]. The algorithm is applied to a 10-mode expansion given by the Gauss–Hermite functions (12.25) and (12.26) discretized on the interval $x \in [-4, 4]$ with $\Delta x = 0.1$. The top panel shows the condition number $\kappa(\mathbf{M})$ as a single sensor is considered at each of the 81 discrete values x_k. The first sensor minimizes the condition number (shown in red) at x_{23}. A second sensor is now considered at all remaining 80 spatial locations, with the minimal condition number occurring at x_{52} (in red). Repeating this process gives x_{37} and x_{77} for the third and fourth sensor locations for iterations 3 and 4 of the algorithm (highlighted in red). Once a location is selected for a sensor, it is no longer considered in future iterations. This is represented by a gap.

3. Add sensor $k + 1$ and repeat the previous two steps.

The algorithm is not optimal, nor is it guaranteed to be so. However, it works quite well in practice since sensor configurations with low condition number produce good reconstructions with the POD modes.

We apply this algorithm to construct the gappy measurement matrix $\mathbf{P}$. As before, the modal basis for our numerical experiments are the Gauss–Hermite functions defined by (12.25) and (12.26). The gappy measurement matrix algorithm for constructing $\mathbf{P}$ is shown in Fig. 13.8 – specifically, the first four iterations of the scheme. Note that the algorithm outlined above sets down one sensor at a time, thus with the 10-POD-mode expansion, the system is underdetermined until 10 sensors are placed. This gives condition numbers on the

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

order of 10^{16} for the first nine sensor placements. It also suggests that the first 10 sensor locations may be generated from inaccurate calculations of the condition number.

Using a 10-mode expansion of the Gauss–Hermite functions, we minimize the condition number and identify the first 20 sensor locations. Specifically, this provides a principled way of producing a measurement matrix $\mathbf{P}$ that allows for good reconstruction of the POD mode expansion with limited measurements. In addition to identifying the placement of the first 20 sensors, reconstruction of the example function given by (13.11) is computed at each iteration of the routine. Note the use of the **setdiff** command, which removes the condition number minimizing sensor location from consideration in the next iteration.

To evaluate the gappy sensor location algorithm, we track the condition number as a function of the number of iterations, up to 20 sensors. Additionally, at each iteration, a reconstruction of the test function (13.11) is computed and a least-squares error evaluated. Figure 13.9 shows the progress of the algorithm as it evaluates the sensor locations for up to 20 sensors. By construction, the algorithm minimizes the condition number $\kappa(\mathbf{M})$ at each step of the iteration; thus, as sensors are added, the condition number steadily decreases (top panel of Fig. 13.9). Note that there is a significant decrease in the condition number once 10 sensors are selected, since the system is no longer under-determined with theoretically infinite condition number. The least-squares error for the reconstruction of the test function (13.11) follows the same general trend, but the error does not monotonically decrease like the condition number. The least-squares error also makes a significant improvement once 10 measurements are made. In general, if an r-mode POD expansion is to be considered, then reasonable results using the gappy reconstruction cannot be achieved until r sensors are placed.

We now consider the placement of the sensors as a function of iteration in the bottom panel of Fig. 13.9. Specifically, we depict when sensors are identified in the iteration. The first sensor location is x_{23} followed by x_{52}, x_{37}, and x_{77}, respectively. The process is continued until the first 20 sensors are identified. The pattern of sensors depicted is important, as it illustrates a fairly uniform sampling of the domain. Alternative schemes will be considered in the following.

As a final illustration of the gappy algorithm, we consider the reconstruction of the test function (13.11) as the number of iterations (sensors) increases. As expected, the more sensors that are used in the gappy framework, the better the reconstruction is, especially if the sensors are placed in a principled way as outlined by Willcox [754]. Figure 13.10 shows the reconstructed function with increasing iteration number. In the left panel, iterations 1–20 are shown with the z-axis set to illustrate the extremely poor reconstruction in the early stages

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

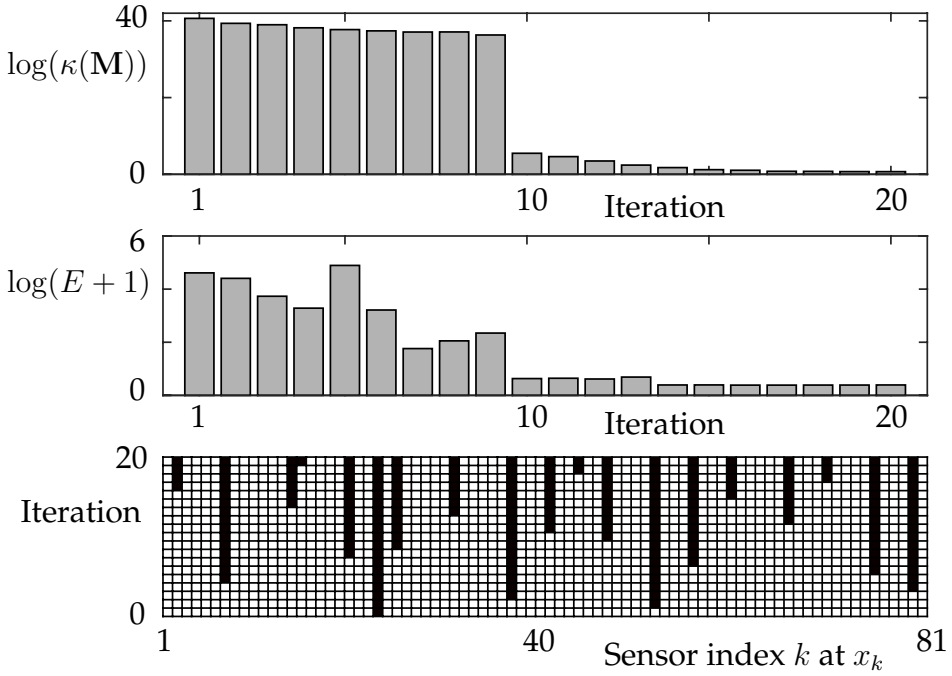


Figure 13.9: Condition number and least-squares error (logarithms) as a function of the number of iterations in the gappy sensor placement algorithm. The log of the condition number $\log(\kappa(\mathbf{M}))$ monotonically decreases, since this is being minimized at each iteration step. The log of the least-squares error in the reconstruction of the test function (13.11) also shows a trend towards improvement as the number of sensors are increased. Once 10 sensors are placed, the system is of full rank and the condition number drops by orders of magnitude. The bottom panel shows the sensors as they turn on (black squares) over the first 20 iterations. The first measurement location is, for instance, at x_{23}.

of the iteration. The right panel highlights the reconstruction from iteration 9 to 20, and on a more limited z-axis scale, where the reconstruction converges to the test function. The true test function is also shown in order to visualize the comparison. This illustrates in a tangible way the convergence of the iteration algorithm to the test solution with a principled placement of sensors.

Proxy Measures to the Condition Number

We end this section by considering alternative measures to the condition number $\kappa(\mathbf{M})$. The computation of the condition number itself can be computationally expensive. Moreover, until r sensors are chosen in an r-POD-mode expansion, the condition number computation is itself numerically unstable. However, it is clear what the condition-number-minimization algorithm is trying to achieve: make the measurement matrix $\mathbf{M}$ as near to the identity as possible.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

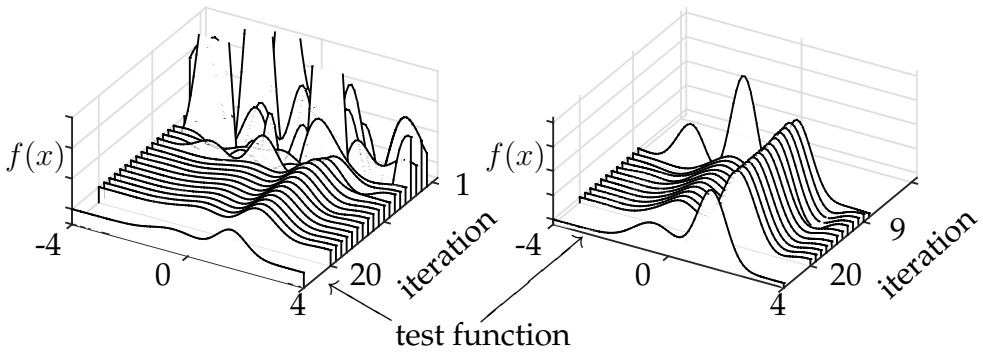


Figure 13.10: Convergence of the reconstruction to the test function (13.11). The left panel shows iterations 1–20 and the significant reconstruction errors of the early iterations and limited number of sensors. Indeed, for the first nine iterations, the condition number and least-squares error are quite large since the system is not full rank. The right panel shows a zoom-in of the solution from iteration 9 to 20 where the convergence is clearly observed. Comparison in both panels can be made to the test function.

This suggests the following alternative algorithm, which was also developed by Willcox [754].

1. Place sensor k at each spatial location possible and evaluate the difference in the sum of the diagonal entries of the matrix $\mathbf{M}$ minus the sum of the off-diagonal components; call this $\kappa_2(\mathbf{M})$. Only points not already containing a sensor are considered.

2. Determine the spatial location that generates the maximum value of the above quantity. This spatial location is now the kth sensor location.

3. Add sensor $k+1$ and repeat the previous two steps.

Modification of two lines of code can enact a new metric which circumvents the computation of the condition number.

To evaluate this new gappy sensor location algorithm, we track the new proxy metric we are trying to maximize as a function of the number of iterations along with the least-squares error of our test function (13.11). In this case, up to 60 sensors are considered, since the convergence is slower than before. Figure 13.11 shows the progress of the algorithm as it evaluates the sensor locations for up to 60 sensors. By construction, the algorithm maximizes the sum of the diagonals minus the sum of the off-diagonals at each step of the iteration; thus, as sensors are added, this measure steadily increases (top left panel of Fig. 13.11). The least-squares error for the reconstruction of the test function (13.11) decreases, but not monotonically. Further, the convergence is very slow.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

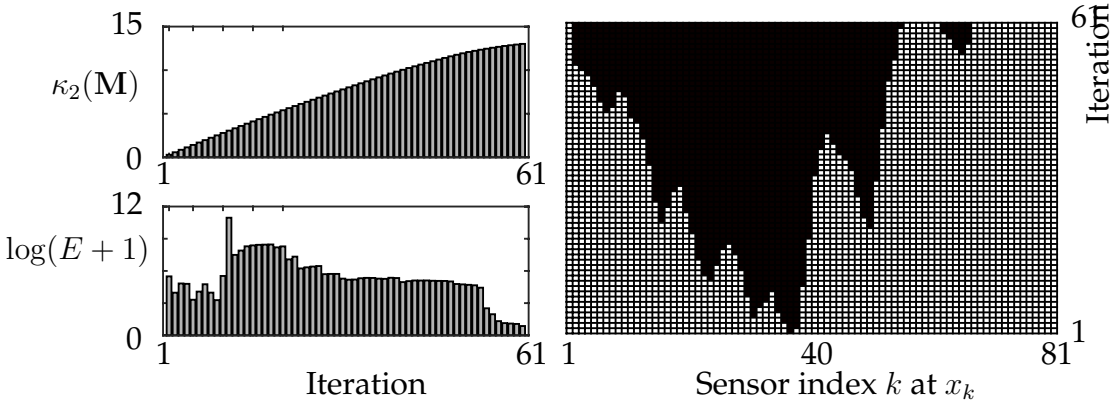


Figure 13.11: Sum of diagonals minus off-diagonals (top left) and least-squares error (logarithm) as a function of the number of iterations in the second gappy sensor placement algorithm. The new proxy metric for condition number monotonically increases, since this is being maximized at each iteration step. The log of the least-squares error in the reconstruction of the test function (13.11) shows a trend towards improvement as the number of sensors is increased, but convergence is extremely slow in comparison to minimizing the condition number. The right panel shows the sensors as they turn on (black squares) over the first 60 iterations. The first measurement location is, for instance, at x_{37}.

At least for this example, the method does not work as well as the condition number metric. However, it can improve performance in certain cases [754], and it is much more computationally efficient to compute.

As before, we also consider the placement of the sensors as a function of iteration in the right panel of Fig. 13.11. Specifically, we depict the turning on process of the sensors. The first sensor location is x_{37} followed by x_{38}, x_{36}, and x_{31}, respectively. The process is continued until the first 60 sensors are turned on. The pattern of sensors depicted is significantly different than in the condition-number-minimization algorithm. Indeed, this algorithm, and with these modes, turns on sensors in local locations without sampling uniformly from the domain.

13.4 Gappy Measurements: Maximal Variance

The previous section developed principled ways to determine the location of sensors for gappy POD measurements. This was a significant improvement over simply choosing sensor locations randomly. Indeed, the minimization of the condition number through location selection performed quite well, quickly

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

improving accuracy and least-squares reconstruction error. The drawback to the proposed method was two-fold: Firstly, the algorithm itself is expensive to implement, requiring a computation of the condition number for every sensor location selected under an exhaustive search. Secondly, the algorithm was ill-conditioned until the rth sensor was chosen in an r-POD-mode expansion. Thus the condition number was theoretically infinite, but on the order of 10^{17} for computational purposes.

Karniadakis and co-workers [767] proposed an alternative to the Willcox [754] algorithm to overcome the computational issues outlined. Specifically, instead of placing one sensor at a time, the new algorithm places r sensors, for an r-POD-mode expansion, at the first step of the iteration. Thus the matrix generated is no longer ill-conditioned with a theoretically infinite condition number.

The algorithm by Karniadakis further proposes a principled way to select the original r sensor locations. This method selects locations that are extrema points of the POD modes, which are designed to maximally capture variance in the data. Specifically, the following algorithm is suggested:

1. Place r sensors initially.

2. Determine the spatial locations of these first r sensors by considering the maximum of each of the POD modes $\boldsymbol{\psi}_k$.

3. Add additional sensors at the next largest extrema of the POD modes.

The performance of this algorithm is not strong for only r measurements, but it at least produces stable condition number calculations. To improve performance, one could also use the minimum of each of the modes $\boldsymbol{\psi}_k$. Thus the maximal value and minimal value of variance are considered. For the harmonic oscillator code, the first mode produces no minimum, as the minima are at $x \to \pm\infty$.

More generally, the Karniadakis algorithm [767] advocates randomly selecting p sensors from M potential extrema, and then modifying the search positions with the goal of improving the condition number. In this case, one must identify all the maxima and minima of the POD modes in order to make the selection. The harmonic oscillator modes and their maxima and minima are illustrated in Fig. 13.12.

In this example, there are 55 possible extrema. This computation assumes the data is sufficiently smooth so that extrema are simply found by considering neighboring points, i.e., a maximum exists if its two neighbors have a lower value, whereas a minimum exists if its neighbors have a higher value.

The maximal-variance algorithm suggests trying different configurations of the sensors at the extrema points. In particular, if 20 gappy measurements are desired, then we would need to search through various configurations of the 55

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

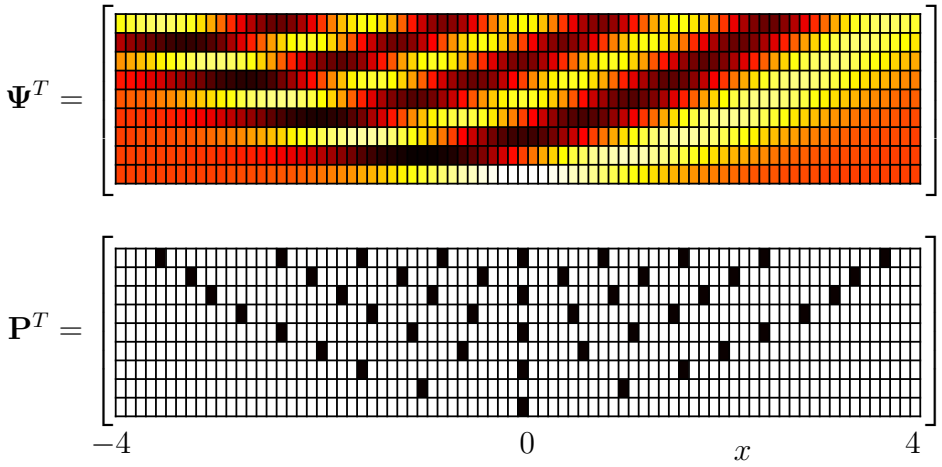


Figure 13.12: The top panel shows the mode structures of the Gauss–Hermite polynomials $\mathbf{\Psi}$ in the low-rank approximation of a POD expansion. The discretization interval is $x \in [-4, 4]$ with a spacing of $\Delta x = 0.1$. The color map shows the maximum (white) and minimum (black) that occur in the mode structures. The bottom panel shows the grid cells corresponding to maxima and minima (extrema) of POD mode variance. The extrema are candidates for sensor locations, or the measurement matrix $\mathbf{P}$, since they represent maximal variance locations. Typically, one would take a random subsample of these extrema to begin the evaluation of the gappy placement.

locations using 20 sensors. This combinatorial search is intractable. However, if we simply attempt 100 random trials and select the best-performing configuration, it is quite close to the performance of the condition-number-minimization algorithm. A full execution of this algorithm, along with a computation of the condition number and least-squares fit error with (13.11), is generated. The condition number and least-squares error for the 100 trials are shown in Fig. 13.13. The configurations perform well compared with random measurements, although some have excellent performance.

A direct comparison of all these methods is shown in Fig. 13.14. Specifically, what is illustrated are the results from using (a) the maximum locations of the POD modes, (b) the maximum and minimum locations of each POD mode, and (c) a random selection of 20 of the 55 extremum locations of the POD modes. These are compared against (d) the best five sensor placement locations of 20 sensors selected from the extremum over 100 random trials, and (e) the condition-number-minimization algorithm (in red). The maximal-variance algorithm performs approximately as well as the condition-number-minimization algorithm. However, the algorithm is faster and never computes condition numbers on ill-conditioned matrices. Karniadakis and co-workers [767] also suggest innovations on this basic implementation. Specifically, it is

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

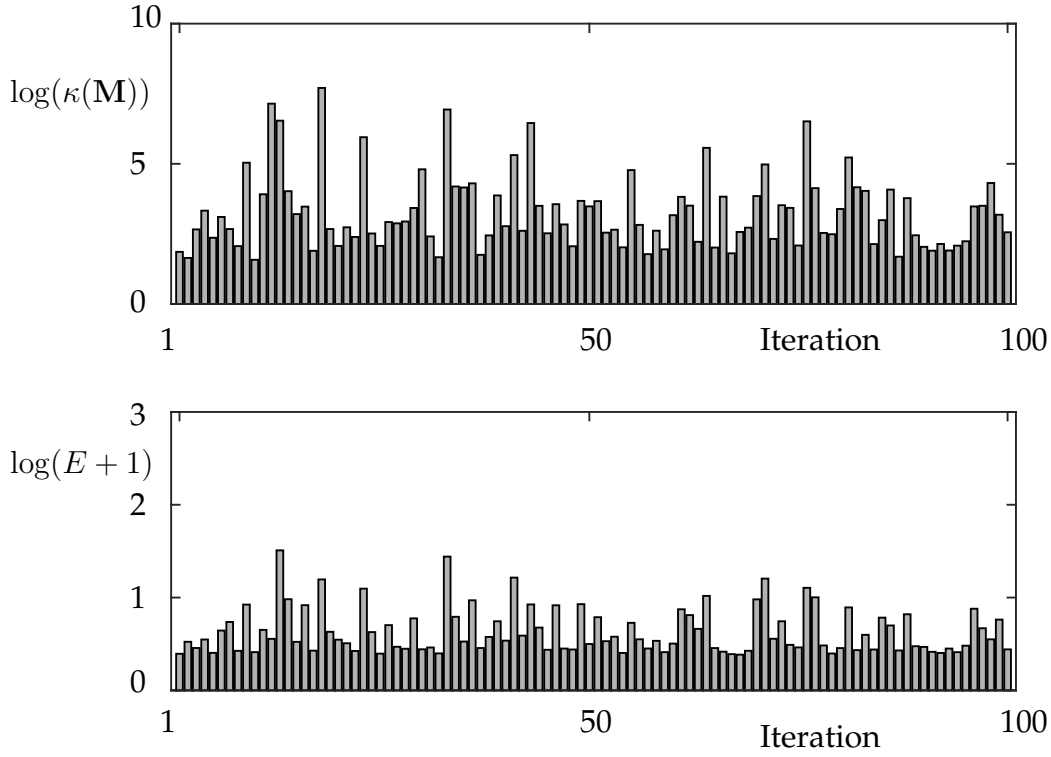


Figure 13.13: Condition number and least-squares error to test function (13.11) over 100 random trials that draw 20 sensor locations from the possible 55 extrema depicted in Fig. 13.12. The 100 trials produce a number of sensor configurations that perform close to the level of the condition-number-minimization algorithm of the last section. However, the computational costs in generating such trials can be significantly lower.

suggested that one consider each sensor, one-by-one, and try placing it in all other available spatial locations. If the condition number is reduced, the sensor is moved to that new location and the next sensor is considered.

13.5 POD and the Discrete Empirical Interpolation Method (DEIM)

The POD method illustrated thus far aims to exploit the underlying low-dimensional dynamics observed in many high-dimensional computations. POD is often used for reduced-order models (ROMs), which are of growing importance in scientific applications and computing. ROMs reduce the computational complexity and time needed to solve large-scale, complex systems [24, 75, 325, 578]. Specifically, ROMs provide a principled approach to approximating high-dimensional spatio-temporal systems [185], typically generated from numerical discretiza-

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

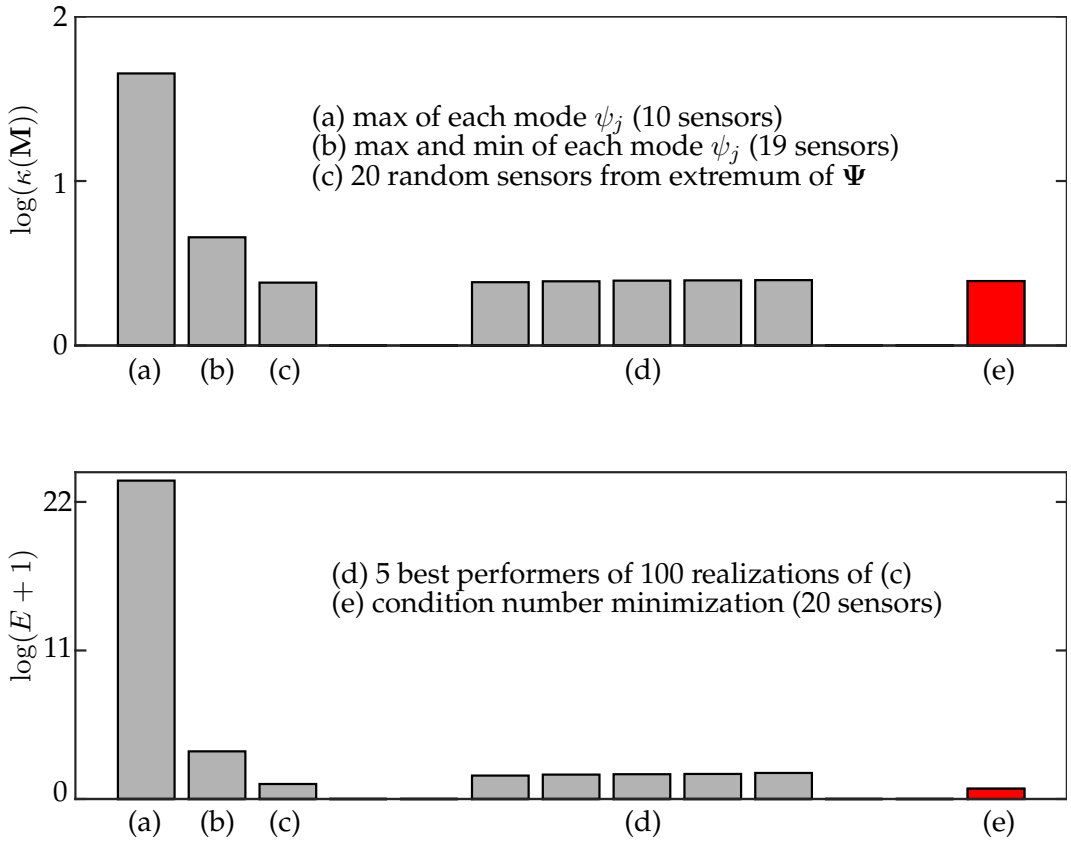


Figure 13.14: Performance metrics for placing sensors based upon the extrema of the variance of the POD modes. Both the least-squares error for the reconstruction of the test function (13.11) and the condition number are considered. Illustrated are the results from using (a) the maximum locations of the POD modes, (b) the maximum and minimum locations of each POD mode, and (c) a random selection of 20 of the 55 extremum locations of the POD modes. These are compared against (d) the five top selections of 20 sensors from the 100 random trials, and (e) the condition-number-minimization algorithm (red bar). The random placement of sensors from the extremum locations provides performance close to that of the condition number minimization without the same high computational costs.

tion, by low-dimensional subspaces that produce nearly identical input/output characteristics of the underlying nonlinear dynamical system. However, despite the significant reduction in dimensionality with a POD basis, the complexity of evaluating higher-order nonlinear terms may remain as challenging as the original problem [55, 171]. The empirical interpolation method (EIM) and the simplified discrete empirical interpolation method (DEIM) for the proper orthogonal decomposition (POD) [335, 463] overcome this difficulty by providing a computationally efficient method for discretely (sparsely) sampling and evaluating the nonlinearity. These methods ensure that the computational com-

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

plexity of ROMs scale favorably with the rank of the approximation, even with complex nonlinearities.

EIM has been developed for the purpose of efficiently managing the computation of the nonlinearity in dimensionality reduction schemes, with DEIM specifically tailored to POD with Galerkin projection. Indeed, DEIM approximates the nonlinearity by using a small, discrete sampling of points that are determined in an algorithmic way. This ensures that the computational cost of evaluating the nonlinearity scales with the rank of the reduced POD basis. As an example, consider the case of an r-mode Galerkin–POD truncation. A simple cubic nonlinearity requires that the Galerkin–POD approximation be cubed, resulting in r^3 operations to evaluate the nonlinear term. DEIM approximates the cubic nonlinearity by using $\mathcal{O}(r)$ discrete sample points of the nonlinearity, thus preserving a low-dimensional ($\mathcal{O}(r)$) computation, as desired. The DEIM approach combines projection with interpolation. Specifically, DEIM uses selected interpolation indices to specify an interpolation-based projection for a nearly ℓ_2 optimal subspace approximating the nonlinearity. EIM/DEIM are not the only methods developed to reduce the complexity of evaluating nonlinear terms; see for instance the missing point estimation (MPE) [29, 522] or gappy POD [162, 619, 754, 767] methods. However, they have been successful in a large number of diverse applications and models [171]. In any case, the MPE, gappy POD, and EIM/DEIM use a small selected set of spatial grid points to avoid evaluation of the expensive inner products required to evaluate nonlinear terms.

POD and DEIM

Consider a high-dimensional system of nonlinear differential equations that can arise, for example, from the finite-difference discretization of a partial differential equation. In addition to constructing a snapshot matrix (13.12) of the solution of the PDE so that POD modes can be extracted, the DEIM algorithm also constructs a snapshot matrix of the nonlinear term of the PDE:

$$\mathbf{N} = \begin{bmatrix} | & | & & | \\ \mathbf{N}_1 & \mathbf{N}_2 & \cdots & \mathbf{N}_m \\ | & | & & | \end{bmatrix}, \tag{13.12}$$

where the columns $\mathbf{N}_k \in \mathbb{C}^n$ are evaluations of the nonlinearity at time t_k.

To achieve high-accuracy solutions, n is typically very large, making the computation of the solution expensive and/or intractable. The Galerkin–POD method is a principled dimensionality reduction scheme that approximates the function $\mathbf{u}(t)$ with rank-r optimal basis functions, where $r \ll n$. As shown in the previous chapter, these optimal basis functions are computed from a singular value decomposition of a series of temporal snapshots of the complex system.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

The standard POD procedure [335] is a ubiquitous algorithm in the reduced-order modeling community. However, it also helps illustrate the need for innovations such as DEIM, gappy POD, and/or MPE. Consider the nonlinear component of the low-dimensional evolution (12.21): $\boldsymbol{\Psi}^T\mathbf{N}(\boldsymbol{\Psi}\mathbf{a}(t))$. For a simple nonlinearity such as $N(u(x,t)) = u(x,t)^3$, consider its impact on a spatially discretized, two-mode POD expansion: $u(x,t) = a_1(t)\psi_1(x) + a_2(t)\psi_2(x)$. The algorithm for computing the nonlinearity requires the evaluation:

$$u(x,t)^3 = a_1^3\psi_1^3 + 3a_1^2a_2\psi_1^2\psi_2 + 3a_1a_2^2\psi_1\psi_2^2 + a_2^3\psi_2^3. \tag{13.13}$$

The dynamics of $a_1(t)$ and $a_2(t)$ would then be computed by projecting onto the low-dimensional basis by taking the inner product of this nonlinear term with respect to both ψ_1 and ψ_2. Thus not only does the number of computations double, but also the inner products must be computed with the n-dimensional vectors. Methods such as DEIM overcome this high-dimensional computation. Figure 13.15 gives an overview of the algorithm that is detailed below.

DEIM

As outlined in the previous section, the shortcomings of the Galerkin–POD method are generally due to the evaluation of the nonlinear term $\mathbf{N}(\boldsymbol{\Psi}\mathbf{a}(t))$. To avoid this difficulty, DEIM approximates $\mathbf{N}(\boldsymbol{\Psi}\mathbf{a}(t))$ through projection and interpolation instead of evaluating it directly. Specifically, a low-rank representation of the nonlinearity is computed from the singular value decomposition,

$$\mathbf{N} = \boldsymbol{\Xi}\boldsymbol{\Sigma}_{\mathbf{N}}\mathbf{V}_{\mathbf{N}}^*, \tag{13.14}$$

where the matrix $\boldsymbol{\Xi}$ contains the optimal basis for spanning the nonlinearity. Specifically, we consider the rank-p basis

$$\boldsymbol{\Xi}_p = [\boldsymbol{\xi}_1\ \boldsymbol{\xi}_2\ \cdots\ \boldsymbol{\xi}_p] \tag{13.15}$$

that approximates the nonlinear function ($p \ll n$ and $p \sim r$). The approximation to the nonlinearity $\mathbf{N}$ is given by

$$\mathbf{N} \approx \boldsymbol{\Xi}_p\mathbf{c}(t), \tag{13.16}$$

where $\mathbf{c}(t)$ is similar to $\mathbf{a}(t)$ in (12.20). Since this is a highly over-determined system, a suitable vector $\mathbf{c}(t)$ can be found by selecting p rows of the system. The DEIM algorithm was developed to identify which p rows to evaluate.

The DEIM algorithm begins by considering the vectors $\mathbf{e}_{\gamma_j} \in \mathbf{R}^n$, which are the γ_jth column of the n-dimensional identity matrix. We can then construct the projection matrix $\mathbf{P} = [\mathbf{e}_{\gamma_1}\ \mathbf{e}_{\gamma_2}\ \cdots\ \mathbf{e}_{\gamma_p}]$, which is chosen so that $\mathbf{P}^T\boldsymbol{\Xi}_p$ is non-singular. Then $\mathbf{c}(t)$ is uniquely defined from $\mathbf{P}^T\mathbf{N} = \mathbf{P}^T\boldsymbol{\Xi}_p\mathbf{c}(t)$, and thus

$$\mathbf{N} \approx \boldsymbol{\Xi}_p(\mathbf{P}^T\boldsymbol{\Xi}_p)^{-1}\mathbf{P}^T\mathbf{N}. \tag{13.17}$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

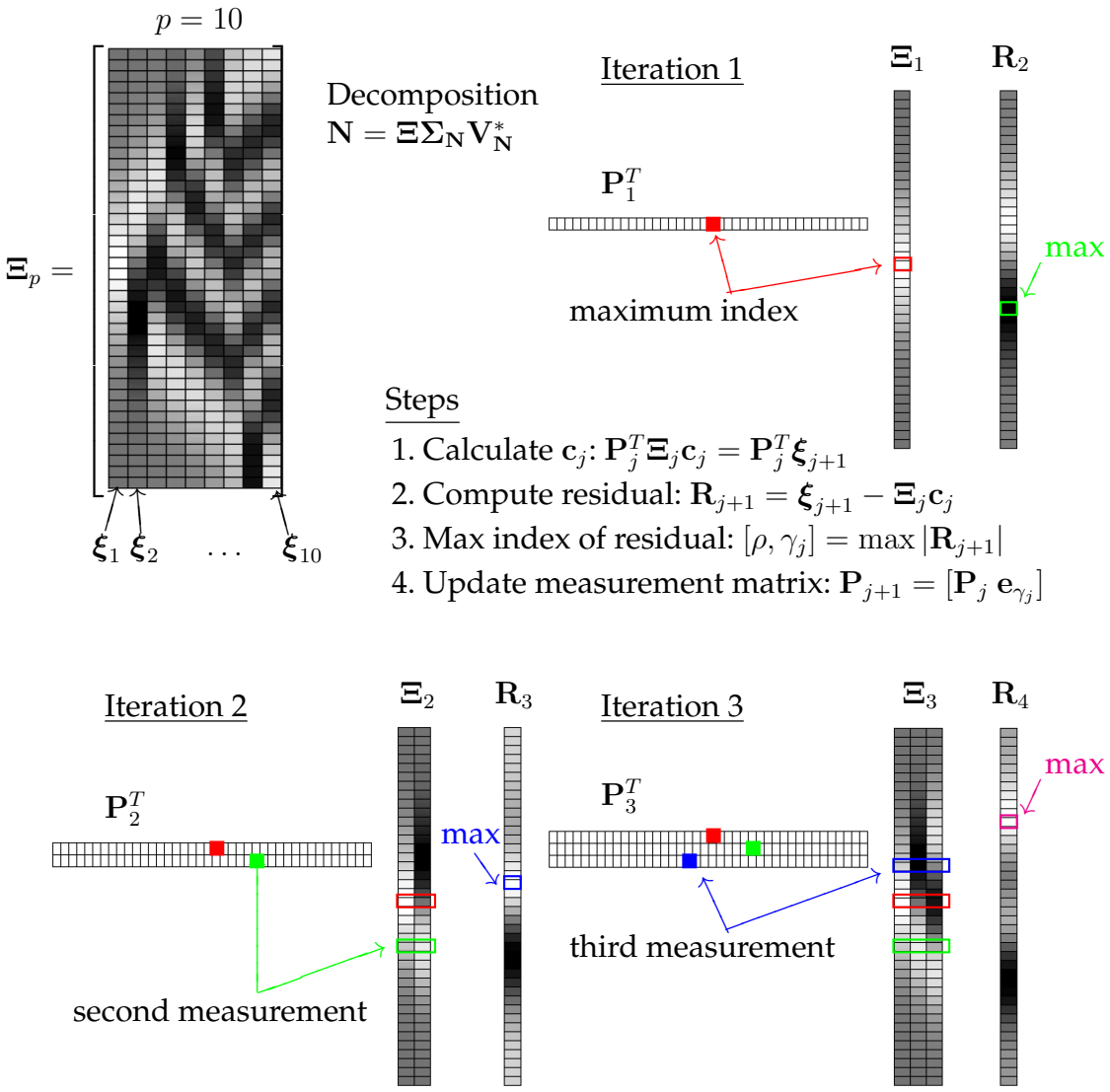


Figure 13.15: Demonstration of the first three iterations of the DEIM algorithm. For illustration only, the nonlinearity matrix $\mathbf{N} = \boldsymbol{\Xi}\boldsymbol{\Sigma}_\mathbf{N}\mathbf{V}_\mathbf{N}^*$ is assumed to be composed of harmonic oscillator modes with the first 10 modes comprising $\boldsymbol{\Xi}_p$. The initial measurement location is chosen at the maximum of the first mode $\boldsymbol{\xi}_1$. Afterwards, there is a three-step process for selecting subsequent measurement locations based upon the location of the maximum of the residual vector $\mathbf{R}_j$. The first (red), second (green), and third (blue) measurement locations are shown along with the construction of the sampling matrix $\mathbf{P}$.

The tremendous advantage of this result for nonlinear model reduction is that the term $\mathbf{P}^T\mathbf{N}$ requires evaluation of the nonlinearity only at $p \ll n$ indices. DEIM further proposes a principled method for choosing the basis vectors $\boldsymbol{\xi}_j$ and indices γ_j. The DEIM algorithm, which is based on a greedy search, is de-

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Table 13.1: DEIM algorithm for finding approximation basis for the nonlinearity and its interpolation indices. The algorithm first constructs the nonlinear basis modes and initializes the first measurement location, and the matrix $\mathbf{P}_1$, as the maximum of $\boldsymbol{\xi}_1$. The algorithm then successively constructs columns of $\mathbf{P}_j$ by considering the location of the maximum of the residual $\mathbf{R}_j$.

DEIM algorithm	
Basis construction and initialization	
collect data, construct snapshot matrix	$\mathbf{X} = [\mathbf{u}(t_1)\ \mathbf{u}(t_2)\ \cdots\ \mathbf{u}(t_m)]$
construct nonlinear snapshot matrix	$\mathbf{N} = [N(\mathbf{u}(t_1))\ N(\mathbf{u}(t_2))\ \cdots\ N(\mathbf{u}(t_m))]$
singular value decomposition of $\mathbf{N}$	$\mathbf{N} = \boldsymbol{\Xi}\boldsymbol{\Sigma}_\mathbf{N}\mathbf{V}_\mathbf{N}^*$
construct rank-p approximating basis	$\boldsymbol{\Xi}_p = [\boldsymbol{\xi}_1\ \boldsymbol{\xi}_2\ \cdots\ \boldsymbol{\xi}_p]$
choose the first index (initialization)	$[\rho, \gamma_1] = \max \lvert\boldsymbol{\xi}_1\rvert$
construct first measurement matrix	$\mathbf{P}_1 = [\mathbf{e}_{\gamma_1}]$
Interpolation indices and iteration loop ($j = 2, 3, \ldots, p$)	
calculate $\mathbf{c}_j$	$\mathbf{P}_j^T\boldsymbol{\Xi}_j\mathbf{c}_j = \mathbf{P}_j^T\boldsymbol{\xi}_{j+1}$
compute residual	$\mathbf{R}_{j+1} = \boldsymbol{\xi}_{j+1} - \boldsymbol{\Xi}_j\mathbf{c}_j$
find index of maximum residual	$[\rho, \gamma_j] = \max \lvert\mathbf{R}_{j+1}\rvert$
add new column to measurement matrix	$\mathbf{P}_{j+1} = [\mathbf{P}_j\ \mathbf{e}_{\gamma_j}]$

tailed in [171] and further demonstrated in Table 13.1.

POD and DEIM provide a number of advantages for nonlinear model reduction of complex systems. POD provides a principled way to construct an r-dimensional subspace $\boldsymbol{\Psi}$ characterizing the dynamics. DEIM augments POD by providing a method to evaluate the problematic nonlinear terms using a p-dimensional subspace $\boldsymbol{\Xi}_p$ that represents the nonlinearity. Thus a small number of points can be sampled to approximate the nonlinear terms in the ROM.

13.6 DEIM Algorithm Implementation

To demonstrate model reduction with DEIM, we again consider the NLS equation (12.29). Specifically, the data set considered is a matrix whose rows represent the time snapshots and whose columns represent the spatial discretization points. As in the first section of this chapter, our first step is to transpose this data so that the time snapshots are columns instead of rows. The following code transposes the data and also performs a singular value decomposition to get the POD modes.

Code 13.1: [MATLAB] Dimensionality reduction for NLS.

```
X=usol.';  % data matrix X
[U,S,W]=svd(X,0);  % SVD reduction
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Code 13.1: [Python] Dimensionality reduction for NLS.

```
X = usol.T # data matrix X
U,S,WT = np.linalg.svd(X,full_matrices=0) # SVD reduction
```

In addition to the standard POD modes, the singular value decomposition of the nonlinear term is also required for the DEIM algorithm. This computes the low-rank representation of $N(u) = |u|^2 u$ directly as $\mathbf{N} = \mathbf{\Xi}\mathbf{\Sigma}_\mathbf{N}\mathbf{V}_\mathbf{N}^*$.

Code 13.2: [MATLAB] Dimensionality reduction for nonlinearity of NLS.

```
NL=i*(abs(X).^2).*X;
[XI,S_NL,W]=svd(NL,0);
```

Code 13.2: [Python] Dimensionality reduction for nonlinearity of NLS.

```
NL = (1j)*np.power(np.abs(X),2)*X
XI,S_NL,WT = np.linalg.svd(NL,full_matrices=0)
```

Once the low-rank structures are computed, the rank of the system is chosen with the parameter r. In what follows, we choose $r = p = 3$ so that both the standard POD modes and nonlinear modes, $\mathbf{\Psi}$ and $\mathbf{\Xi}_p$, have three columns each. The following code selects the POD modes for $\mathbf{\Psi}$ and projects the initial condition onto the POD subspace.

Code 13.3: [MATLAB] Rank selection and POD modes.

```
r=3;  % select rank truncation
Psi=U(:,1:r); % select POD modes
a=Psi'*u0; % project initial conditions
```

Code 13.3: [Python] Rank selection and POD modes.

```
r = 3 # select rank truncation
Psi = U[:,:r] # select POD modes
a0 = Psi.T @ u0 # project initial conditions
```

We now build the interpolation matrix $\mathbf{P}$ by executing the DEIM algorithm outlined in the last section. The algorithm starts by selecting the first interpolation point from the maximum of the first most dominant mode of $\mathbf{\Xi}_p$.

Code 13.4: [MATLAB] First DEIM point.

```
[Xi_max,nmax]=max(abs(XI(:,1)));
XI_m=XI(:,1);
z=zeros(n,1);
P=z; P(nmax)=1;
```

Code 13.4: [Python] First DEIM point.

```
nmax = np.argmax(np.abs(XI[:,0]))
XI_m = XI[:,0].reshape(n,1)
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

```
z = np.zeros((n,1))
P = np.copy(z)
P[nmax] = 1
```

The algorithm iteratively builds $\mathbf{P}$ one column at a time. The next step of the algorithm is to compute the second to rth interpolation point via the greedy DEIM algorithm. Specifically, the vector $\mathbf{c}_j$ is computed from $\mathbf{P}_j^T \boldsymbol{\Xi}_j \mathbf{c}_j = \mathbf{P}_j^T \boldsymbol{\xi}_{j+1}$, where $\boldsymbol{\xi}_j$ are the columns of the nonlinear POD modes matrix $\boldsymbol{\Xi}_p$. The actual interpolation point comes from looking for the maximum of the residual $\mathbf{R}_{j+1} = \boldsymbol{\xi}_{j+1} - \boldsymbol{\Xi}_j \mathbf{c}_j$. Each iteration of the algorithm produces another column of the sparse interpolation matrix $\mathbf{P}$. The integers **nmax** give the location of the interpolation points.

Code 13.5: [MATLAB] DEIM points 2 through r.

```
for j=2:r
    c=(P'*XI_m)\(P'*XI(:,j));
    res=XI(:,j)-XI_m*c;
    [Xi_max,nmax]=max(abs(res));
    XI_m=[XI_m,XI(:,j)];
    P=[P,z]; P(nmax,j)=1;
end
```

Code 13.5: [Python] DEIM points 2 through r.

```
for jj in range(1,r):
    c=np.linalg.solve(P.T@XI_m, P.T@XI[:,jj].reshape(n,1))
    res = XI[:,jj].reshape(n,1) - XI_m @ c
    nmax = np.argmax(np.abs(res))
    XI_m=np.concatenate((XI_m,XI[:,jj].reshape(n,1)),axis=1)
    P = np.concatenate((P,z),axis=1)
    P[nmax,jj] = 1
```

With the interpolation matrix, we are ready to construct the ROM. The first part is to construct the linear term $\boldsymbol{\Psi}^T \mathbf{L} \boldsymbol{\Psi}$ of (12.21) where the linear operator for NLS is the Laplacian. The derivatives are computed using the Fourier transform.

Code 13.6: [MATLAB] Projection of linear terms.

```
for j=1:r  % linear derivative terms
    Lxx(:,j)=ifft(-k.^2.*fft(Psi(:,j)));
end
L=(i/2)*(Psi')*Lxx;  % projected linear term
```

Code 13.6: [Python] Projection of linear terms.

```
Lxx = np.zeros((n,r),dtype='complex_')
for jj in range(r):
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

```
    Lxx[:,jj] = np.fft.ifft(-np.power(k,2)*np.fft.fft(Psi[:,
        jj]))
L = 0.5 * (1j) * Psi.T @ Lxx # projected linear term
```

The projection of the nonlinearity is accomplished using the interpolation matrix $\mathbf{P}$ with the formula (13.17). Recall that the nonlinear term in (12.21) is multiplied by $\mathbf{\Psi}^T$. Also computed is the interpolated version of the low-rank subspace spanned by $\mathbf{\Psi}$.

Code 13.7: [MATLAB] Projection of nonlinear terms.

```
P_NL=Psi'*( XI_m*inv(P'*XI_m) ); % nonlinear projection
P_Psi=P'*Psi; % interpolation of Psi
```

Code 13.7: [Python] Projection of nonlinear terms.

```
P_NL = Psi.T @ (XI_m @ np.linalg.inv(P.T @ XI_m))
P_Psi = P.T @ Psi # interpolation of Psi
```

It only remains now to advance the solution in time using a numerical time-stepper. This is done with a fourth-order Runge–Kutta routine.

Code 13.8: [MATLAB] Time-stepping of ROM.

```
[tt,a]=ode45('rom_deim_rhs',t,a,[],P_NL,P_Psi,L);
Xtilde=Psi*a';  % DEIM approximation
waterfall(x,t,abs(Xtilde')), shading interp, colormap gray
```

Code 13.8: [Python] Time-stepping of ROM.

```
a0_split = np.concatenate((np.real(a0),np.imag(a0))) #
    Separate real/complex pieces
a_split = integrate.odeint(rom_deim_rhs,a0_split,t,mxstep
    =10**6)
a = a_split[:,:r] + (1j)*a_split[:,r:]
Xtilde = Psi @ a.T # DEIM approximation
```

The right-hand side of the time-stepper is now completely low-dimensional.

Code 13.9: [MATLAB] Right-hand side of ROM.

```
function rhs=rom_deim_rhs(tspan, a,dummy,P_NL,P_Psi,L)
N=P_Psi*a;
rhs=L*a + i*P_NL*((abs(N).^2).*N);
```

Code 13.9: [Python] Right-hand side of ROM.

```
def rom_deim_rhs(a_split,tspan,P_NL=P_NL,P_Psi=P_Psi,L=L):
    a = a_split[:r] + (1j)*a_split[r:]
    N = P_Psi @ a
    rhs = L @ a + (1j) * P_NL @ (np.power(np.abs(N),2)*N)
```

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

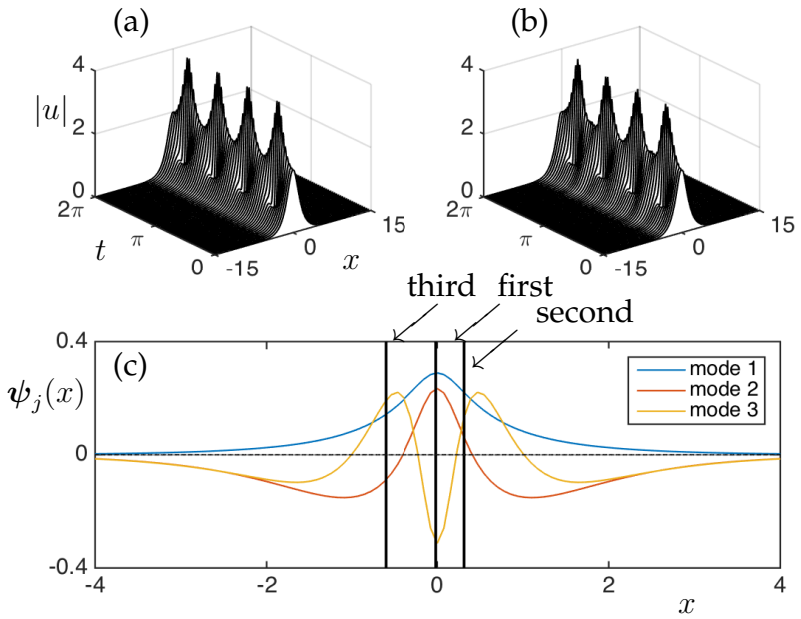


Figure 13.16: Comparison of the (a) full simulation dynamics and (b) rank $r = 3$ ROM using the three DEIM interpolation points. (c) A detail of the three POD modes used for simulation is shown along with the first, second, and third DEIM interpolation point locations. These three interpolation points are capable of accurately reproducing the evolution dynamics of the full PDE system.

```
    rhs_split = np.concatenate((np.real(rhs),np.imag(rhs)))
    return rhs_split
```

A comparison of the full simulation dynamics and rank $r = 3$ ROM using the three DEIM interpolation points is shown in Fig. 13.16. Additionally, the location of the DEIM points relative to the POD modes is shown. Aside from the first DEIM point, the other locations are not on the minima or maxima of the POD modes. Rather, the algorithm places them to maximize the residual.

QDEIM Algorithm

Although DEIM is an efficient greedy algorithm for selecting interpolation points, there are other techniques that are equally efficient. The recently proposed QDEIM algorithm [215] leverages the QR decomposition to provide efficient, greedy interpolation locations. This has been shown to be a robust mathematical architecture for sensor placement in many applications [481]. See Section 3.8 for a more general discussion. The QR decomposition can also provide a greedy strategy to identify interpolation points. In QDEIM, the QR pivot locations are the sensor locations. The following code can replace the DEIM algorithm to produce the interpolation matrix $\mathbf{P}$.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Code 13.10: [MATLAB] QR-based interpolation points.

```
[Q,R,pivot]=qr(NL.');
P=pivot(:,1:r);
```

Code 13.10: [Python] QR-based interpolation points.

```
Q,R,pivot = qr(NL.T,pivoting=True)
P_qr = np.zeros_like(x)
P_qr[pivot[:3]] = 1
```

Using this interpolation matrix gives identical interpolation locations as shown in Fig. 13.16. More generally, there are estimates that show that the QDEIM may improve error performance over standard DEIM [215]. The ease of use of the QR algorithm makes this an attractive method for sparse interpolation.

13.7 Decoder Networks for Interpolation

The gappy interpolation methods presented thus far are all based upon *linear* mappings between the measurement space and the full-state reconstruction. Equation (13.1) provides a mathematical representation of this mapping, which dictates how measurements in an r-dimensional (low-rank) space can be related to the original n-dimensional (high-dimensional) state space. The focus thus far is in leveraging SVD modes $\mathbf{\Psi}$ for reconstruction tasks. Specifically, if the original state space is represented in the low-dimensional subspace so that $\mathbf{u} = \mathbf{\Psi a}$, then the suite of gappy interpolation methods can be executed in order to approximate the high-fidelity solution.

To be more precise, recall that, in the gappy POD formulation, the measurement matrix specifies the interpolation to be used:

$$\tilde{\mathbf{u}} = \mathbf{P}\mathbf{u} \approx \mathbf{P}\mathbf{\Psi}\mathbf{a}, \tag{13.18}$$

where the state vector is expressed in terms of POD modes in the second approximation. Given measurements $\tilde{\mathbf{u}}$ along with a measurement matrix $\mathbf{P}$ and POD modes $\mathbf{\Psi}$, the coefficients for reconstruction can now be computed by least-squares:

$$\mathbf{a} = (\mathbf{P}\mathbf{\Psi})^{\dagger}\tilde{\mathbf{u}}, \tag{13.19}$$

where † is the Moore–Penrose pseudo-inverse. In terms of an optimization problem, this is alternatively formulated as

$$\mathbf{a} \in \underset{\tilde{\mathbf{a}}}{\operatorname{argmin}} \|\tilde{\mathbf{u}} - \mathbf{P}\mathbf{\Psi}\tilde{\mathbf{a}}\|_2^2. \tag{13.20}$$

This is the standard POD reconstruction error formulation. This is revisited here since the optimization formulation can be improved to stabilize POD reconstructions. Specifically, just like neural networks, additional regularizations

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

can be applied in order to ensure a more stable solution. The improved formulation adds an elastic net [777] regularization, which is a combination of ℓ_1 and ℓ_2 penalties:

$$\mathbf{a} \in \underset{\tilde{\mathbf{a}}}{\operatorname{argmin}} \|\tilde{\mathbf{u}} - \mathbf{P}\boldsymbol{\Psi}\tilde{\mathbf{a}}\|_2^2 + \lambda_1\|\tilde{\mathbf{a}}\|_1 + \lambda_2\|\tilde{\mathbf{a}}\|_2^2, \tag{13.21}$$

where $\lambda_{1,2}$ are hyperparameters that control the ℓ_1- and ℓ_2-norms, respectively. In what follows, this is referred to as POD PLUS, since it is an augmentation of standard POD. As will be shown, such a regularization improves the standard linear mapping from measurements to the state space.

Instead of linear mappings between measurements and state space, we leverage the universal approximation properties of neural networks to construct *nonlinear* mappings between measurements and state space. Specifically, we construct a decoder neural network so that [235]

$$\hat{\mathbf{u}} = \mathbf{f}_{\boldsymbol{\theta}}(\tilde{\mathbf{u}}), \tag{13.22}$$

where $\hat{\mathbf{u}}$ is an approximation to the full state $\mathbf{u}$ and $\mathbf{f}_{\boldsymbol{\theta}}(\cdot)$ is a decoder neural network. The optimization procedure evaluates the expression

$$\underset{\boldsymbol{\theta}}{\operatorname{argmin}} \sum_{j=1}^{N} \|\mathbf{u}_j - \mathbf{f}_{\boldsymbol{\theta}}(\tilde{\mathbf{u}}_j)\|_2^2 \tag{13.23}$$

with N training data pairs $\{\mathbf{u}_j, \tilde{\mathbf{u}}_j\}$ for $j = 1, 2, \ldots, N$. This supervised algorithm uses the N sample pairs from measurement $\tilde{\mathbf{u}}_j$ to its corresponding full state-space representation $\mathbf{u}_j$ in order to build a nonlinear mapping between them.

Figure 13.17 shows the architecture of the decoder mapping from measurements to the state space. The only thing that needs to be determined is the number of layers and their widths along with the activation functions. Erichson et al. [235] showed that a shallow decoder, in which only a few layers were used, provides an effective nonlinear mapping while using only modest amounts of training data. In addition, the optimization was modified to regularize the weights so that

$$\underset{\boldsymbol{\theta}}{\operatorname{argmin}} \sum_{j=1}^{N} \|\mathbf{u}_j - \mathbf{f}_{\boldsymbol{\theta}}(\tilde{\mathbf{u}}_j)\|_2^2 + \lambda\|\boldsymbol{\theta}\|_2^2, \tag{13.24}$$

where λ is a hyperparameter that determines the strength of the norm (ℓ_2) regularization. The Adam optimization algorithm [386] was used to train the shallow decoder. Various hyperparameters can be fine-tuned in practice, but the choice of parameters used worked well in practice for several physics-related examples.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

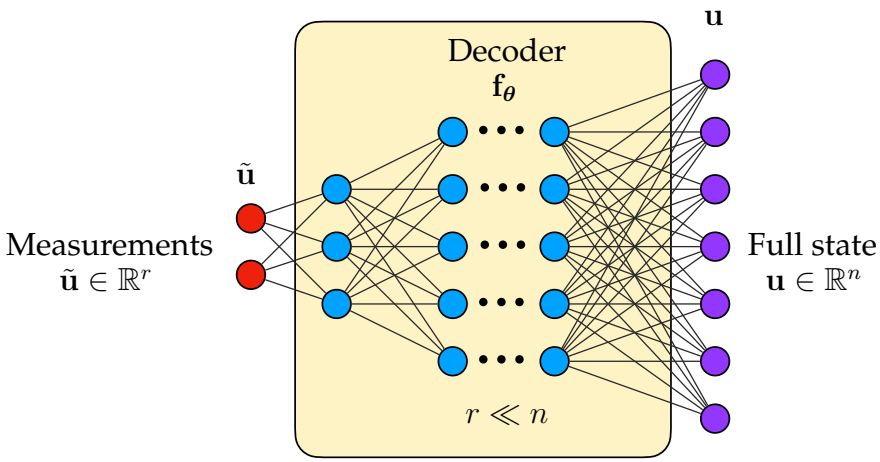


Figure 13.17: Decoder network providing a nonlinear map from measurements to the high-dimensional state space: $\hat{\mathbf{u}} = \mathbf{f}_{\boldsymbol{\theta}}(\tilde{\mathbf{u}})$. The decoder trains on data pairs $\{\mathbf{u}_j, \tilde{\mathbf{u}}_j\}$ for $j = 1, 2, \ldots, N$. The network is implemented in Python using PyTorch; research code for flow behind the cylinder is available via `https://github.com/erichson/ShallowDecoder`.

Modal Comparison: POD versus Shallow Decoder

ROMs exploit low-rank features of the data, which are often interpreted as *modes* [689] that characterize physical processes. POD modes provide optimal representations in an ℓ_2 sense. However, this does not guarantee that they are the best modes in a broader sense when considering noisy and dynamic data. Indeed, POD modes can be easily corrupted by outliers and noise so that they are compromised in producing accurate reconstructions of the underlying high-dimensional data from which they are extracted.

The shallow decoder network highlighted above also produces modal structures. In contrast with POD modes, which can be linearly superimposed to produce an approximation, the decoder network is a nonlinear transformation and linear superposition does not hold. Figure 13.18 shows the contrasting dominant modal structures that are generated from the flow around a cylinder example. Note that for POD modes, the dominant modes alternate between symmetric and antisymmetric modes in the vertical direction. Linearly superimposing these modes in time generates the canonical dynamics of von Kármán vortex shedding. In contrast, the shallow decoder modes are not symmetric. Rather, their shapes are very much like what is observed in the fluid flows, i.e., the modes look like snapshots of the fluid itself. The modes are not orthogonal, yet

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

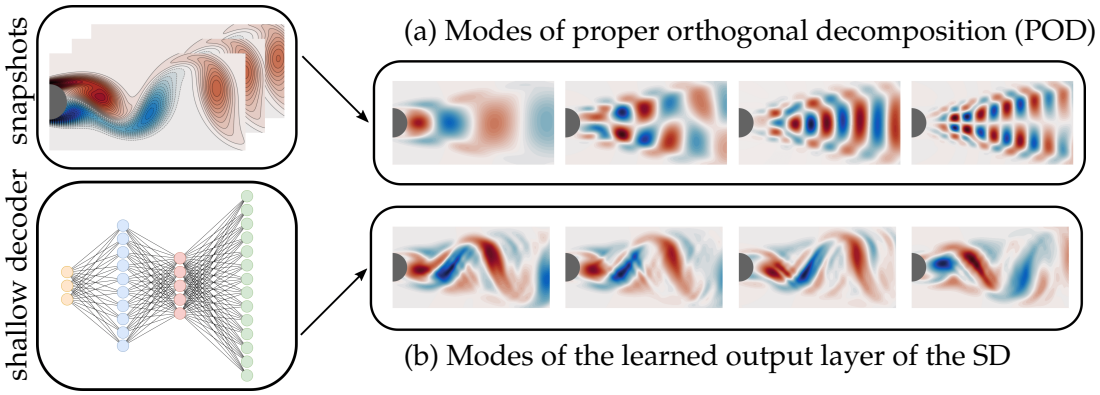


Figure 13.18: Dominant modes learned by the shallow decoder in comparison with dominant POD modes. The modal features show that the shallow decoder network constructs a reasonable characterization of the flow behind a cylinder using very different modal structures. Indeed, by not constraining the modes to be linear and orthogonal, as is enforced with POD, a potentially more interpretable feature space can be extracted from data. Such modes can be exploited for reconstruction of the state space from limited measurements and limited data. From Erichson et al. [235].

they are used in the nonlinear shallow encoder to reconstruct the fluid dynamics.

There is more than just a distinct difference in modal profiles between the POD and shallow decoder. The robustness of the linear versus nonlinear encoding strategies is remarkably different. To characterize the robustness and flexibility of the shallow decoder, we consider flow reconstruction in the presence of additive white noise. In practical experimental settings, noisy measurements are common and can have significant impact on building ROMs. Figure 13.19 shows the difference between the POD and POD PLUS methods in contrast to the shallow decoder. The shallow decoder shows a clear advantage and a de-noising effect. Indeed, the reconstructed snapshots allow for a meaningful interpretation of the underlying structure while also being highly robust. Interestingly, POD PLUS also significantly outperforms the standard gappy methods typically used in ROMs, while still maintaining linear superposition. Thus POD PLUS offers a hybrid method where performance is increased while retaining the advantageous features of linearity. But, overall, the shallow decoder shows that a neural network model $\mathbf{f}_{\boldsymbol{\theta}}(\cdot)$ can provide significant performance gains.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

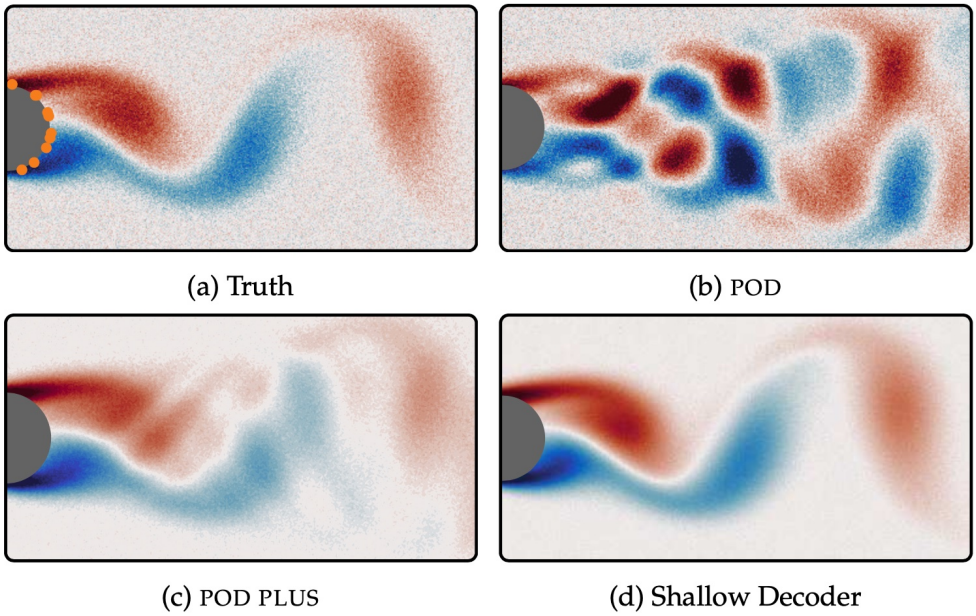


Figure 13.19: Reconstruction results for the flow around the cylinder with noise. For this simulation, the signal-to-noise ratio is 10. In (a) the target snapshot and the corresponding sensor configuration (using 10 sensors) is shown. Both POD and POD PLUS are not able to reconstruct the flow field, as shown in (b) and (c). The shallow decoder is able to reconstruct the coherent structure of the flow field, as shown in (d). From Erichson et al. [235].

13.8 Randomization and Compression for ROMs

This chapter has been largely concerned with the interpolation problem associated with ROMs. Specifically, how does one construct a ROM without recourse to the high-dimensional state. Gappy interpolation techniques aim to construct ROMs and compute nonlinear terms in PDEs in an efficient manner. Specifically, we recall that we are interested in building ROMs for (12.13). Assuming a solution ansatz $\mathbf{u} = \boldsymbol{\Psi}\mathbf{a}$ allows for the construction of POD and DMD ROMs for the evolution dynamics of $\mathbf{a}(t)$. Specifically, we have the following ROM models:

$$\frac{\mathrm{d}\mathbf{a}}{\mathrm{d}t} = \boldsymbol{\Psi}^T\mathbf{L}\boldsymbol{\Psi}\mathbf{a} + \boldsymbol{\Psi}^T\mathbf{N}(\boldsymbol{\Psi}\mathbf{a}, \boldsymbol{\beta}) \qquad \text{(POD)}, \tag{13.25a}$$

$$\frac{\mathrm{d}\mathbf{a}}{\mathrm{d}t} = \boldsymbol{\Psi}^T\mathbf{L}\boldsymbol{\Psi}\mathbf{a} + \boldsymbol{\Psi}^T\boldsymbol{\Phi}\exp(\boldsymbol{\Omega}t)\mathbf{b} \quad \text{(POD–DMD)}. \tag{13.25b}$$

The computational bottleneck addressed in this chapter is the repeated evaluation of the nonlinear term $\boldsymbol{\Psi}^T\mathbf{N}(\boldsymbol{\Psi}\mathbf{a}, \boldsymbol{\beta})$, which is done using a gappy (e.g., the DEIM or Q-DEIM) methods. The greedy algorithms outlined in the preceding

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

sections highlight how a small number of measurements can be used with the low-rank modes $\mathbf{\Psi}$ to accomplish this task of evaluating inner products repeatedly.

Ignored overall in the ROM formulation is the *offline* cost of producing low-rank embeddings, manifest here by $\mathbf{\Psi}$ and $\mathbf{\Phi}$. The high-fidelity simulation data $\mathbf{X} \in \mathbb{C}^{n\times m}$ used to produce ROMs is often exceptionally high-dimensional. Thus the cost of producing a full or economy SVD is large when $n, m \gg 1$. This often is not a problem if only a single SVD needs to be performed, but often the ROM needs to be updated, and recourse to the original high-dimensional system is required in order to update $\mathbf{\Psi}$ and $\mathbf{\Phi}$. To avoid costly re-computations, randomized linear algebra [309, 471] and compressive sampling [153, 156, 278] techniques can be used for enhanced computational efficiency.

The main idea is to consider basis functions $\mathbf{\Psi}$ not from the full set of measurements but from a few spatially incoherent measurements. The measurement matrix $\mathbf{C} \in \mathbb{R}^{p\times m}$, which was originally used as a matrix for defining gappy interpolation points, is now used to characterize the random measurements of the system and produce the compressed matrix $\mathbf{Z} \in \mathbb{R}^{p\times n}$ such that

$$\mathbf{Z} = \mathbf{C}\mathbf{X}.$$

Here, we consider sparse measurements of the snapshots matrix in order to compute POD and DMD from this new compressed snapshot matrix. To start, it is assumed that the snapshot matrix $\mathbf{X}$ is almost square, e.g., $n \approx m$, and one can imagine this is a realistic situation working with an explicit time scheme or in a many-query context. Section 1.8 shows that the smaller matrix is now decomposed using QR so that $\mathbf{Z} = \mathbf{Q}\mathbf{R}$. The original data is then projected onto the QR basis $\mathbf{Y} = \mathbf{Q}^*\mathbf{X}$ and the SVD of the much smaller matrix is computed, $\mathbf{Y} = \mathbf{U}_\mathbf{Y}\mathbf{\Sigma}\mathbf{V}^*$. This allows one to transform the low-rank matrix $\mathbf{U}_\mathbf{Y}$ back to the original coordinates and approximate the POD basis $\mathbf{\Psi} = \mathbf{Q}\mathbf{U}_\mathbf{Y}$. This provides a computationally efficient method for extracting the POD modes. ROMs constructed from randomized POD methods have now been investigated by several groups [11, 43, 141], with all them demonstrating the computational performance advantages gained by randomization.

The randomized architecture can also be used to produce DMD approximations represented by $\mathbf{\Phi}$ [135]. Thus, instead of performing the DMD algorithm on snapshot pairs associated with $\mathbf{X}$, DMD is instead performed on the compressively sampled matrices

$$\mathbf{Z}' = \mathbf{A}_\mathbf{Z}\mathbf{Z}, \tag{13.26}$$

where $\mathbf{Z} = [\mathbf{z}_1\ \mathbf{z}_2\ \cdots\ \mathbf{z}_m]$ and $\mathbf{Z}' = [\mathbf{z}'_1\ \mathbf{z}'_2\ \cdots\ \mathbf{z}'_m]$ as in Section 7.2. The DMD algorithm computes the DMD eigenvalues and DMD modes $(\mathbf{\Lambda}_\mathbf{Z}, \mathbf{\Phi}_\mathbf{Z})$ along with the eigenvectors matrix $\mathbf{W}_\mathbf{Z}$ of the similarity matrix $\tilde{\mathbf{A}}_\mathbf{Z}$. The DMD eigenvalues are self-similar so that $\mathbf{\Lambda} \approx \mathbf{\Lambda}_\mathbf{Z}$ and the DMD modes are given by

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

$\mathbf{\Phi} = \mathbf{X}'\mathbf{V}_\mathbf{Z}\mathbf{\Sigma}_\mathbf{Z}^{-1}\mathbf{W}_\mathbf{Z}$. Thus the computation avoids an expensive SVD by performing the SVD in the low-rank subspace of $\mathbf{Z}$. Like POD-based randomization, DMD with randomization provides a scalable architecture for building ROMs, as demonstrated by a number of groups [28, 93, 135, 234]

Overall, randomized techniques are a promising approach to circumvent expensive offline computations in model order reduction. In particular, when dealing with large snapshot matrices, there is now abundant evidence to suggest the use of randomized SVD methods for POD- and DMD-based decompositions. They both provide very accurate solutions and promise significant computational savings in the offline stage, which turns out to be the most expensive part of constructing the surrogate model. Indeed, the rapid computation of $\mathbf{\Psi}$ and $\mathbf{\Phi}$ through such techniques can greatly aid in solving the surrogate models (13.25)

13.9 Machine Learning ROMs

Inspired by machine learning methods, the various POD bases for a parameterized system are merged into a master library of POD modes $\mathbf{\Psi}_L$ which contains all the low-rank subspaces exhibited by the dynamical system. This leverages the fact that POD provides a principled way to construct an r-dimensional subspace $\mathbf{\Psi}_r$ characterizing the dynamics while sparse sampling augments the POD method by providing a method to evaluate the problematic nonlinear terms using a p-dimensional subspace projection matrix $\mathbf{P}$. Thus a small number of points can be sampled to approximate the nonlinear terms in the ROM. Figure 13.20 illustrates the library building procedure whereby a dynamical regime is sampled in order to construct an appropriate POD basis $\mathbf{\Psi}$.

The method introduced here capitalizes on these methods by building low-dimensional libraries associated with the full nonlinear system dynamics as well as the specific nonlinearities. Interpolation points, as will be shown in what follows, can be used with sparse representation and compressive sensing to (i) identify dynamical regimes, (ii) reconstruct the full state of the system, and (iii) provide an efficient nonlinear model reduction and Galerkin–POD prediction for the future state.

The concept of library building of low-rank *features* from data is well established in the computer science community. In the reduced-order modeling community, it has recently become an enabling computational strategy for parametric systems. Indeed, a variety of recent works have produced libraries of ROM models [15, 112, 136, 179, 553, 554, 555, 619] that can be selected and/or interpolated through measurement and classification. Alternatively, cluster-based reduced-order models use a k-means clustering to build a Markov transition model between dynamical states [367]. These recent innovations are similar to

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

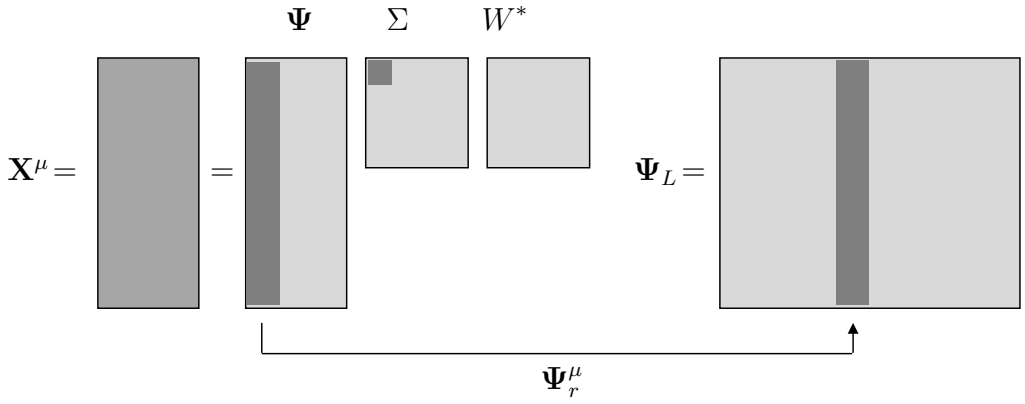


Figure 13.20: Library construction from numerical simulations of the governing equations (12.1). Simulations are performed of the parameterized system for different values of a bifurcation parameter μ. For each regime, low-dimensional POD modes $\mathbf{\Psi}_r$ are computed via an SVD decomposition. The various rank-r truncated subspaces are stored in the library of modes matrix $\mathbf{\Psi}_L$. This is the learning stage of the algorithm. Reproduced from Kutz et al. [424].

the ideas advocated here. However, our focus is on determining how a suitably chosen $\mathbf{P}$ can be used across all the libraries for POD mode selection and reconstruction. One can also build two sets of libraries: one for the full dynamics and a second for the nonlinearity so as to make it computationally efficient with the DEIM strategy [619]. Before these more formal techniques based on machine learning were developed, it was already realized that parameter domains could be decomposed into subdomains and a local ROM/POD computed in each subdomain. Patera and co-workers [229] used a partitioning based on a binary tree, whereas Amsallem et al. [16] used a Voronoi tessellation of the domain. Such methods were closely related to the work of Du and Gunzburger [216] where the data snapshots were partitioned into subsets and multiple reduced bases computed. The multiple bases were then recombined into a single basis, so it does not lead to a library, *per se*. For a review of these domain partitioning strategies, please see [17].

POD Mode Selection

Although there are a number of techniques for selecting the correct POD library elements to use, including the workhorse k-means clustering algorithm [15, 179, 553, 554, 555], one can also instead make use of sparse sampling and the sparse representation for classification (SRC) innovations outlined in Chapter 3 to characterize the nonlinear dynamical system [112, 136, 619]. Specifically, the goal is to use a limited number of sensors (interpolation points) to classify the dynamical regime of the system from a range of potential POD library ele-

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

ments characterized by a parameter β. Once a correct classification is achieved, a standard ℓ_2 reconstruction of the full state space can be accomplished with the selected subset of POD modes, and a Galerkin–POD prediction can be computed for its future.

In general, we will have a sparse measurement vector $\tilde{\mathbf{u}}$ given by (13.1). The full-state vector $\mathbf{u}$ can be approximated with the POD library modes ($\mathbf{u} = \mathbf{\Psi}_L \mathbf{a}$), therefore

$$\tilde{\mathbf{u}} = \mathbf{P}\mathbf{\Psi}_L \mathbf{a}, \tag{13.27}$$

where $\mathbf{\Psi}_L$ is the low-rank matrix whose columns are POD basis vectors concatenated across all β regimes and $\mathbf{c}$ is the coefficient vector giving the projection of $\mathbf{u}$ onto these POD modes. If $\mathbf{P}\mathbf{\Psi}_L$ obeys the restricted isometry property and $\mathbf{u}$ is sufficiently sparse in $\mathbf{\Psi}_L$, then it is possible to solve the highly underdetermined system (13.27) with the sparsest vector $\mathbf{a}$. Mathematically, this is equivalent to an ℓ_0 optimization problem, which is NP-hard. However, under certain conditions, a sparse solution of (13.27) can be found (see Chapter 3) by minimizing the l_1-norm instead, so that

$$\mathbf{c} = \underset{\mathbf{a}'}{\operatorname{argmin}} ||\mathbf{a}'||_1 \quad \text{subject to} \quad \tilde{\mathbf{u}} = \mathbf{P}\mathbf{\Psi}_L \mathbf{a}. \tag{13.28}$$

The last equation can be solved through standard convex optimization methods. Thus the ℓ_1-norm is a proxy for sparsity. Note that we use the sparsity only for classification, not for reconstruction. Figure 13.21 demonstrates the sparse sampling strategy and prototypical results for the sparse solution $\mathbf{a}$.

Example: Flow Around a Cylinder

To demonstrate the sparse classification and reconstruction algorithm developed, we consider the canonical problem of flow around a cylinder. This problem is well understood and has already been the subject of studies concerning sparse spatial measurements [112, 122, 136, 376, 491, 619, 732]. Specifically, it is known that, for low to moderate Reynolds numbers, the dynamics are spatially low-dimensional and POD approaches have been successful in quantifying the dynamics. The Reynolds number, Re, plays the role of the bifurcation parameter β in (12.1), i.e., it is a parameterized dynamical system.

The data we consider comes from numerical simulations of the incompressible Navier–Stokes equation:

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} + \nabla p - \frac{1}{Re} \nabla^2 \mathbf{u} = 0, \tag{13.29}$$

$$\nabla \cdot \mathbf{u} = 0, \tag{13.30}$$

where $\mathbf{u}(x, y, t) \in \mathbb{R}^2$ represents the 2D velocity, and $p(x, y, t)$ the corresponding pressure field. The boundary conditions are as follows: (i) constant flow of $\mathbf{u} =$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

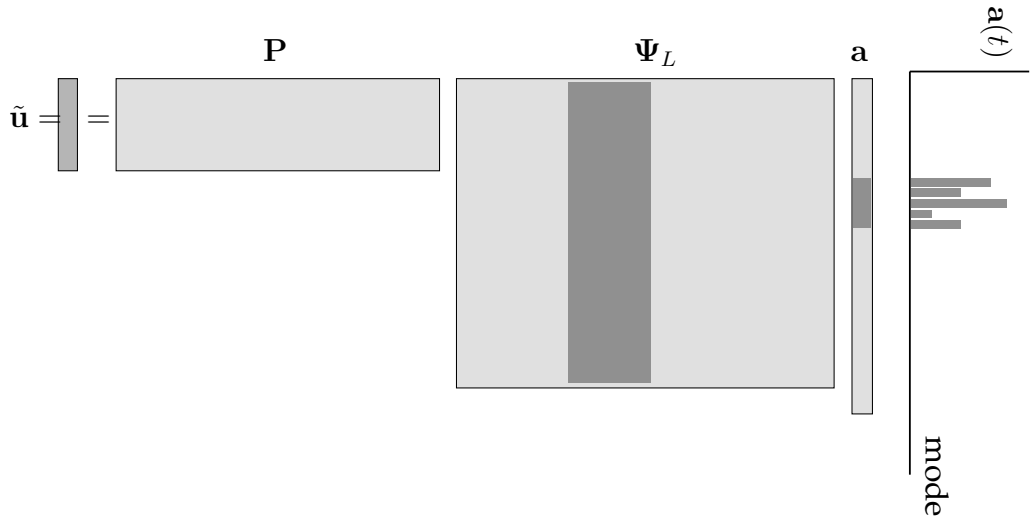


Figure 13.21: The sparse representation for classification (SRC) algorithm for library mode selection; see Section 3.6 for more details. In this mathematical framework, a sparse measurement is taken of the system (12.1) and a highly under-determined system of equations $\mathbf{P}\boldsymbol{\Psi}_L\mathbf{a} = \tilde{\mathbf{u}}$ is solved subject to ℓ_1 penalization so that $\|\mathbf{a}\|_1$ is minimized. Illustrated is the selection of the μth POD modes. The bar plot on the right depicts the non-zero values of the vector $\mathbf{a}$, which correspond to the $\boldsymbol{\Psi}_r$ library elements. Note that the sampling matrix $\mathbf{P}$ that produces the sparse sample $\tilde{\mathbf{u}} = \mathbf{P}\mathbf{u}$ is critical for success in classification of the correct library elements $\boldsymbol{\Psi}_r$ and the corresponding reconstruction. Reproduced from Kutz et al. [424].

$(1, 0)^T$ at $x = -15$, i.e., the entry of the domain; (ii) constant pressure of $p = 0$ at $x = 25$, i.e., the end of the domain; and (iii) Neumann boundary conditions, i.e., $\partial\mathbf{u}/\partial\mathbf{n} = 0$ on the boundary of the domain and the cylinder (centered at $(x, y) = (0, 0)$ and of radius unity).

For each relevant value of the parameter Re, we perform an SVD on the data matrix in order to extract POD modes. It is well known that, for relatively low Reynolds number, a fast decay of the singular values is observed so that only a few POD modes are needed to characterize the dynamics. Figure 13.22 shows the three most dominant POD modes for Reynolds numbers $Re = 40$, 150, 300, and 1000. Note that 99% of the total energy (variance) is selected for the POD mode selection cut-off, giving a total of one, three, three, and nine POD modes to represent the dynamics in the regimes shown. For a threshold of 99.9%, more modes are required to account for the variability.

Classification of the Reynolds number is accomplished by solving the optimization problem (13.28) and obtaining the sparse coefficient vector $\mathbf{a}$. Note that each entry in $\mathbf{a}$ corresponds to the energy of a single POD mode from our library. For simplicity, we select a number of local minima and maxima

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

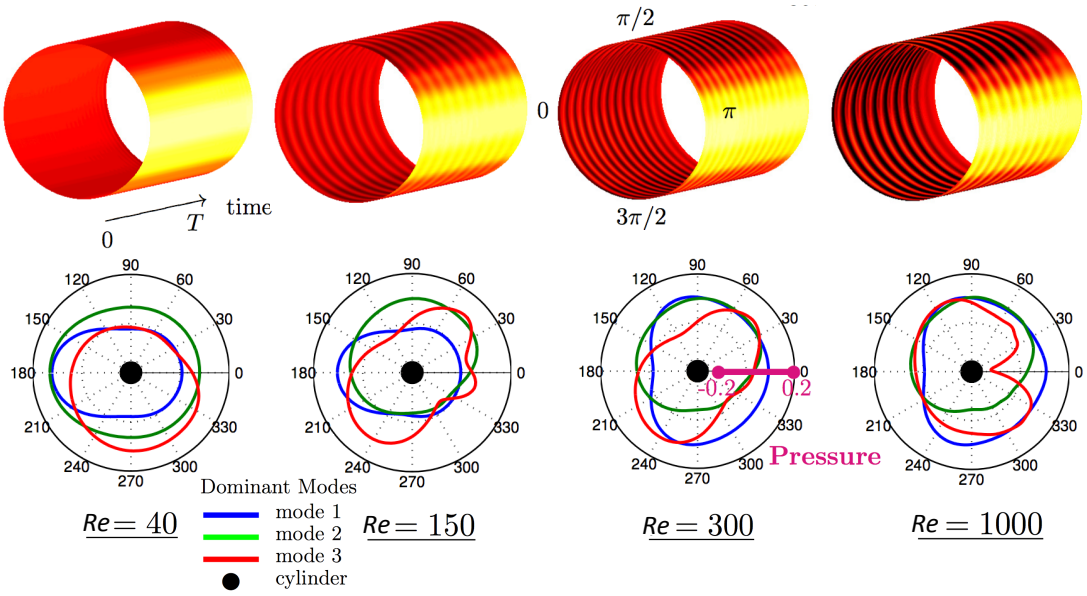


Figure 13.22: Time dynamics of the pressure field (top panels) for flow around a cylinder for Reynolds numbers $Re = 40$, 150, 300, and 1000. Collecting snapshots of the dynamics reveals that low-dimensional structures dominate the dynamics. The dominant three POD pressure modes for each Reynolds number regime are shown (bottom panels) in polar coordinates. The pressure scale is in magenta (bottom right). Reproduced from Kutz et al. [424].

of the POD modes as sampling locations for the matrix $\mathbf{P}$. The classification of the Reynolds number is done by summing the absolute value of the coefficient that corresponds to each Reynolds number. To account for the large number of coefficients allocated for the higher Reynolds number (which may be 16 POD modes for 99.9% variance at $Re = 1000$, rather than a single coefficient for Reynolds number 40), we divide by the square root of the number of POD modes allocated in a for each Reynolds number. The classified regime is the one that has the largest magnitude after this process.

Although the classification accuracy is high, many of the false classifications are due to categorizing a Reynolds number from a neighboring flow, i.e., Reynolds number 1000 is often mistaken for Reynolds number 800. This is due to the fact that these two Reynolds numbers are strikingly similar and the algorithm has a difficult time separating their modal structures. Figure 13.23 shows a schematic of the sparse sensing configuration along with the reconstruction of the pressure field achieved at $Re = 1000$ with 15 sensors. Classification and reconstruction performance can be improved using other methods for constructing the sensing matrix $\mathbf{P}$ [112, 122, 136, 376, 491, 619, 732]. Regardless, this example demonstrates the usage of sparsity-promoting techniques for POD mode

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

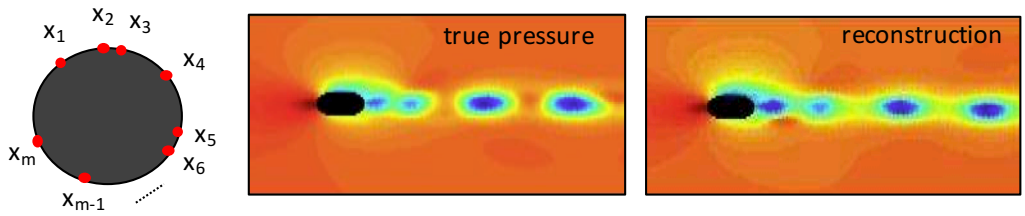


Figure 13.23: Illustration of m sparse sensor locations (left panel) for classification and reconstruction of the flow field. The selection of sensory/interpolation locations can be accomplished by various algorithms [112, 122, 136, 376, 491, 619, 732]. For a selected algorithm, the sensing matrix $\mathbf{P}$ determines the classification and reconstruction performance. Reproduced from Kutz et al. [424].

selection (ℓ_1 optimization) and subsequent reconstruction (ℓ_2 projection).

Finally, to visualize the entire sparse sensing and reconstruction process more carefully, Fig. 13.24 shows both the Reynolds number reconstruction for the time-varying flow field along with the pressure field and flow field reconstructions at select locations in time. Note that the SRC scheme along with the supervised ML library provide an effective method for characterizing the flow strictly through sparse measurements. For higher Reynolds numbers, it becomes much more difficult to accurately classify the flow field with such a small number of sensors. However, this does not necessarily jeopardize the ability to reconstruct the pressure field, as many of the library elements at higher Reynolds numbers are fairly similar.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

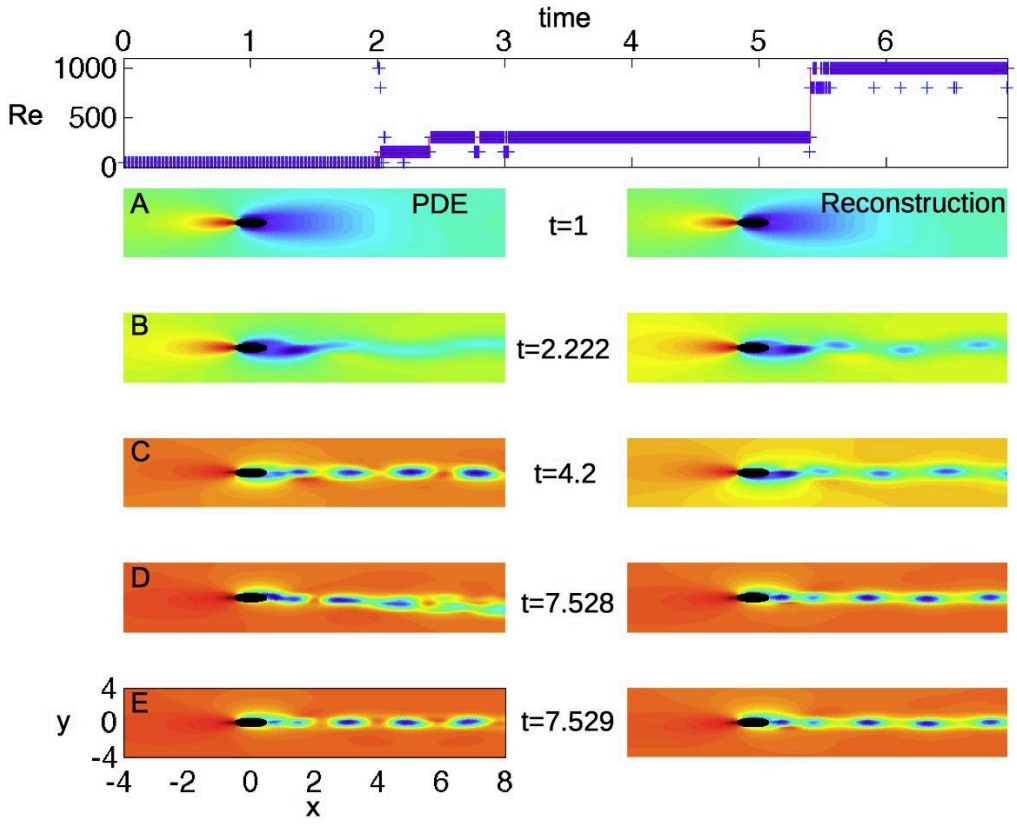


Figure 13.24: Sparse-sensing Reynolds-number identification and pressure-field reconstruction for a time-varying flow. The top panel shows the actual Reynolds number used in the full simulation (thick solid lines) along with its compressive sensing identification (crosses). Panels A–E show the reconstruction of the pressure field at five different locations in time (top panel) demonstrating an accurate (qualitatively) reconstruction of the pressure field. (The left side the simulated pressure field is presented, while the right side contains the reconstruction.) Note that, for higher Reynolds numbers, the classification becomes more difficult. Reproduced from Bright et al. [112].

Suggested Reading

Texts

(1) Certified reduced basis methods for parametrized partial differential equations, by J. Hesthaven, G. Rozza, and B. Stamm, 2015 [325].

(2) Reduced basis methods for partial differential equations: An introduction, by A. Quarteroni, A. Manzoni, and N. Federico, 2015 [578].

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

(3) Model reduction and approximation: Theory and algorithms, by P. Benner, A. Cohen, M. Ohlberger, and K. Willcox, 2017 [74].

Papers and reviews

(1) A survey of model reduction methods for parametric systems, by P. Benner, S. Gugercin, and K. Willcox, *SIAM Review*, 2015 [75].

(2) Model reduction using proper orthogonal decomposition, by S. Volkwein, *Lecture Notes, Institute of Mathematics and Scientific Computing, University of Graz*, 2011 [737].

(3) Nonlinear model reduction for dynamical systems using sparse sensor locations from learned libraries, by S. Sargsyan, S. L. Brunton, and J. N. Kutz, *Physical Review E*, 2015 [619].

(4) An online method for interpolating linear parametric reduced-order models, by D. Amsallem and C. Farhat, *SIAM Journal of Scientific Computing*, 2011 [15].

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Homework

Exercise 13-1. Consider the three functions:

$$f(x) = x\exp(-x^2), \tag{13.31}$$
$$f(x) = \exp[-(x-0.5)^2] + 3\exp[-2(x+3/2)^2], \tag{13.32}$$
$$f(x) = \sin(\pi x/8)\cos(\pi x/4), \tag{13.33}$$

on the interval $x \in [-4, 4]$. Approximate each function with $n \gg 1$ points and use r random point measurements to reconstruct the function using (i) the first r Gauss–Hermite functions, and (ii) the first $r/2$ cosine and sine modes $\cos(n\pi x/L)$ and $\sin(n\pi x/L)$, where $n = 0, 2, \ldots, r/2$. For the reconstruction, produce the least-squares error as a function of the rank r. Ensemble the results by considering a large number of random point measurements to produce a mean and variance of the error statistics.

Repeat the experiment above but use the QR algorithm and the DEIM algorithm to compute the r random point measurement locations. Compare the error to the statistical distribution of errors for random measurement locations.

Exercise 13-2. Train a decoder network that maps high-fidelity, well-resolved solutions for the Kuramoto–Sivashinsky (KS) equation in a parameter regime where spatio-temporal chaos is exhibited to randomly chosen point measurements of the system. With test data, evaluate the performance of the decoder as a function of the r point measurements. Also evaluate statistically the stability of the decoder for reconstruction as a function of the random point measurement locations.

Exercise 13-3. Consider the nonlinear Schrödinger equation solver with DEIM and QDEIM integration. Repeat the experiment of constructing a ROM using r random measurements. Determine the value of r for which the ROM model gives similar performance to DEIM and QDEIM with high probability.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Chapter 14

Physics-Informed Machine Learning

In this chapter, many of the critically enabling aspects of machine learning are brought together for modeling problems in science and engineering. Specifically, the goal of this chapter is to highlight a number of methods that have been recently developed under the aegis of physics-informed machine learning. The process of machine learning may be broken down into a number of key stages, each of which provides an opportunity to embed or enforce prior physical knowledge: (1) formulating a problem to model, (2) collecting and curating the data used to train the model, (3) choosing an architecture to best represent or model the data, (4) designing a loss function to assess the performance of the model and guide the learning process, and (5) selecting and implementing an optimization algorithm to train the model to minimize the loss function over the training data. In the following, several of these physics-informed machine learning strategies will be investigated.

These techniques often involve the training of neural networks in the overall workflow. The neural networks will be denoted by

$$\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}), \tag{14.1}$$

where $\boldsymbol{\theta}$ are the neural network weights and $\mathbf{f}(\cdot)$ characterizes the network architecture (number of layers, structure, regularizers). The weights $\boldsymbol{\theta}$ are then optimized to minimize a loss function over training data $\mathbf{X}$:

$$\boldsymbol{\theta}^* = \text{argmin}_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}, \mathbf{X}). \tag{14.2}$$

The various physics-informed networks highlight interesting structures and constraints imposed on the model $\mathbf{f}_{\boldsymbol{\theta}}$ and loss function $\mathcal{L}$. Often multiple neural networks are trained simultaneously in order to exploit a structural constraint of a spatio-temporal system. Moreover, targeting the use of deep learning is important to retain interpretability and explainability of models in the discovery process. This is not an exhaustive survey, but rather a targeted exploration of some methods that have had broad appeal in the community due to their

effectiveness, ease of use and/or interpretability. More details can be found in recent reviews [224, 111, 131, 375].

Importantly, *parsimony* has long been a guiding principle in physical modeling, favoring the simplest model that describes the data to avoid overfitting and promote generalization. This principle of parsimony has been central in physics for centuries, from Aristotle to Einstein. In modern machine learning, parsimony is still a guiding principle for interpretable and generalizable models, and it may be enforced through (i) a low-dimensional coordinate system, (ii) a sparse representation of governing equations, or (iii) in capturing parametric dependencies.

14.1 Mathematical Foundations

Data-driven models are an emerging and important paradigm for science and engineering. They also provide the foundational mathematical framing required for virtual instantiations of physical systems, i.e., the *digital twin*. Specifically, digital twins integrate with Kalman filtering architectures, which together produce predictions that are a combination of models and data. These data-driven discovery tools are achieved using simple regression techniques outlined in previous chapters that can often lead to improved interpretable and generalizable models. Although DNN architectures are used to learn physics, there remain critical issues concerning generalizability, interpretability, overfitting, and significant data requirements, limiting their usefulness and computational tractability for data-driven models. Regardless of the method used, they are compromised in practice by limited data, corruption due to noise, unmeasured latent variables, parametric dependences, and unaccounted-for physics. The targeted use of data-driven techniques provides a structure for model reduction, much like autoencoder structures, which facilitates a rapid and adaptive ROM construction paradigm with a diversity of potential methods for learning time evolution (DMD, Koopman, SINDy, etc.).

An overarching goal in data-driven modeling is to leverage machine learning algorithms to learn physically interpretable and generalizable models of spatio-temporal systems from offline and/or online streaming data. There are three critical scenarios to consider, corresponding to when the baseline physical model, or parametric form, is known, unknown, or only partially known. Thus we seek to perform system identification from data $\mathbf{y} \in \mathbb{R}^p$ to learn a model in a high-dimensional state space $\mathbf{x} \in \mathcal{X} \subseteq \mathbb{R}^n$, where $n \gg 1$; often $p \ll n$. Specifically,

$$\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} = \mathbf{f}(\mathbf{x}, t, \boldsymbol{\theta}, \mathbf{w}_d), \tag{14.3a}$$

$$\mathbf{y} = \mathbf{h}(t, \mathbf{x}(t)) + \mathbf{w}_n, \tag{14.3b}$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

where the dynamics are prescribed by $\mathbf{f} : \mathcal{X} \to \mathbb{R}^n$, and the observation operator is $\mathbf{h} : \mathcal{X} \to \mathbb{R}^p$. The measurements typically occur at discrete times t_k, in which case they are denoted by $\mathbf{y}_k$. Observations are compromised by measurement noise $\mathbf{w}_n$, which is typically described by a probability distribution (e.g., a normal distribution $\mathbf{w}_n \sim \mathcal{N}(\mu, \sigma)$). The dynamics are prescribed by a set of parameters $\boldsymbol{\theta}$. Moreover, the dynamics may be subject to stochastic effects characterized by $\mathbf{w}_d$.

The goal is as follows: Given m measurements $\mathbf{y}_k$ arranged in the matrix $\mathbf{Y} = [\mathbf{y}_1\ \mathbf{y}_2\ \cdots\ \mathbf{y}_m] \in \mathbb{R}^{p\times m}$, infer the dynamics $\mathbf{f}(\cdot)$ (or *unknown* portion of dynamics) with parameterization $\boldsymbol{\theta}$, the measurement operator $\mathbf{h}(\cdot)$, or a proxy model of the true system, so that tasks such as control and forecasting can be accomplished. Adding to the difficulty of the task are multi-scale and multi-physics phenomena. Even the simplest multi-scale system can challenge many data-driven methodologies. To be more specific, a simple two-scale system, for example, represents difficulties in trying to extract the governing equations which are modified to:

$$\frac{\mathrm{d}\mathbf{x}_1}{\mathrm{d}t} = \mathbf{f}_1(\mathbf{x}_1, \mathbf{x}_2, t, \tau, \boldsymbol{\theta}_1, \mathbf{w}_{d,1}) \quad \text{and} \quad \frac{\mathrm{d}\mathbf{x}_2}{\mathrm{d}\tau} = \mathbf{f}_2(\mathbf{x}_1, \mathbf{x}_2, t, \tau, \boldsymbol{\theta}_2, \mathbf{w}_{d,2}), \tag{14.4}$$

where $\tau = \epsilon t$ (with $\epsilon \ll 1$) is a slow scale.

If $\mathbf{h}(\cdot)$ is not the identity and/or $\mathbf{w}_n$ is not zero, then we have *imperfect data.* In general, inferring $\mathbf{f}(\mathbf{x})$ is an ill-posed problem whose solution must be accomplished through judiciously chosen regularization. In the case of (14.4), this is accomplished through first decomposing the data into its constitutive fast and slow timescales.

Solving the ill-posed problem (14.3) is a fundamental scientific and mathematical challenge. To date, it has only been accomplished in highly specialized settings, typically with full-state measurements and high-quality (low-noise) data. Significant mathematical innovations are still required in order to make this a general and robust architecture. The multi-scale equation (14.4) remains exceptionally challenging since it requires the integration of a broad set of mathematical tools.

Sensors and Limited Data

Everything starts with data acquisition. This is often overlooked in machine learning methods, which assume that access to the correct variables is available. Thus the mapping $\mathbf{h}(\cdot)$ and its inverse are important to learn. For many complex systems, the latent variable space is an important aspect of the discovery process. For instance, time-delay embeddings [126], and recourse to Taken's theorem, help establish a critical connection to dynamical systems theory and a potential reconstruction of the full state space with limited measurements. There are four critical aspects to developing a robust sensing frame-

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

work: (i) sensor placement, (ii) sensor cost, (iii) discovery of the measurement map $\mathbf{h}(\cdot)$, and (iv) multi-modal data integration from diverse sensor types (video, audio, thermal, etc.).

Recent efforts have established some of the earliest rigorous mathematical results on formulating optimal sensor placement and minimal cost strategies for complex spatio-temporal systems [481]. New fundamental mathematical innovations are required to identify near optimal sensor locations for systems with nonlinear manifold embeddings, which are typical of real data. DNNs can be used for decoder networks capable of producing a highly improved mapping between the data and the underlying state space [235]. It would also be advantageous to use the time-delay embedding structure to try and reconstruct, as best as possible, the latent variables and to reframe the greedy algorithms based upon the time-delay data. To date, it is unknown what the limits and mathematical possibilities are for using such a method to extract the full-state variable $\mathbf{x}$ from measurements $\mathbf{y}$. In addition to extracting critical information on the state space, DNNs have been recently shown to be capable of de-noising data sets in a manner that is comparable to, and in many cases better than, Kalman filtering methods. Potentially helping improve these results are multi-modal data fusion techniques which can be potentially used to help improve decision making or predictions. Sensors are critical for determining $\mathbf{h}(\cdot)$, and targeted use of DNNs [235] suggests robust architectures for making the best use of data collected for model discovery.

Coordinate Discovery and Data Representation

Data processed from the multi-modal sensors can discover a transformation $\mathbf{z} = \mathbf{g}(\mathbf{x})$ for parsimonious, low-dimensional dynamics [168, 465]

$$\dot{\mathbf{z}} = \mathbf{F}(\mathbf{z}, t, \boldsymbol{\theta}, \mathbf{w}_d), \tag{14.5}$$

where $\mathbf{z} \in \mathcal{Z} \subseteq \mathbb{R}^r$ is an r-dimensional ($r \ll n$) model of the physics specified by $\mathbf{F}(\cdot)$. Ultimately, the discovery of the nonlinear transform $\mathbf{g}(\cdot)$, through training neural network autoencoders, gives the coordinates for parsimonious dynamics $\mathbf{F}(\cdot)$.

If only limited data is available, then it may be required to produce a low-fidelity, online model using a linear map. This can be done with an r-rank SVD/POD mode truncation of the snapshots of $\mathbf{x}$. Dynamic mode decomposition, or a Koopman approximation using augmented state-space measurements, can then be used on this low-rank subspace to produce the best-fit linear model through the data. This provides a baseline architecture for diagnostics and forecasting. As more data is required, a full nonlinear mapping and nonlinear model can be used to refine the results, both in terms of building a lower-rank nonlinear subspace and for producing a parsimonious nonlinear

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

dynamics. As sufficient data is acquired from the sensors, the data-discovery pipeline then produces the flow

$$\mathbf{y} \in \mathbb{R}^p \text{ (measurements)} \longrightarrow \mathbf{x} \in \mathbb{R}^n \text{ (state space)} \longrightarrow \mathbf{z} \in \mathbb{R}^r \text{ (ROM)} \quad (14.6)$$

with two mappings to discover, $\mathbf{h}$ and $\mathbf{g}$. With limited data, SVD provides a linear approximation. Refinement of the linear approximation can be learned over time using architectures where the identity (linear) mapping is the leading-order approximation [279, 544].

Overall, solving the ill-posed problem (14.3) is the central aim of physics-informed machine learning. In what follows, a diversity of techniques are presented which leverage the data to build advantageous models (neural networks, for instance) that can be used for forecasting and characterization.

14.2 SINDy Autoencoder: Coordinates and Dynamics

In this first vignette on physics-informed machine learning, we explore an architecture capable of jointly and simultaneously learning coordinates and parsimonious dynamics. Specifically, Champion et al. [168] present a method for the simultaneous discovery of sparse dynamical models (SINDy) and coordinates (autoencoders) that enable these simple representations. The aim in the architecture is to leverage the parsimony and interpretability of SINDy with the universal approximation capabilities of deep neural networks to discover an appropriate coordinate system in which to embed the dynamics. This can produce interpretable and generalizable models capable of extrapolation and forecasting, since the dynamical model is minimally parameterized. The architecture is shown in Fig. 14.1, where an autoencoder is used to embed the original data $\mathbf{x}$ into a new coordinate $\mathbf{z}$ amenable to a parsimonious representation.

While in the original coordinate system a dynamical model may be dense in terms of functions of the original measurement coordinates $\mathbf{x}$, this method determines through an autoencoder a reduced coordinate system $\mathbf{z}(t) = \varphi(\mathbf{x}(t)) \in \mathbb{R}^r$ ($r \ll n$) where the following dynamical model holds:

$$\frac{\mathrm{d}\mathbf{z}(t)}{\mathrm{d}t} = \mathbf{g}(\mathbf{z}(t)). \quad (14.7)$$

Specifically, a parsimonious description of the dynamics is sought where $\mathbf{g}$ contains only a few active terms from a SINDy library. Thus, in addition to a dynamical model, the method learns coordinate transforms φ and ψ that map the measurements to intrinsic coordinates via $\mathbf{z} = \varphi(\mathbf{x})$ (encoder) and back via $\mathbf{x} \approx \psi(\mathbf{z})$ (decoder).

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

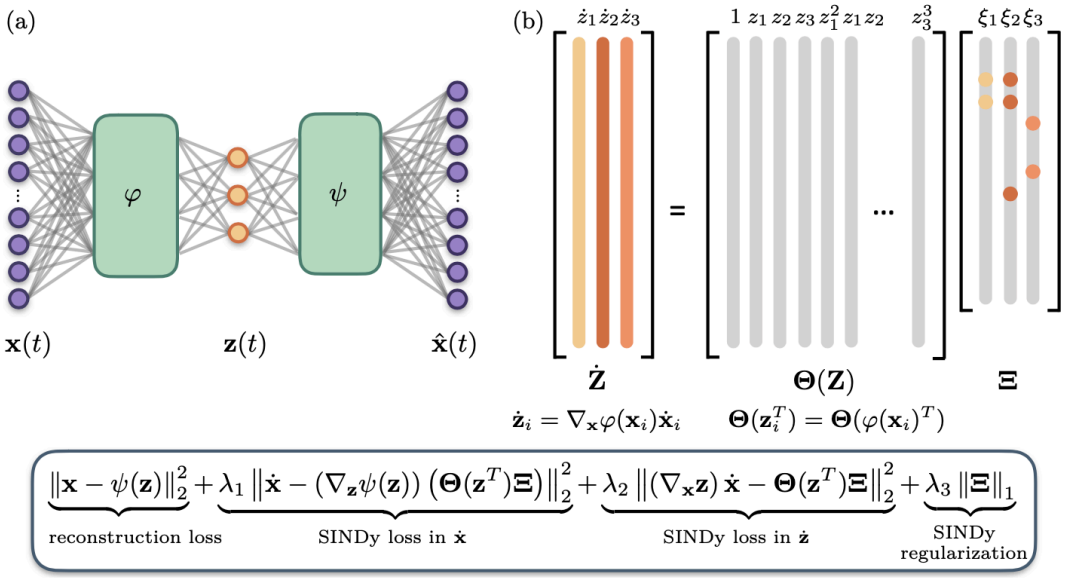


Figure 14.1: Schematic of the SINDy autoencoder method for simultaneous discovery of coordinates and parsimonious dynamics. (a) An autoencoder architecture is used to discover intrinsic coordinates $\mathbf{z}$ from high-dimensional input data $\mathbf{x}$. The network consists of two components: an encoder $\varphi(\mathbf{x})$, which maps the input data to the intrinsic coordinates $\mathbf{z}$, and a decoder $\psi(\mathbf{z})$, which reconstructs $\mathbf{x}$ from the intrinsic coordinates. (b) A SINDy model captures the dynamics of the intrinsic coordinates. The active terms in the dynamics are identified by the non-zero elements in $\boldsymbol{\Xi}$, which are learned as part of the NN training. The time derivatives of $\mathbf{z}$ are calculated using the derivatives of $\mathbf{x}$ and the gradient of the encoder φ. The bottom panel shows the pointwise loss function used to train the network. The loss function encourages the network to minimize both the autoencoder reconstruction error and the SINDy loss in $\mathbf{z}$ and $\mathbf{x}$. Also L_1 regularization on $\boldsymbol{\Xi}$ is included to encourage parsimonious dynamics. From Champion et al. [168].

The autoencoder is a flexible, feedforward neural network that allows one to discover underlying low-dimensional coordinates in which to represent the data. Thus the layers of the autoencoder learn a latent representation of a new variable in which to express the data, in this case the evolution dynamics. Often an autoencoder is used for classification and prediction. However, here, its targeted use is for learning a new coordinate system. Section 6.8 highlights the use of an autoencoder to discover a low-dimensional modal embedding for flow around a cylinder. The network is trained to output an approximate reconstruction of its input, and the restrictions placed on the network architecture (e.g., the type, number, and size of the hidden layers) characterize the

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

intrinsic coordinates [290]. The autoencoder gives a nonlinear generalization of a principal component analysis (PCA) [46].

In many science and engineering applications, the goal is to determine the underlying intrinsic coordinate system that best characterizes the data. For instance, in celestial mechanics, it took one and a half millennia to discover that a heliocentric coordinate system was a more appropriate choice of coordinates. This quickly led to the discovery of the $\mathbf{F} = m\mathbf{a}$ model of gravitation. Indeed, this pairing of coordinates and model is exactly what the SINDy autoencoder attempts to automate. In practice, it is common to discover intrinsic coordinates $\mathbf{z}$ that are much lower in dimension than the original state-space observations $\mathbf{x}$. The autoencoder learns a *nonlinear* embedding from measurement data $\mathbf{x}(t) \in \mathbb{R}^n$ to an intrinsic coordinate $\mathbf{z}(t) \in \mathbb{R}^r$, where $r \ll n$ is chosen as a hyperparameter prior to training the network.

Autoencoders can learn a low-dimensional representation in isolation without need to specify any other constraints. This is exactly what was done in Section 6.8 to embed fluid flow in a nonlinear coordinate system. Without further specifications, the intrinsic coordinates learned have no particular meaning or interpretation. However, if, in the latent space, additional constraints are imposed, then additional structure and meaning can be imposed on the model. For the SINDy autoencoder model, the network is required to learn coordinates associated with parsimonious dynamics. Thus it integrates the sparse regression framework of SINDy in the latent space, or intrinsic coordinates $\mathbf{z}$. This constraint in the autoencoder provides a regularization framework whereby model discovery is achieved by constructing a library $\boldsymbol{\Theta}(\mathbf{z}) = [\boldsymbol{\theta}_1(\mathbf{z}), \boldsymbol{\theta}_2(\mathbf{z}), \ldots, \boldsymbol{\theta}_p(\mathbf{z})]$ of candidate basis functions, e.g., polynomials, and learning a sparse set of coefficients $\boldsymbol{\Xi} = [\boldsymbol{\Xi}_1, \ldots, \boldsymbol{\Xi}_r]$ that defines the dynamical system

$$\frac{\mathrm{d}\mathbf{z}(t)}{\mathrm{d}t} = \mathbf{g}(\mathbf{z}(t)) = \boldsymbol{\Theta}(\mathbf{z}(t))\boldsymbol{\Xi}.$$

Typical of SINDy, the library is specified before training occurs, where library loadings (coefficients) $\boldsymbol{\Xi}$ are learned along with the autoencoder weights during training (optimization). Importantly, the derivatives $\dot{\mathbf{x}}(t)$ of the original states are computed in order to pass these along to the encoder variables as $\dot{\mathbf{z}}(t) = \nabla_{\mathbf{x}}\varphi(\mathbf{x}(t))\dot{\mathbf{x}}(t)$. This helps enforce accurate prediction of the dynamics by incorporating the loss function:

$$\mathcal{L}_{\mathrm{d}\mathbf{z}/\mathrm{d}t} = \|\nabla_{\mathbf{x}}\varphi(\mathbf{x})\dot{\mathbf{x}} - \boldsymbol{\Theta}(\varphi(\mathbf{x})^T)\boldsymbol{\Xi}\|_2^2. \tag{14.8}$$

This term uses both the typical SINDy regression along with the gradient of the encoder to promote learning of a sparse dynamical model which accurately predicts the time derivatives of the encoder variables. Additional loss terms require that the SINDy predictions accurately reconstruct the time derivatives

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

of the original data:

$$\mathcal{L}_{\mathrm{d}\mathbf{x}/\mathrm{d}t} = \|\dot{\mathbf{x}} - (\nabla_{\mathbf{z}}\psi(\varphi(\mathbf{x})))(\boldsymbol{\Theta}(\varphi(\mathbf{x})^T)\boldsymbol{\Xi})\|_2^2. \tag{14.9}$$

These loss terms (14.8) and (14.9) are added to the standard autoencoder loss

$$\mathcal{L}_{\text{recon}} = \|\mathbf{x} - \psi(\varphi(\mathbf{x}))\|_2^2,$$

which ensures that the autoencoder can accurately reconstruct the original input data. To help promote sparsity in the SINDy architecture, an ℓ_1 regularization penalty is included on the SINDy coefficients $\boldsymbol{\Xi}$. This promotes a parsimonious model for the dynamics by selecting only a small number of terms. The combination of the above four terms gives the following overall loss function:

$$\mathcal{L}_{\text{recon}} + \lambda_1\mathcal{L}_{\mathrm{d}\mathbf{x}/\mathrm{d}t} + \lambda_2\mathcal{L}_{\mathrm{d}\mathbf{z}/\mathrm{d}t} + \lambda_3\mathcal{L}_{\text{reg}},$$

where the hyperparameters λ_1, λ_2, and λ_3 determine the relative weighting of the three terms in the loss function.

In addition to the ℓ_1 regularization, sequential thresholding has been shown to be an effective proxy for the ℓ_0-norm [775]. This technique is inspired by the original algorithm used for SINDy [132], which combined least-squares fitting with sequential thresholding to obtain a sparse model. Thresholding is applied at fixed intervals throughout the training, with all coefficients below the threshold being set to zero and training resuming using only the terms left in the model. The Adam optimizer [386] provides a robust framework for the optimization procedure. In addition to the loss function weightings and SINDy coefficient threshold, training requires the choice of several other hyperparameters, including learning rate, number of intrinsic coordinates r, network size, and activation functions [168]. Figure 14.2 shows the SINDy autoencoder method applied to a video of a pendulum. From the video, it is able to learn the underlying variables that characterize the pendulum motion, i.e., the angle and its angular velocity. These latent state-space variables are learned by enforcing SINDy, thus producing the coordinates and dynamics $\ddot{z} = -\sin z$ of the model correctly. This framework allows for going straight from videos to physics discovery models.

The basic architecture developed by Champion et al. [168] is highly flexible. Indeed, it has already been illustrated in Section 7.4 to encode a Koopman operator [465]. Figure 14.3 shows how it can also be used to embed the dynamics into its normal-form dynamics near instabilities [369]. Normal forms are exceptional representations of the dynamics, as they capture the underlying intrinsic behavior with minimal parameterization. They also highlight the nature of underlying instabilities, which is a critical component for understanding pattern forming systems, for instance.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

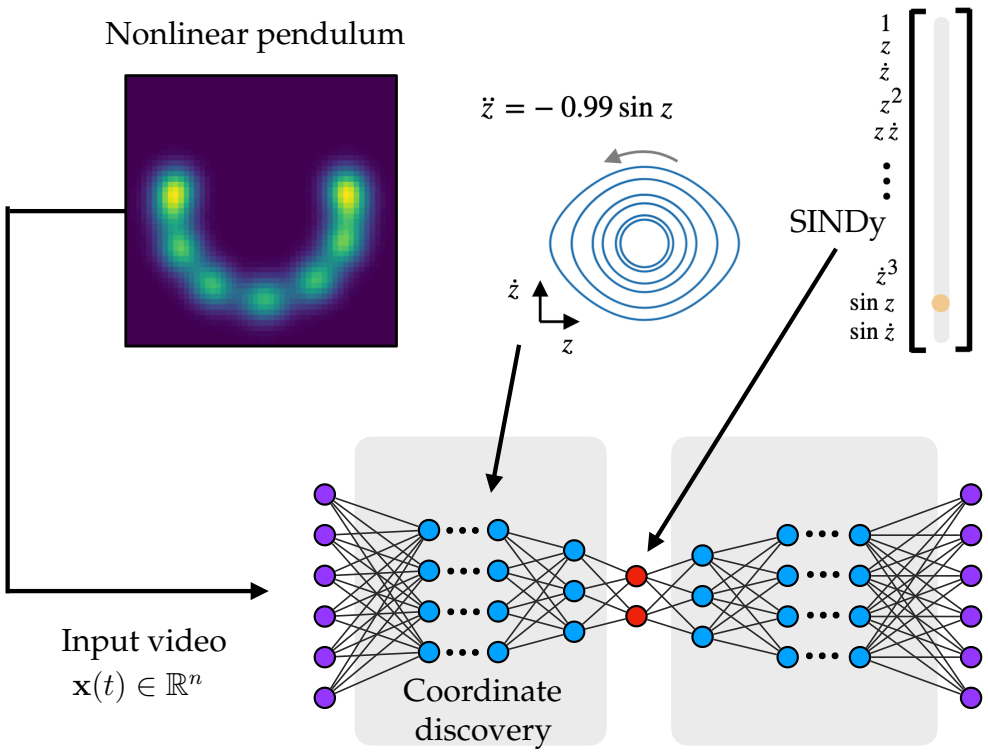


Figure 14.2: Illustration of the SINDy autoencoder method whereby high-dimensional input data $\mathbf{x}(t) \in \mathbb{R}^n$ (video of a pendulum) is used as the input data stream. The autoencoder constructs a low-rank embedding and enforces SINDy. As such, it discovers a latent variable space $\mathbf{z}(t) \in \mathbb{R}^r$ where $r = 2$. Specifically, it discovers the angle of the pendulum (z) and its angular velocity $\dot{z}$. SINDy then learns (approximates) the underlying dynamics $\ddot{z} = -\sin z$.

14.3 Koopman Forecasting

Dynamic mode decomposition and Koopman theory have already been introduced in previous chapters. Highlighted here is an extension of Koopman theory whereby neural networks transform time-series data into a form more amenable to a Koopman representation [428]. Thus, instead of transforming the spatial coordinate system as in the last section, here a transformation of time is learned, whereby the temporal evolution is made to be as sinusoidal as possible. In its simplest form, the underlying optimization is given by

$$\underset{\mathbf{A},\mathbf{B}}{\operatorname{argmin}} \sum_{k=1}^{m} (\mathbf{x}_k - \mathbf{A}\mathbf{z}_k)^2 \quad \text{subject to} \quad \mathbf{z}_{k+1} = \mathbf{B}\mathbf{z}_k \tag{14.10}$$

for data snapshots $\mathbf{x}_k$, model snapshots $\mathbf{z}_k$, and $k = 1, 2, \ldots, m$. This optimization framework is similar to DMD [422] which regresses to a best fit linear

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

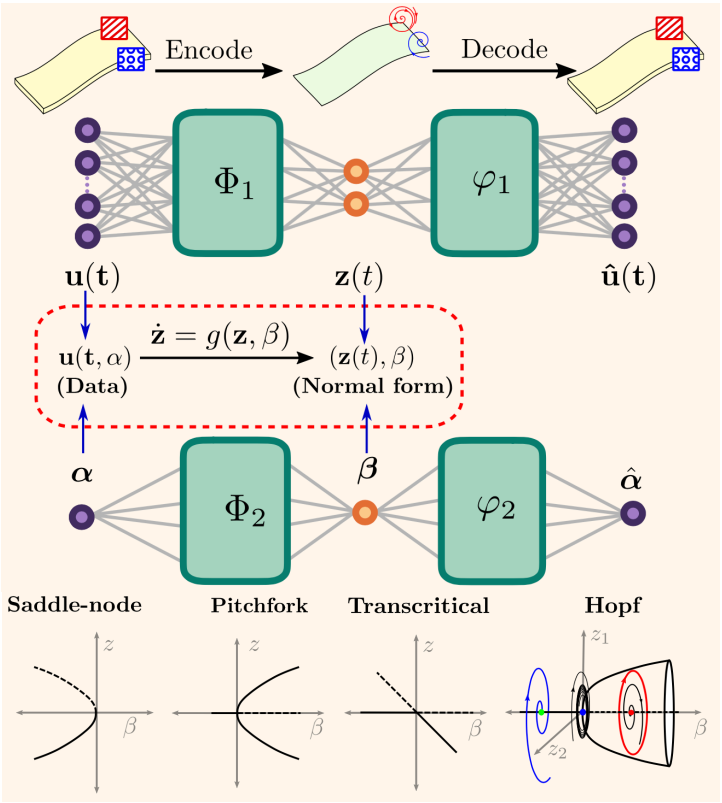


Figure 14.3: Instabilities lead to canonical pattern formation in various physical systems that are characterized by underlying normal forms. The autoencoder collapses data to the underlying normal-form coordinates (z, β), with bifurcation parameter β. The dynamics on the reduced coordinates (z, β) are given by normal-form equations, which are typically given by four different canonical forms. From Kalia et al. [369].

model. However, in this formulation, both a linear dynamic model $\mathbf{B}$ is learned along with a linear mapping to the data $\mathbf{A}$, allowing the mapping to be generalized to a neural network embedding. It is assumed that the data is collected over a time frame $t \in [0, T]$. In the context of forecasting, it is typically advantageous to produce long-term forecasts of a system, which would further require enforcing $\Re\{\text{Eig}(\mathbf{B})\} = 0$. Such a constraint guarantees that the solutions do not decay to zero or grow to infinity. Enforcing this constraint allows us to rewrite

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

the original optimization problem as

$$\underset{\mathbf{A},\boldsymbol{\omega}}{\operatorname{argmin}} \sum_{k=1}^{m} \left(\mathbf{x}_k - \mathbf{A} \begin{bmatrix} \sin(\omega_1 t_k) \\ \vdots \\ \sin(\omega_N t_k) \\ \cos(\omega_1 t_k) \\ \vdots \\ \cos(\omega_N t_k) \end{bmatrix} \right)^2 = \underset{\mathbf{A},\boldsymbol{\omega}}{\operatorname{argmin}} \sum_{k=1}^{m} (\mathbf{x}_k - \mathbf{A}(\boldsymbol{\Omega}(\boldsymbol{\omega} t_k)))^2, \quad (14.11)$$

where the model fit is to N distinct frequencies. This is a constrained version of Koopman. Koopman generically fits to exponentials, allowing for real and imaginary parts of the eigenvalues. This fitting procedure is constrained only to the imaginary part, which, as will be shown, allows for an exceptional forecasting tool for data that is periodic or quasi-periodic in nature.

An obvious connection to make is with the Fourier transform, and more specifically its computational engine, the *fast Fourier transform* (FFT). FFT also transforms a given time series into a frequency representation. The FFT constructs its representation with frequencies that are periodic on the time interval $t \in [0, T]$. This is problematic for signals that display only a fraction of a period. Specifically, the Gibbs phenomenon is generated due to the periodic continuation enforced by the FFT. Thus many high frequencies are generated which are artificial in nature, since the solution is forced to be periodic on $t \in [0, T]$. This makes forecasting with FFT difficult and inaccurate unless the data is sampled perfectly on periodic data. The regression (14.11) provides a more general framework, as the frequencies are determined during optimization and no underlying periodicity on the time interval $t \in [0, T]$ is assumed. As a consequence, a non-convex optimization must then be performed, which is often detrimental for gradient descent methods, which get stuck in local minima. To overcome the issues with non-convex optimization, the FFT is used to seed the gradient descent algorithm used for (14.11), thus providing a more stable algorithm for the frequency fitting procedure. This allows the *global* properties of the FFT to inform the *local* frequencies that should be optimized upon in gradient descent [428].

In addition to generalizing the FFT, Lange et al. [428] go further and *warp* the time-series data to be more amenable to Fourier analysis. Specifically, neural networks are used to transform data from its original form into data that is more sinusoidal in nature with as few frequencies as possible. This is done by replacing the linear operator $\mathbf{A}$ in (14.11) with a nonlinear (neural network)

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Table 14.1: Performance in long-term forecasting of distribution-level energy consumption as measured by the relative cumulative error for various algorithms. Note that long-term predictions are obtained by recursively feeding predictions back into algorithms where applicable. Furthermore, the column "Patterns" indicates whether the algorithms at hand have successfully extracted daily (D), weekly (W) or yearly (Y) patterns.

Algorithm	**Forecast horizon**				**Patterns**		
	25%	50%	75%	100%	D	W	Y
Koopman forecast	**0.19**	**0.21**	**0.19**	**0.19**	✓	✓	✓
Fourier forecast	0.31	0.39	0.33	0.30	✓	✓	✓
LSTM	0.37	0.4	0.42	0.45	✓	✗	✗
GRU	0.53	0.55	0.52	0.50	✓	✗	✗
Echo state network	0.67	0.73	0.76	0.73	✓	✗	✗
AR$(1, 12, 24, 168, 4380, 8760)$	0.75	0.95	1.07	1.13	✓	✓	✓
CW-RNN (data clocks)	1.10	1.14	1.14	1.15	(✓)	✗	✗
CW-RNN	1.05	1.08	1.08	1.09	(✓)	✗	✗
AutoARIMA	0.83	1.11	1.18	1.26	✗	✗	✗
Temporal convolutional nets	0.96	1.69	1.87	2.33	✓	(✓)	✗
Fourier neural networks	1.10	1.15	1.21	1.21	✓	✗	✗

transformation

$$\underset{\mathbf{A},\boldsymbol{\omega}}{\operatorname{argmin}} \sum_{k=1}^{m} \left(\mathbf{x}_k - \mathbf{f}_{\boldsymbol{\theta}} \left(\begin{bmatrix} \sin(\omega_1 t_k) \\ \vdots \\ \sin(\omega_N t_k) \\ \cos(\omega_1 t_k) \\ \vdots \\ \cos(\omega_N t_k) \end{bmatrix} \right) \right)^2 = \underset{\mathbf{A},\boldsymbol{\omega}}{\operatorname{argmin}} \sum_{k=1}^{m} (\mathbf{x}_k - \mathbf{f}_{\boldsymbol{\theta}}(\boldsymbol{\Omega}(\boldsymbol{\omega} t_k)))^2, \tag{14.12}$$

where $\mathbf{f}_{\boldsymbol{\theta}}$ defines the neural network that is learned for best representing the signal in N learned frequencies. This problem is nonlinear and non-convex, yet global optima can be computed [428]. Indeed, the loss function is periodic in nature, and this is exploited in the training process as well.

The Koopman forecasting method that is enabled by training (14.12) provides a mid- to long-term forecasting algorithm that has superior performance. Table 14.1 compares a number of techniques on power grid data, from leading statistical time-series methods to state-of-the-art machine learning algorithms for time series, against the Koopman forecasting tool (14.12) along with its Fourier forecasting counterpart (14.11). Both methods provide superior performance, with the Koopman forecasting producing improvements that are sig-

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

nificant.

A number of further examples are given in Fig. 14.4. Specifically, it shows the performance of the Fourier-based algorithm graphically, predicting the last time frame for a Kolmogorov 2D flow, flow around a cylinder, flow over a cavity, and a video of a fan. The data from the Kolmogorov 2D flow was taken from the experiments conducted in [705]. Note that the Kolmogorov 2D flow and the data for the video frame prediction task constitute real measurements and therefore exhibit a considerable amount of noise, whereas the cylinder and cavity flow data are from simulation. The codes for the Fourier and Koopman forecasting algorithms are available at `https://github.com/helange23/from_fourier_to_koopman`.

14.4 Learning Nonlinear Operators

The universal approximation capabilities of neural networks are well known. Specifically, neural networks can generically approximate any continuous function. More recently, Lu et al. [461] (*DeepONet*) and Kovachki et al. [407] (*neural operator*) have highlighted results by Chen and Chen [174] that prove that neural networks with a single hidden layer can accurately approximate any nonlinear continuous operator. Thus a nonlinear operator is learned mapping functions to functions. In practice, this is perhaps an even more impactful theory, as the operator is often the more important quantity to compute, since the operator contains information about the physics and dynamics of the system. Note that this approach is fundamentally different than what was considered in the previous section with Koopman theory. Koopman theory attempts to approximate the dynamics with a linear operator while Lu et al. [461] and Kovachki et al. [407] directly construct a nonlinear operator using neural networks.

The original work of Chen and Chen [174] constructs a universal approximation proof of an operator that DeepONet constructs through training. The theorem is the following:

Theorem (Universal approximation theorem for operators). *Suppose that σ is a continuous non-polynomial function, X is a Banach space, $K_1 \subset X$ and $K_2 \subset \mathbb{R}^d$ are two compact sets in X and $\mathbb{R}^d$, respectively, V is a compact set in $C(K_1)$, and $\mathcal{G}$ is a nonlinear continuous operator, which maps V into $C(K_2)$. Then, for any $\epsilon > 0$, there are positive integers n, p, m, constants $c_i^k, \xi_{ij}^k, \theta_i^k, \zeta_k \in \mathbb{R}$, $\omega_k \in \mathbb{R}^d$, and $\mathbf{x}_j \in K_1$, where $i = 1, 2, \ldots, n$, $k = 1, 2, \ldots, p$, and $j = 1, 2, \ldots, m$, such that*

$$\left| \mathcal{G}(u)(y) - \sum_{k=1}^{p} \sum_{i=1}^{n} c_i^k \sigma \left(\sum_{j=1}^{m} \xi_{ij}^k u(x_j) + \theta_i^k \right) \sigma(\omega_k y + \zeta_k) \right| < \epsilon \tag{14.13}$$

holds for all $u \in V$ and $y \in K_2$.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

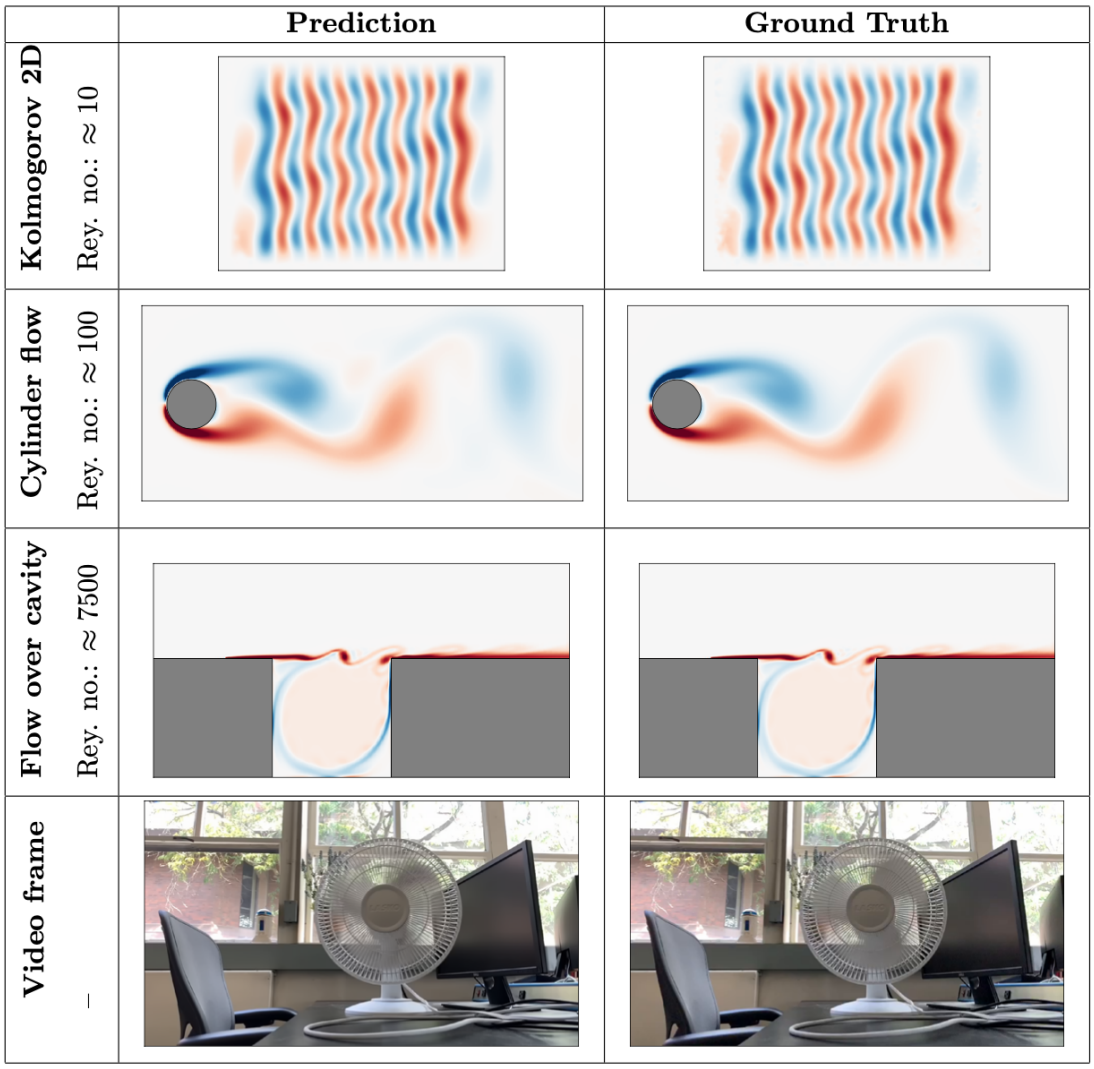


Figure 14.4: The last frame as predicted by PCA in conjunction with the Fourier-based algorithm of fluid flows and video frame prediction. For a video that shows the performance, visit `https://www.youtube.com/watch?v=trbXYMqi2Tw`. From Lange et al. [428].

The theorem provides theoretical bounds on the ability of a neural network to approximate the operator $\mathcal{G}(\cdot)$. The theorem also highlights the construction of two neural networks, so that it can be more compactly represented as

$$|\mathcal{G}(\mathbf{u})(\mathbf{y}) - \mathbf{f}_{\boldsymbol{\theta}_1}(\mathbf{u}) \cdot \tilde{\mathbf{f}}_{\boldsymbol{\theta}_2}(\mathbf{y})| < \epsilon \tag{14.14}$$

when considering the discretized representation of $u(x) \rightarrow \mathbf{u}$ and new measurement (function evaluation) locations $y \rightarrow \mathbf{y}$. Figure 14.5 highlights the neural

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

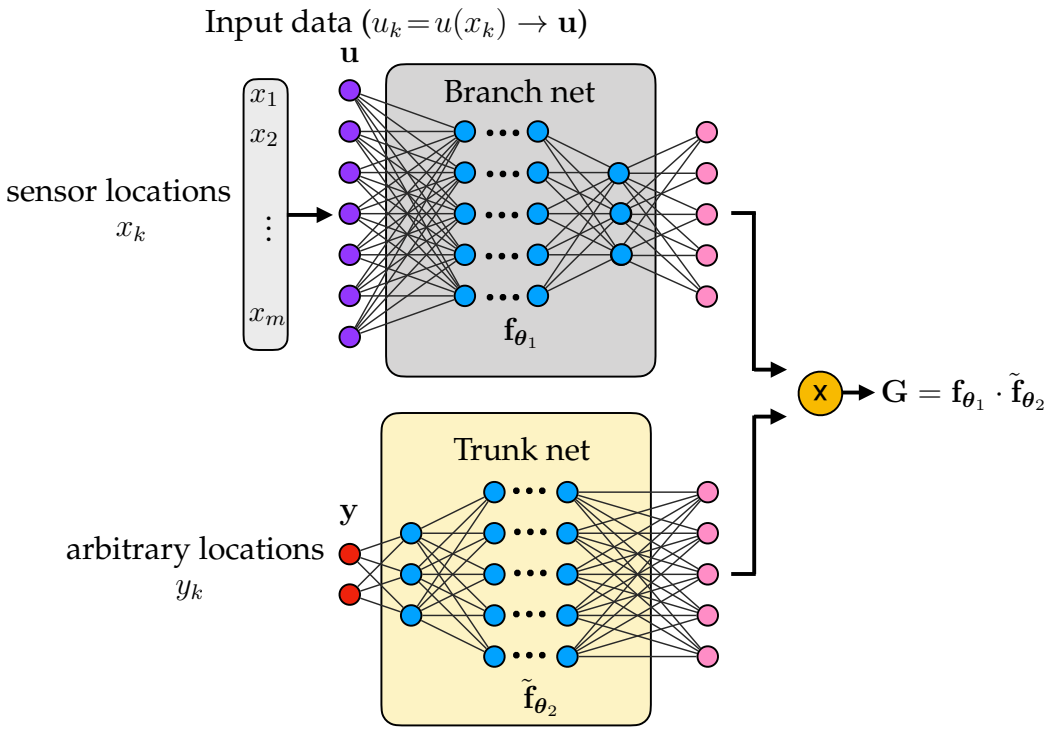


Figure 14.5: Architecture for learning nonlinear operators. The DeepONet trains two networks: (i) a branch network $\mathbf{f}_{\theta_1}$ that maps the original field variable $\mathbf{u}$ evaluated at m measurement points x_k (where $u_k = u(x_k)$) to a latent representation $\mathbf{H}_b$; and (ii) a trunk network $\tilde{\mathbf{f}}_{\theta_2}$ that maps arbitrary and new spatial locations $\mathbf{y}$ to a latent representation $\mathbf{H}_t \in$ ı. The operator is then given by the expression from Chen and Chen [174] as $\mathbf{G} = \mathbf{f}_{\theta_1} \cdot \tilde{\mathbf{f}}_{\theta_2}$.

network trained by Lu et al. [461] that leverages the universal operator approximation theorem of Chen and Chen [174]. The two simultaneously trained networks are called the branch network $\mathbf{f}_{\theta_1}(\mathbf{u})$ and the trunk network $\tilde{\mathbf{f}}_{\theta_2}(\mathbf{y})$.

Mathematically, the concept is quite simple. Given a number of measurement (sensor) locations x_k (usually selected from a computational grid) which prescribes the input function $u_k = u(x_k)$, a vector of training input data $\mathbf{u}$ can be constructed. The input data has corresponding output data $\mathcal{G}(\mathbf{u})$. In addition, training data mapping selections of random measurement points $\mathbf{y}$ to the output $\mathcal{G}(\mathbf{u})(\mathbf{y})$ is required. Thus the input functions $\mathbf{u}$ are encoded in a separate network than the location variables $\mathbf{y}$. These are merged at the end, as shown in the universal approximation proof of Chen and Chen [174]. Figure 14.6 shows the results of training from the original DeepONet paper of Lu et al. [441, 461] on a reaction–diffusion system. DeepONet also can achieve small generalization errors by employing inductive biases. Remarkably, exponential convergence is observed in the deep learning algorithm. The code is

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

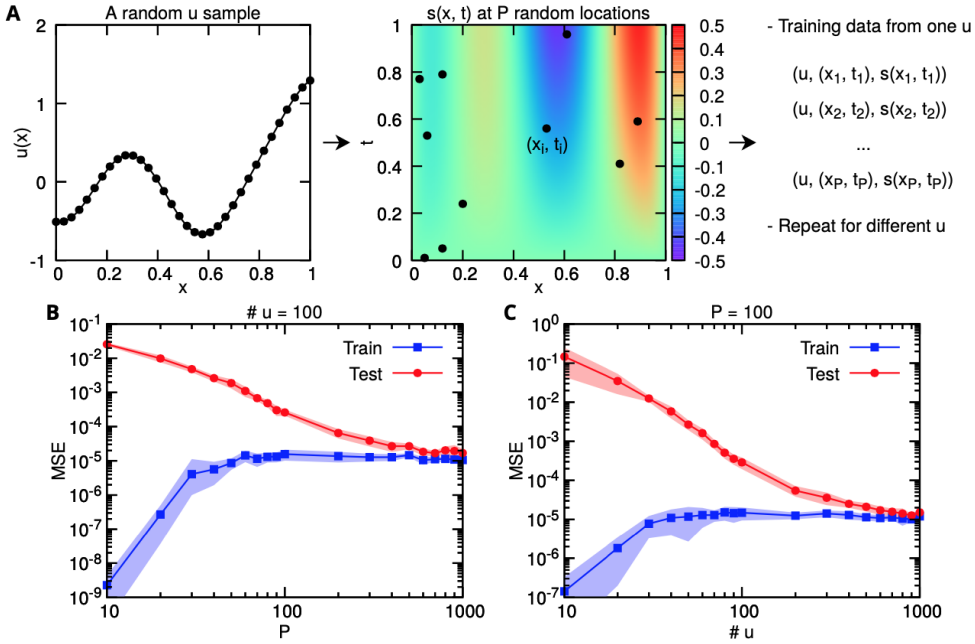


Figure 14.6: Learning a reaction–diffusion system with DeepONet. (A) An example of a random sample of the input function $u(x)$ (left). The corresponding output function $s(x,t)$ at P different (x,t) locations (middle). Pairing of inputs and outputs at the training data points (right). The total number of training data points is the product of P times the number of samples of u. (B) Training error (blue) and test error (red) for different values of the number of random points P when 100 random u samples are used. (C) Training error (blue) and test error (red) for different number of u samples when $P = 100$. The shaded regions denote one standard deviation. From Lu et al. [461].

available at `https://github.com/lululxvi/deepxde`.

Neural operators are a closely related method for producing mappings between function spaces, thus allowing for the approximation of operators that encode governing equations and physics [407, 441, 442, 443]. Neural operators generalize standard feedforward neural networks to learn mappings between infinite-dimensional spaces of functions defined on bounded domains of $\mathbb{R}^d$. The non-local component of the architecture is instantiated either through a parameterized integral operator or through multiplication in the spectral domain, which is a specific form of the kernel in the integral operator. As with DeepONet, neural operators, once trained, have the property of being discretization-invariant: sharing the same network parameters between different discretizations of the underlying functional data. This is their specific advantage: neural operators and DeepONets are mesh-free methods once trained.

Neural operators have a different structure than DeepONet. Specifically,

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

they leverage an integral kernel representation in their approximation of the operator. For instance, neural operators can make explicit use of multi-pole [442] and Fourier [441] kernels in order to construct operator representations. Thus nonlocal representations of the solution are parameterized by the integral operator. Recall that learning a nonlinear operator $\mathcal{G}(\cdot)$ is equivalent to learning the inverse of the PDE evolution $\dot{\mathbf{u}} = \mathbf{N}(\mathbf{u})$. Thus kernel operators are intuitively appealing for the construction of the nonlinear operator. The overall representation of the operator is a trained neural network

$$\mathbf{G} = \mathbf{f}_{\boldsymbol{\theta}} \tag{14.15}$$

where individual layers of the neural network are constructed from learned integral representations that are updated according to the following:

$$u_{k+1}(x) = \sigma_{k+1}\left(W_t u_k + \int_{D_k} K^{(k)}(x,y)u_k(y)\,\mathrm{d}\nu_k(y) + b_k(x)\right) \tag{14.16}$$

where ν_k is a Lebesgue measure on $\mathbb{R}^{d_t}$. The kernel $K^{(k)}(x,y)$ is typically chosen to leverage advantageous representations, such as the multi-pole or Fourier kernels. Thus each layer of the network is trained using a physics-inspired concept of an integral (inverse) representation of the PDE dynamics. The kernel representation is strongly motivated by the concept of the Green's function, which provides a fundamental solution for linear PDEs by expressing the solution as an integration over the Green's function kernel.

Although both neural operators and DeepONets accomplish the same goal, they do so with different architectures. Neural operators exploit the kernel structure of generic operators, while DeepONets train by separating the input function from the spatial locations. Both have achieved promising results, highlighting the fact that the learning of operators can potentially allow for mesh-free models of physics systems. Of course, in order for this to be viable in practice, exceptional large training data that resolves all scales is required for training. Figure 14.7 highlights the results from Kovachki et al. [407] where neural operators are used to model fluid flows. The codes are available at `https://github.com/zongyi-li/graph-pde` and `https://github.com/zongyi-li/fourier_neural_operator`.

14.5 Physics-Informed Neural Networks (PINNs)

An elegant solution technique for solving many physics-based problems is the *physics-informed neural network* (PINN) pioneered by Raissa, Perdikaris, and Karniadakis [584]. The method is simple in concept: find a solution to a PDE by enforcing satisfaction of the PDE in the neural network loss function. The

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

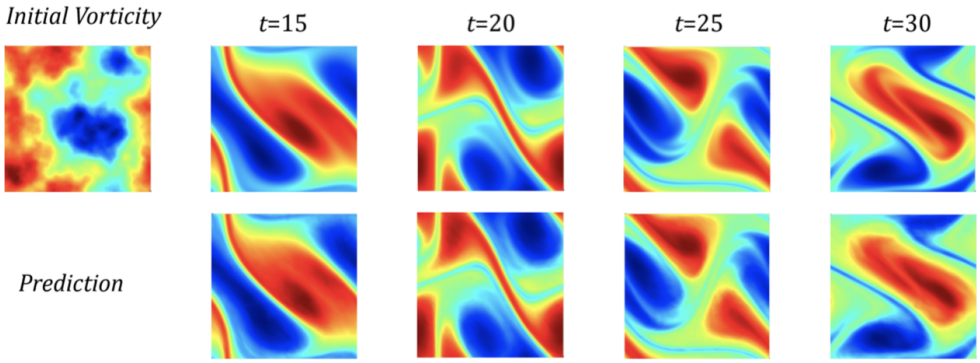


Figure 14.7: Zero-shot super-resolution. The vorticity field of the solution to the two-dimensional Navier–Stokes equation with viscosity 10^4 ($Re = O(200)$). The ground truth is on the top and prediction on the bottom. The model is trained on data that is discretized on a uniform 64×64 spatial grid and a 20-point uniform temporal grid. The model is evaluated with a different initial condition that is discretized on a uniform 256×256 spatial grid and an 80-point uniform temporal grid. From Kovachki et al. [407].

method can be used both for approximating the solution of a PDE and also for system identification, much like the SINDy algorithm.

To be more mathematically precise, we again consider generically a system of nonlinear PDEs of a single spatial variable that can be modeled as

$$\mathbf{u}_t = \mathbf{N}(\mathbf{u}, \mathbf{u}_x, \mathbf{u}_{xx}, \ldots, x, t; \boldsymbol{\beta}) + \mathbf{g}, \tag{14.17}$$

where the subscripts denote partial differentiation, $\mathbf{g}$ is a forcing, and $\mathbf{N}(\cdot)$ prescribes the generically nonlinear evolution. The parameter $\boldsymbol{\beta}$ will represent a bifurcation parameter for our later considerations. Further, associated with (14.17) are a set of initial and boundary conditions on a domain $x \in [-L, L]$.

PINNs define the function

$$\mathbf{f} := \mathbf{u}_t - \mathbf{N}(\mathbf{u}, \mathbf{u}_x, \mathbf{u}_{xx}, \ldots, x, t; \boldsymbol{\beta}). \tag{14.18}$$

The original formulation by Raissa et al. [584] was generalized to the representation of the PDE as $\mathcal{L}(\mathbf{u}, \boldsymbol{\theta})$. Figure 14.8 highlights the basic architecture. The goal is to determine an approximation $\tilde{\mathbf{u}}$ to the spatio-temporal data $\mathbf{u}$. Not only should the approximation satisfy the PDE, it should also fit the actual data and its boundary and initial conditions. To be more precise, there is a neural network trained to map the spatio-temporal location to the data. The loss function for this is given by

$$\mathcal{L}_{\mathbf{u}} = \sum_{k=1}^{N} \|\mathbf{u} - \tilde{\mathbf{u}}\|. \tag{14.19}$$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

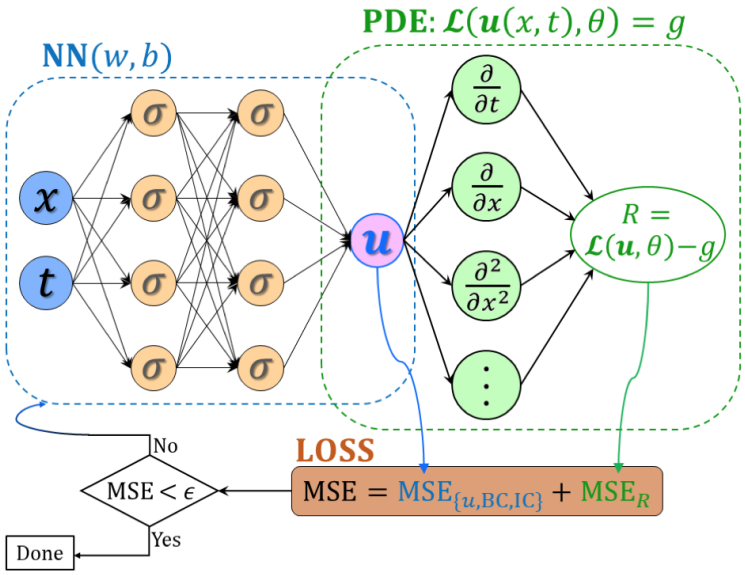


Figure 14.8: Schematic of a PINN architecture, where the loss function of the trained neural network contains a mismatch in the given data on the state variables and/or boundary and initial conditions. In addition, the loss function is required to minimize the loss satisfying the PDE evolution. From Meng et al. [496].

Note that by choosing data points at $x = \pm L$ and/or $t = 0$, the boundary and initial conditions are satisfied, respectively. In addition, the approximation should satisfy the PDE

$$\mathcal{L}_{\mathbf{f}} = \sum_{k=1}^{N} \|\mathbf{f} - \mathbf{g}\|. \tag{14.20}$$

The neural network is then trained to minimize the loss functions. In summary: one trains the network to find a solution $\tilde{\mathbf{u}}$ that best matches the data and satisfies the PDE.

The PINN architecture was also used to perform system identification tasks, i.e., discover the underlying governing equations [584]. In this case, the PDE is formulated in much the same way as the SINDy algorithm, where now

$$\mathbf{f} := \mathbf{u}_t - \boldsymbol{\theta}(\mathbf{u})\boldsymbol{\Xi}, \tag{14.21}$$

and the coefficients $\boldsymbol{\Xi}$ are also determined in the regression process. Figure 14.8 shows the potential library terms in green, which can be used to construct the governing equations. The loadings $\boldsymbol{\Xi}$ dictate which terms contribute. In the original work, the library of dynamic terms $\boldsymbol{\theta}$ was quite limited, unlike SINDy, which builds a large library of potential terms.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

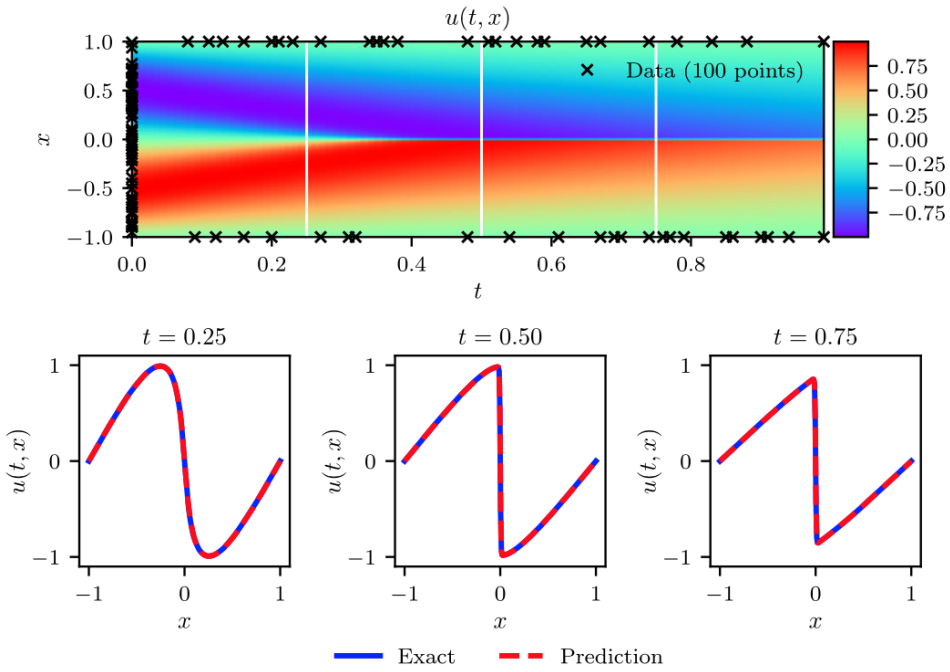
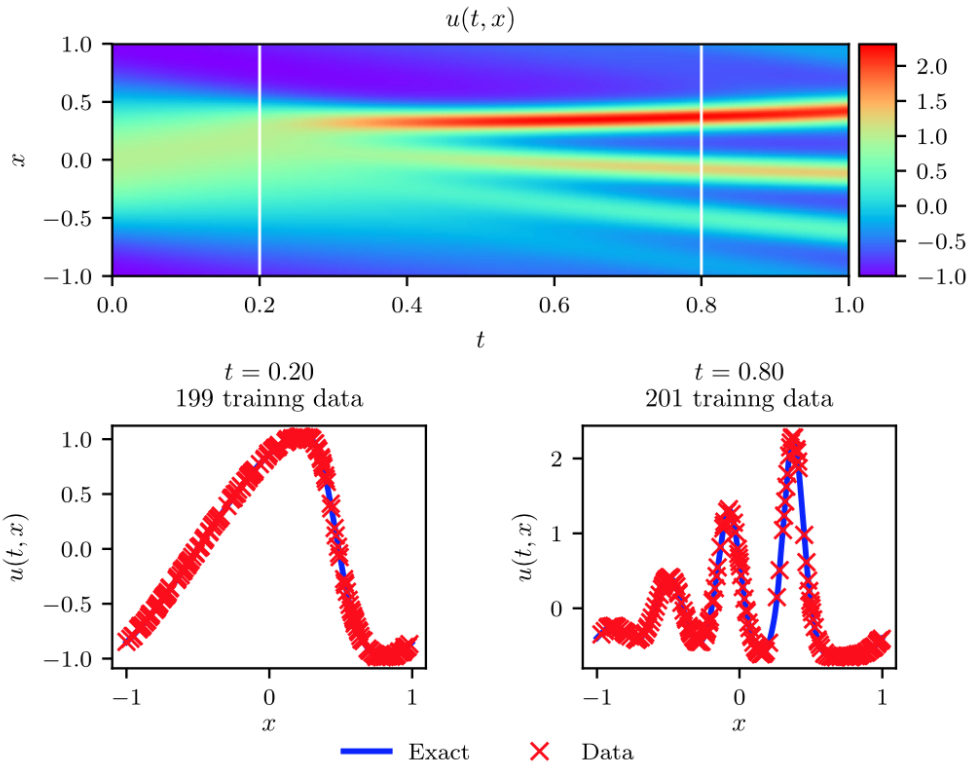


Figure 14.9: Burgers' equation. (top) Predicted solution $\mathbf{u}(x,t)$ along with the initial and boundary training data. In addition, 10 000 collocation points were used as data generated using a Latin hypercube sampling strategy. (bottom) Comparison of the predicted and exact solutions corresponding to the three temporal snapshots depicted by the white vertical lines in the top panel. From Raissa et al. [584].

Two figures illustrate the power of the PINNs. In the first (Fig. 14.9), training data is used to build a representation of Burgers' equation. The PINN model converges to an accurate representation of the PDE dynamics and can then serve as a proxy to computational data. In fact, one simply needs to specify time and space to produce a value of the field $\mathbf{u}$. In the second example (Fig. 14.10), the Korteweg–de Vries (KdV) equation is analyzed with the PINN. The PINN not only produces an accurate neural network proxy for the spatio-temporal field, but also further identifies the PDE dynamics by identifying the parameters in front of the appropriate terms. The success, modularity, and simplicity of PINNs has led to significant advancements and extensions of the method, many of which are reviewed by Karniadakis et al. [375]. Krishnapriyan et al. [413] have also recently modified PINN architectures with improved regularization to help make them more amenable to complex systems.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.



Correct PDE	$u_t + uu_x + 0.0025u_{xxx} = 0$
Identified PDE (clean data)	$u_t + 1.000uu_x + 0.0025002u_{xxx} = 0$
Identified PDE (1% noise)	$u_t + 0.999uu_x + 0.0024996u_{xxx} = 0$

Figure 14.10: KdV equation. (top) Solution $\mathbf{u}(x,t)$ along with the temporal locations of the two training snapshots. (middle) Training data and exact solution corresponding to the two temporal snapshots depicted by the dashed vertical lines in the top panel. (bottom) Correct partial differential equation along with the identified one obtained by learning $\boldsymbol{\Xi}$. From Raissa et al. [584].

14.6 Learning Coarse-Graining for PDEs

The modeling of multi-scale physics remains particularly challenging due to the need of numerical algorithms to resolve spatial and temporal scales that can vary across many orders of magnitude. Even if one is interested in macro-scale phenomena, accurate models are only produced by resolving the fastest timescales and finest spatial resolutions. This generates significant computational expense. Two common methods have been used to circumvent this computational expense: multi-grid methods and coarse graining. Multi-grid methods [494, 721], for instance, have been extensively developed for physics-based simulation models, where coarse-grained models must be progressively refined

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

in order to achieve a required numerical precision while keeping the simulation tractable. Multi-grid architectures provide a principled method for targeting the refinement process, constituting a mature field with widespread applications in the engineering and physical sciences. In contrast, coarse-graining methods attempt to construct a macro-scale physics model by progressive construction of coarse-grained variables and their dynamics. Mathematical algorithms such as the *heterogeneous multi-scale modeling* (HMM) [748, 749] and *equation-free method* [383] provide principled methods for multi-scale systems. Additional work has focused on testing for the presence of multi-scale dynamics, so that analyzing and simulating multi-scale systems is more computationally efficient [254, 255].

Deep learning and neural networks provide an alternative to these multi-scale modeling efforts. In this case, the goal is to train a neural network to coarse-grain a model directly from data. Featured here is work by Bar-Sinai et al. [52], who, instead of deriving an approximate coarse-grained continuum model and discretizing it, suggest directly learning low-resolution discrete models that encapsulate unresolved physics. Consider the governing PDE (14.17). Numerical discretization immediately turns the continuous PDE into an n-dimensional system of coupled differential equations. This is best illustrated by finite-difference discretization. Finite-difference methods generate a solution vector $\mathbf{u} = [u_1\ u_2\ \cdots\ u_n]^T$, where $u_k = u(x_k)$. Derivatives are then computed by using differences in u_k. For instance, the first derivative is given by

$$\frac{\partial u_k}{\partial x} \approx \frac{u_{k+1} - u_{k-1}}{2\Delta x}, \tag{14.22}$$

where $\Delta x = x_{k+1} - x_k$. Thus the value of the field at u_k depends on the neighbors $u_{k\pm1}$. This creates a global coupling between all discretization points. Of course, there are alternatives to differentiation using finite differences, including polynomial expansions and spectral methods. But each, in turn, generates coupling between n differential equations. For instance, spectral methods generate coupling in global Fourier modes [420].

Ultimately, differentiation schemes result in a general expression of the form

$$\frac{\partial^p u_k}{\partial x^p} \approx \sum_{k=1}^{n} \alpha_k^{(p)} u_k, \tag{14.23}$$

where $\alpha_k^{(p)}$ are pre-computed coefficients from a prescribed differentiation scheme. Importantly, error estimates are directly related to the spatial discretization Δx. The discretization must be small enough to resolve all spatial scales, and thus it sets the resolution limit and computational time required to solve the PDE. Similarly, discretization of time generates a corresponding Δt, which is related to Δx through a Courant–Friedrichs–Lewy (CFL) condition for the stability of a numerical scheme [420].

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

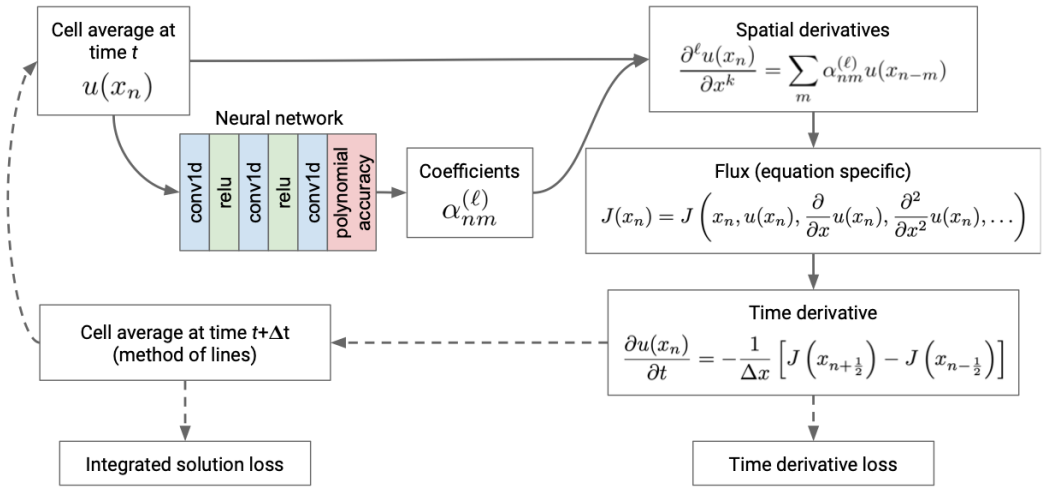


Figure 14.11: Neural network architecture. During training, the model is optimized to predict cell-average time derivatives or time-evolved solution values from cell-average values, based on a pre-computed data set of snapshots from high-resolution simulations. During inference, the optimized model is repeatedly applied to predict the time evolution using the method of lines. From Bar-Sinai et al. [52].

Standard schemes use one set of pre-computed coefficients for all points in space, while more sophisticated methods alternate between different sets of coefficients according to local rules. Regardless, the governing PDE (14.17) is now the high-dimensional system of ODEs

$$\frac{\mathrm{d}\mathbf{u}_k}{\mathrm{d}t} = \mathbf{N}(\mathbf{u}, \mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_n, x, t; \boldsymbol{\beta}). \tag{14.24}$$

One criticism of this discretized model is the computational cost of simulating it if there are significant time and space scales that need to be resolved.

Bar-Sinai et al. [52] learn directly from data a flexible parameterization of the derivative (14.23). The training data in this case are highly resolved simulations of the multi-scale dynamics. From such simulations, a model can be learned for the coefficients $\alpha_k^{(p)}$. Indeed, the coefficients $\alpha_k^{(p)}$ are learned (chosen) in order to model the coarse-grained dynamics most accurately. Figure 14.11 shows the architecture used to train such a model. The philosophy behind this parameterization can be carried over to time integration as well. Importantly, the integrator must be numerically stable and also generalize as best as possible. This is achieved in Bar-Sinai et al. [52] by leveraging a multi-layer neural network to parameterize the solution manifold. The multi-layer network's flexibility also allows one to impose physical constraints and interpretability through choice of model architecture. Details of the neural network and its implementation can

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

be found at `https://github.com/google/data-driven-discretization-1d`.

Figure 14.12 compares the integration results for a particular realization of the forcing of Burgers' equation for different values of the resampling factor, that is, the ratio between the number of grid points in the low-resolution calculation and that of the fully converged solution. The learned models, with both constant and solution-dependent coefficients, can propagate the solution in time and dramatically outperform the baseline method at low resolution. Importantly, the ringing effect around the shocks, which leads to numerical instabilities, is practically eliminated. Since the model is trained on fully resolved simulations, a crucial requirement for the method to be of practical use is that training can be done on small systems, but still produce models that perform well on larger ones. This is expected to be the case, since the models, being based on convolutional neural networks, use only local features and by construction are translation-invariant. Figure 14.12B illustrates the performance of the model trained on the domain $[0, 2\pi]$ for predictions on a 10-times larger spatial domain of size $[0, 20\pi]$. The learned model generalizes well. For example, it shows good performance when function values are all positive in a region of size greater than 2π, which, due to the conservation law, cannot occur in the training data set. Overall, the work of Bar-Sinai et al. [52] provides an elegant closure solution for the parameterization of fine-scale physics, something that is of high value for producing reasonable solution times for multi-scale physics systems.

An alternative deep learning approach uses a *multi-resolution convolutional autoencoder* (MrCAE) [450] architecture that integrates and leverages three highly successful mathematical architectures: (i) multi-grid methods, (ii) convolutional autoencoders, and (iii) transfer learning. The method provides an adaptive, hierarchical architecture that capitalizes on a progressive training approach for multi-scale spatio-temporal data. This framework allows for inputs across multiple scales: starting from a compact (small number of weights) network architecture and low-resolution data, this network progressively deepens and widens itself in a principled manner to encode new information in the higher-resolution data based on its current performance of reconstruction. Basic transfer learning techniques are applied to ensure information learned from previous training steps can be rapidly transferred to the larger network. As a result, the network can dynamically capture different scaled features at different depths of the network. For details, see `https://github.com/luckystarufo/MrCAE`.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

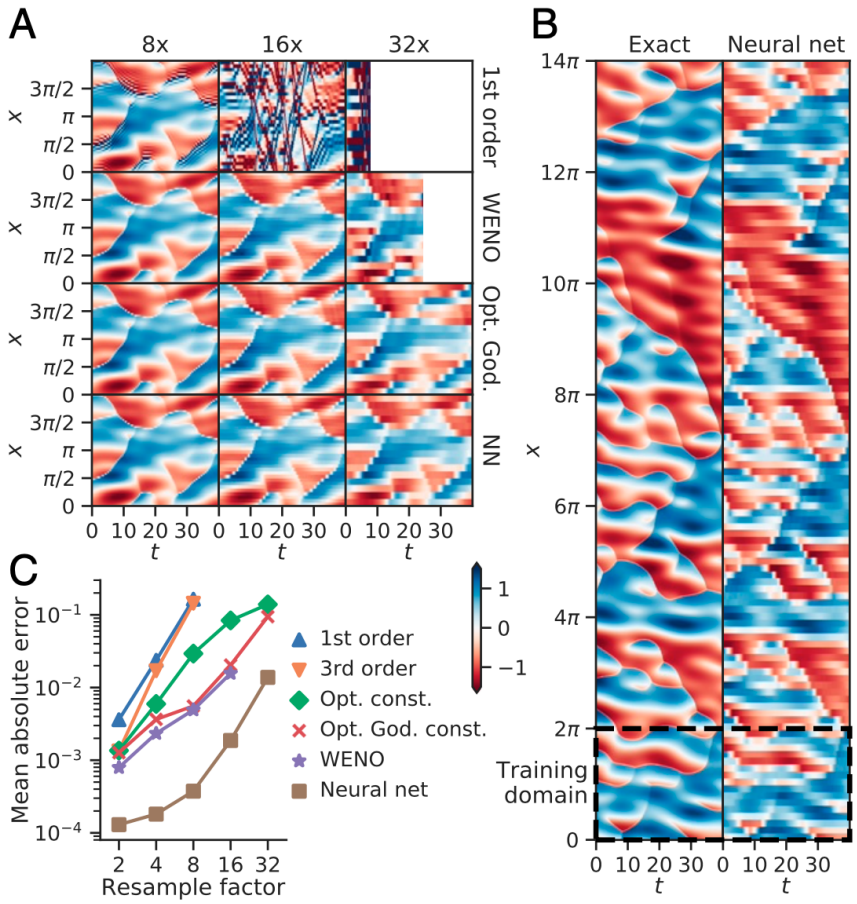


Figure 14.12: Time integration results for Burgers' equation. (A) A particular realization of a solution at varying resolution solved by the baseline first-order finite-volume method, weighted essentially non-oscillatory (WENO), optimized constant coefficients with Godunov flux (Opt. God.), and the neural network (NN), with the white region indicating times when the solution diverged. Both learned methods manifestly outperform the baseline method and even outperform WENO at coarse resolutions. (B) Inference predictions for the $32\times$ neural network model, on a 10 times larger spatial domain (only partially shown). The box surrounded by the dashed line shows the spatial domain used for training. (C) Mean absolute error between integrated solutions and the ground truth, averaged over space, times less than 15, and 10 forcing realizations on the 10-times larger inference domain. These metrics almost exactly match results on the smaller training domain $[0, 2\pi]$. As ground truth, we use WENO simulations on a $1\times$ grid. Markers are omitted if some simulations diverged or if the average error is worse than fixing $\mathbf{u} = 0$. From Bar-Sinai et al. [52].

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

14.7 Deep Learning and Boundary Value Problems

Thus far, boundary value problems (BVPs) have not been discussed. However, the differential and partial differential equations that represent BVPs are amenable to all the data-driven methods discussed thus far. As an example of how deep learning can be used in the context of BVPs, we consider the ubiquitous Green's function. The Green's function constructs the solution to a BVP for any given forcing by *linear* superposition. Specifically, consider the classical linear BVP [673]

$$L[v(\mathbf{x})] = f(\mathbf{x}), \tag{14.25}$$

where L is a linear differential operator, f is a forcing, $\mathbf{x} \in \Omega$ is the spatial coordinate, and Ω is an open set. The boundary conditions $\mathbf{B}(\mathbf{x}) = 0$ are imposed on $\partial\Omega$ with a linear operator $\mathbf{B}$. The fundamental solution is constructed by considering the adjoint equation

$$L^\dagger[G(\mathbf{x}, \boldsymbol{\xi})] = \delta(\mathbf{x} - \boldsymbol{\xi}), \tag{14.26}$$

where $L^\dagger$ is the adjoint operator (along with its associated boundary conditions) and $\delta(\mathbf{x}-\xi)$ is the Dirac delta function. Taking the inner product of (14.25) with respect to the Green's function gives the fundamental solution

$$v(\mathbf{x}) = (f(\boldsymbol{\xi}), G(\boldsymbol{\xi}, \mathbf{x})) = \int_\Omega G(\xi, \mathbf{x}) f(\boldsymbol{\xi})\, \mathrm{d}\boldsymbol{\xi}, \tag{14.27}$$

which is valid for any forcing $f(\mathbf{x})$. Thus, once the Green's function is computed, the solution for arbitrary forcing functions can be extracted from integration. This integration represents a superposition of a continuum of delta function forcings that are used to represent $f(\mathbf{x})$. The Green's function was a motivating example in the neural operator approach [407, 441, 442, 443] since it provides a kernel representation of the solution. Of course, the Green's function only works for *linear* problems, since superposition of solutions must hold.

Neural networks can, however, transform our view of Green's functions. Specifically, as already illustrated in previous sections, data-driven modeling can jointly learn coordinates and models. Thus, for BVPs, one can learn a coordinate transformation and kernel representation jointly, which would allow for the Green's function methodology. Thus we can turn nonlinear problems linear so as to exploit linear superposition. Indeed, in many modern applications, nonlinearity plays a fundamental role so that the BVP is of the form

$$N[u(\mathbf{x})] = F(\mathbf{x}), \tag{14.28}$$

where $N[\cdot]$ is a nonlinear differential operator. For this case, the principle of linear superposition no longer holds and the notion of a fundamental solution

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

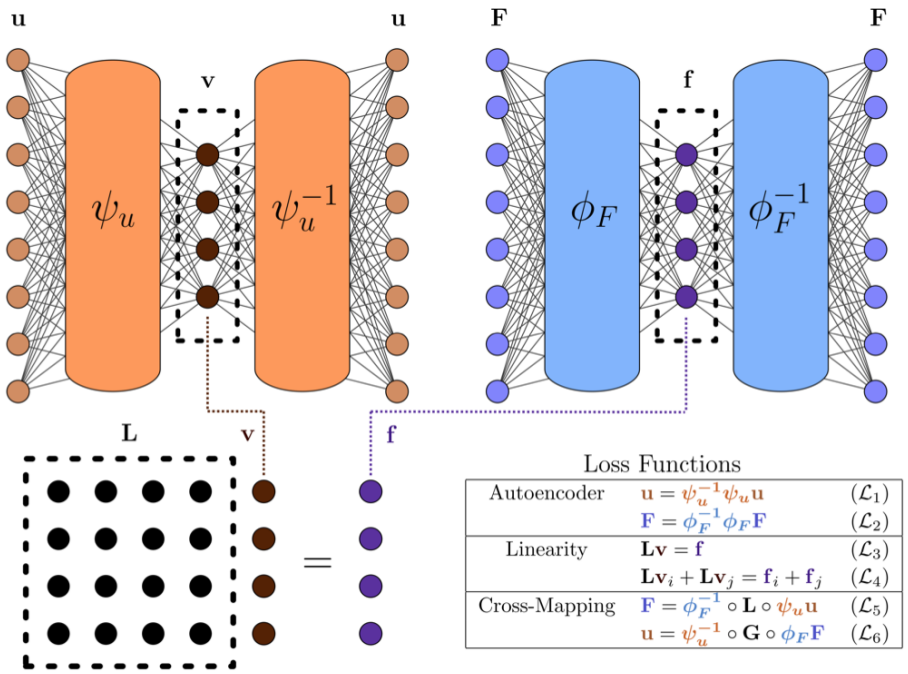


Figure 14.13: DeepGreen architecture. Two autoencoders learn invertible coordinate transformations that linearize a nonlinear boundary value problem. The latent space is constrained to exhibit properties of a linear system, including linear superposition, which enables discovery of a Green's function for nonlinear boundary value problems. From Gin et al. [280].

is lost. However, modern deep learning algorithms allow us the flexibility of learning coordinate transformations (and their inverses) of the form

$$v = \boldsymbol{\psi}(u), \tag{14.29a}$$
$$f = \boldsymbol{\phi}(F), \tag{14.29b}$$

such that v and f satisfy the linear BVP (14.25) for which we generated the fundamental solution (14.27). This gives a nonlinear fundamental solution through use of this deep learning transformation.

DeepGreen [280] leverages the success of DNNs for dynamical systems to discover coordinate transformations that linearize nonlinear BVPs so that the Green's function solution can be recovered. This allows for the discovery of the fundamental solutions for nonlinear BVPs, opening many opportunities for the engineering and physical sciences. DeepGreen exploits physics-informed learning by using autoenconders (AEs) to take data from the original high-dimensional input space to the new coordinates at the intrinsic rank of the underlying physics [168, 465, 544]. The architecture also leverages the success of *deep residual networks* (DRNs) [317], which enables our approach to efficiently

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

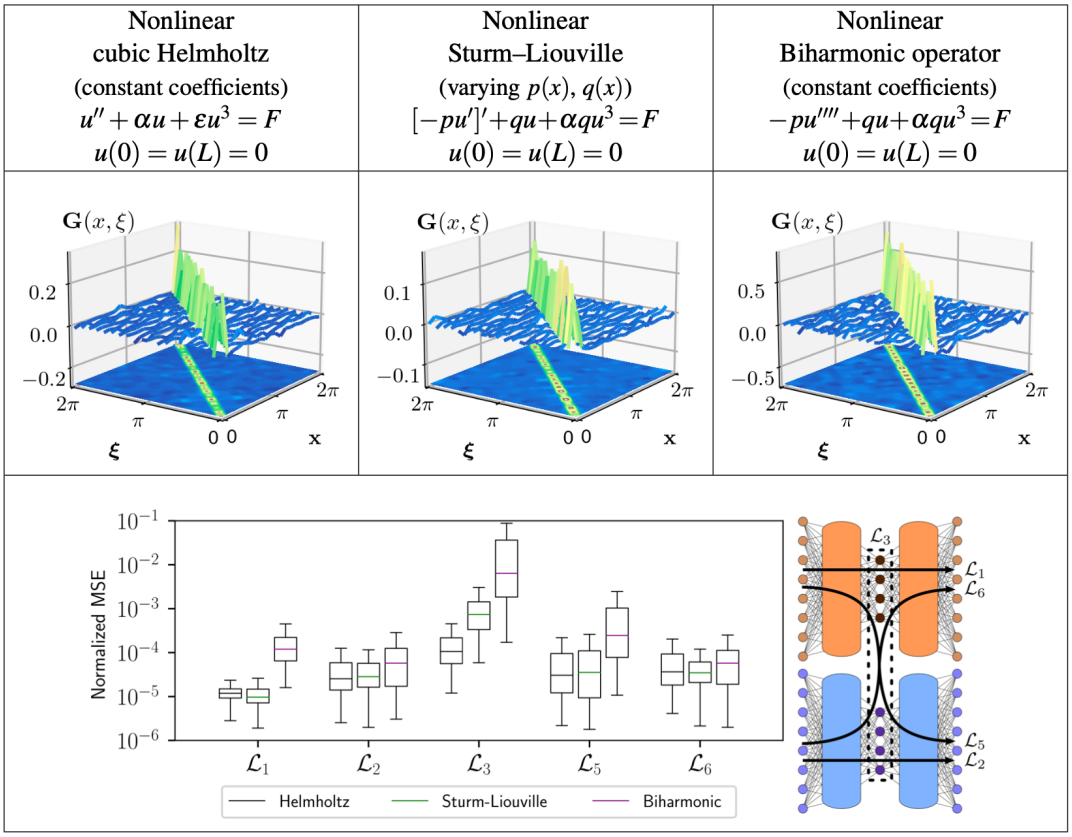


Figure 14.14: Summary of results for three one-dimensional models. The models are provided with the Green's function learned by DeepGreen. A summary box plot shows the relative losses for all three model systems. The loss functions are shown in Fig. 14.13 and are associated with the autoencoder, linearity, and cross-mapping. From Gin et al. [280].

handle near-identity coordinate transformations [279]. Figure 14.13 highlights the deep learning approach which leverages a dual autoencoder architecture. DeepGreen transforms a nonlinear BVP to a linear BVP, solves the linearized BVP, and then inverse-transforms the linear solution to solve the nonlinear BVP. Figure 14.14 highlights the nonlinear Green's functions found for a number of prototype nonlinear BVPs. The success of the algorithm again shows how multiple neural networks can be simultaneously trained to produce high-quality characterizations of physics-based problems.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Suggested Reading and Homework

Exercise 14-1. Read and reproduce the results of Champion et al. [168]:
`https://github.com/kpchamp/SindyAutoencoders`

Exercise 14-2. Read and reproduce the results of Lange et al. [428]:
`https://github.com/helange23/from_fourier_to_koopman`

Exercise 14-3. Read and reproduce the results of Lu et al. [461]:
`https://github.com/lululxvi/deepxde`

Exercise 14-4. Read and reproduce the results of Kovachki et al. [407]:
`https://github.com/zongyi-li/graph-pde`
`https://github.com/zongyi-li/fourier_neural_operator`

Exercise 14-5. Read and reproduce the results of Raissa et al. [584]:
`https://github.com/maziarraissi/PINNs`

Exercise 14-6. Read and reproduce the results of Bar-Sinai et al. [52]:
`https://github.com/google/data-driven-discretization-1d`

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Glossary

Actor–critic – A reinforcement learning algorithm that simultaneously learns a policy function and a value function, with the goal of taking the best from both.

Adjoint – For a finite-dimensional linear map (i.e., a matrix $\mathbf{A}$), the adjoint $\mathbf{A}^*$ is given by the complex conjugate transpose of the matrix. In the infinite-dimensional context, the adjoint $\mathcal{A}^*$ of a linear operator $\mathcal{A}$ is defined so that $\langle \mathcal{A}f, g\rangle = \langle f, \mathcal{A}^* g\rangle$, where $\langle \cdot, \cdot \rangle$ is an inner product.

Agent – In reinforcement learning (RL), an agent senses the state $\mathbf{s}$ of its environment and learns to take appropriate actions $\mathbf{a}$ to achieve an optimal future reward $\mathbf{r}$.

Akaike information criterion (AIC) – An estimator of the relative quality of statistical models for a given set of data. Given a collection of models for the data, AIC estimates the quality of each model, relative to each of the other models. Thus, AIC provides a means for model selection.

Autoencoder – Autoencoders are a class of machine learning models that are used to learn efficient latent codings of unlabeled data (unsupervised learning). Autoencoders learn efficient codings by performing nonlinear dimensionality reduction. Autoencoders are typically trained with both an encoding layer and a decoding layer so that one can map to the latent representation and back.

Backpropagation (backprop) – A method used for computing the gradient descent required for the training of neural networks (NNs). Based upon the chain rule, backprop exploits the compositional nature of NNs in order to frame an optimization problem for updating the weights of the network. It is commonly used to train deep neural networks (DNNs).

Balanced input–output model – A model expressed in a coordinate system where the states are ordered hierarchically in terms of their joint controllability and observability. The controllability and observability Gramians are equal and diagonal for such a system.

Bayesian information criterion (BIC) – An estimator of the relative quality of statistical models for a given set of data. Given a collection of models for the data, BIC estimates the quality of each model, relative to each of the other models. Thus, BIC provides a means for model selection.

Bellman optimality – A cornerstone of dynamic programming, stating that an optimal multi-step sequence must also be locally optimal in every sub-sequence of steps.

Classification – A general process related to categorization, the process in which ideas and objects are recognized, differentiated, and understood. Classification is a common task for machine learning algorithms.

Closed-loop control – A control architecture where the actuation is informed by sensor data about the output of the system.

Clustering – A task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense) to each other than to those in other groups (clusters). It is a primary goal of exploratory data mining, and a common technique for statistical data analysis.

Coherent structure – A spatial mode that is correlated with the data from a system.

Compressed sensing – The process of reconstructing a high-dimensional vector signal from a random undersampling of the data using the fact that the high-dimensional signal is sparse in a known transform basis, such as the Fourier basis.

Compression – The process of reducing the size of a high-dimensional vector or array by approximating it as a sparse vector in a transformed basis. For example, MP3 and JPG compression use the Fourier basis or wavelet basis to compress audio or image signals.

Control theory – The framework for modifying a dynamical system to conform to desired engineering specification through sensing and actuation.

Controllability – A system is controllable if it is possible to steer the system to any state with actuation. Degrees of controllability are determined by the controllability Gramian.

Convex optimization – An algorithmic framework for minimizing convex func-

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

tions over convex sets.

Convolutional neural network (CNN) – A class of deep, feedforward neural networks that is especially amenable to analyzing natural images. The convolution is typically a spatial filter which synthesizes local (neighboring) spatial information.

Cross-validation – A model validation technique for assessing how the results of a statistical analysis will generalize to an independent (withheld) data set.

Data matrix – A matrix where each column vector is a snapshot of the state of a system at a particular instant in time. These snapshots may be *sequential* in time, or they may come from an ensemble of initial conditions or experiments.

Deep learning – A class of machine learning algorithms that typically uses deep convolutional neural networks (CNNs) for feature extraction and transformation. Deep learning can leverage supervised (e.g., classification) and/or unsupervised (e.g., pattern analysis) algorithms, learning multiple levels of representations that correspond to different levels of abstraction; the levels form a hierarchy of concepts.

Deep reinforcement learning – Reinforcement learning algorithms that leverage deep neural networks, such as deep policy networks and deep Q-learning.

DeepONet – DeepONets are a class of machine learning models for sequential data typically generated by a deterministic dynamical system. Specifically, a DeepONet learns the underlying operator associated with the dynamical system or PDE. DeepONets train two neural networks simultaneously: one for encoding the input function at a fixed number of sensors/measurement locations (branch net), and another for encoding the locations for the output function (trunk net).

DMD amplitude – The amplitude of a given DMD mode (see *Dynamic mode decomposition*) as expressed in the data. These amplitudes may be interpreted as the significance of a given DMD mode, similar to the power spectrum in the fast Fourier transform (FFT).

DMD eigenvalue – Eigenvalues of the best-fit DMD operator $\mathbf{A}$ (see *Dynamic mode decomposition*) representing an oscillation frequency and a growth or decay term.

DMD mode (also *dynamic mode*) – An eigenvector of the best-fit DMD opera-

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

tor $\mathbf{A}$ (see *Dynamic mode decomposition*). These modes are spatially coherent and oscillate in time at a fixed frequency and a growth or decay rate.

Dynamic mode decomposition (DMD) – The leading eigendecomposition of a best-fit linear operator $\mathbf{A} = \mathbf{X}'\mathbf{X}^\dagger$ that propagates the data matrix $\mathbf{X}$ into a future data matrix $\mathbf{X}'$. The eigenvectors of $\mathbf{A}$ are DMD modes and the corresponding eigenvalues determine the time dynamics of these modes.

Dynamic programming – A powerful optimization approach used extensively for optimal nonlinear control and reinforcement learning. Dynamic programming reformulates large multi-step optimization problems into a recursive optimization of smaller sub-problems, relying on Bellman's principle of optimality.

Dynamical system – A mathematical model for the dynamic evolution of a system. Typically, a dynamical system is formulated in terms of ordinary differential equations (ODEs) on a state space. The resulting equations may be linear or nonlinear and may also include the effect of actuation inputs and represent outputs as sensor measurements of the state.

Eigensystem realization algorithm (ERA) – A system identification technique that produces balanced input–output models of a system from impulse-response data. ERA has been shown to produce equivalent models to balanced proper orthogonal decomposition (BPOD) and dynamic mode decomposition (DMD) under some circumstances.

Emission – The measurement functions for a hidden Markov model.

Environment – The external system or world in which a reinforcement learning agent takes actions to interact with. Often, the environment is a Markov decision process.

Fast Fourier transform (FFT) – A numerical algorithm to compute the discrete Fourier transform (DFT) in $\mathcal{O}(n \log(n))$ operations. The FFT has revolutionized modern computations, signal processing, compression, and data transmission.

Feedback control – Closed-loop control where sensors measure the downstream effect of actuators, so that information is fed back to the actuators. Feedback is essential for robust control where model uncertainty and instability may be counteracted with fast sensor feedback.

Feedforward control – Control where sensors measure the upstream distur-

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

bances to a system, so that information is fed forward to actuators to cancel disturbances proactively.

Fourier transform – A change of basis used to represent a function in terms of an infinite series of sines and cosines.

Galerkin projection – A process by which governing partial differential equations (PDEs) are reduced into ordinary differential equations (ODEs) in terms of the dynamics of the coefficients of a set of orthogonal basis modes that are used to approximate the solution.

Gated recurrent unit (GRU) – GRUs are a class of machine learning models for sequential data. GRUs are a subset of RNNs and LSTMs, but with a forget gate and with fewer parameters than an LSTM, since a GRU does not have an output gate.

Generative adversarial network (GAN) – GANs are a class of machine learning models that learn to generate new data with the same statistics as the training set. GANs include a generative network that learns to map from a latent space to a data distribution of interest, while a second discriminator network classifies data candidates produced by the generator from the true data distribution. The generative network's training objective is to increase the error rate of the discriminative network. Thus the generator can fool the discriminator network by producing novel candidates that the discriminator thinks are not synthesized data.

Gramian – The controllability (respectively, observability) Gramian determines the degree to which a state is controllable (respectively, observable) via actuation (respectively, estimation). The Gramian establishes an inner product on the state space.

Hidden Markov model (HMM) – A Markov model where there is a hidden state that is only observed through a set of measurements known as emissions.

Hilbert space – A generalized vector space with an inner product. When referred to in this text, a Hilbert space typically refers to an infinite-dimensional function space. These spaces are also complete metric spaces, providing a sufficient mathematical framework to enable calculus on functions.

Hindsight experience replay – The process of learning from past experiences in off-policy reinforcement learning algorithms, such as Q-learning.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Imitation learning – The process of learning from other more experienced agents in off-policy reinforcement learning algorithms, such as Q-learning.

Incoherent measurements – Measurements that have a small inner product with the basis vectors of a sparsifying transform. For instance, single-pixel measurements (i.e., spatial delta functions) are incoherent with respect to the spatial Fourier transform basis, since these single-pixel measurements excite all frequencies and do not preferentially align with any single frequency.

Kalman filter – An estimator that reconstructs the full state of a dynamical system from measurements of a time series of the sensor outputs and actuation inputs. A Kalman filter is itself a dynamical system that is constructed for observable systems to stably converge to the true state of the system. The Kalman filter is optimal for linear systems with Gaussian process and measurement noise of a known magnitude.

Koopman eigenfunction – An eigenfunction of the Koopman operator. These eigenfunctions correspond to measurements on the state space of a dynamical system that form intrinsic coordinates. In other words, these intrinsic measurements will evolve linearly in time despite the underlying system being nonlinear.

Koopman operator – An infinite-dimensional linear operator that propagates measurement functions from an infinite-dimensional Hilbert space through a dynamical system.

Laplace transform – A generalization of the Fourier transform for a larger class of functions that are not Lebesgue-integrable, such as exponential functions. The Laplace transform may be thought of as a weighted, one-sided Fourier transform for badly behaved functions.

Least-squares regression – A regression technique where a best-fit line or vector is found by minimizing the sum of squares of the error between the model and the data.

Linear–quadratic regulator (LQR) – An optimal proportional feedback controller for full-state feedback, which balances the objectives of regulating the state while not expending too much control energy. The proportional gain matrix is determined by solving an algebraic Riccati equation.

Linear system – A system where superposition of any two inputs results in the superposition of the two corresponding outputs. In other words, doubling the

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

input doubles the output. Linear time-invariant dynamical systems are characterized by linear operators, which are represented as matrices.

Long short-term memory (LSTM) – LSTMs are a class of machine learning models for sequential data. LSTMs are a subset of RNNs, with specific filtering functions for improving sequential (temporal) modeling. A common LSTM unit is composed of a cell, an input gate, an output gate, and a forget gate. The cell remembers values over arbitrary time intervals and the three gates regulate the flow of information into and out of the cell.

Low rank – A property of a matrix where the number of linearly independent rows and columns is small compared with the size of the matrix. Generally, low-rank approximations are sought for large data matrices.

Machine learning – A set of statistical tools and algorithms that are capable of extracting the dominant patterns in data. The data mining can be supervised or unsupervised, with the goal of clustering, classification and prediction.

Markov decision process (MDP) – A common environment in reinforcement learning, in which the probability of the system being in the next state is determined entirely by the current state and action.

Markov model – A probabilistic dynamical system where the state vector contains the probability that the system will be in a given state; thus, this state vector must always sum to unity. The dynamics are given by the Markov transition matrix, which is constructed so that each row sums to unity.

Markov parameters – The output measurements of a dynamical system in response to an impulsive input.

Max pooling – A data downsampling strategy whereby an input representation (image, hidden-layer output matrix, etc.) is reduced in dimensionality, thus allowing for assumptions to be made about features contained in the downsampled sub-regions.

Model predictive control (MPC) – A form of optimal control that optimizes a control policy over a finite-time horizon, based on a model. The models used for MPC are typically linear and may be determined empirically via system identification.

Moore's law – The observation that transistor density, and hence processor speed, increases exponentially in time. Moore's law is commonly used to pre-

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

dict future computational power and the associated increase in the scale of problem that will be computationally feasible.

Multi-scale – The property of having many scales in space and/or time. Many systems, such as turbulence, exhibit spatial and temporal scales that vary across many orders of magnitude.

Neural operator – Neural operators are a class of machine learning models for sequential data typically generated by a deterministic dynamical system. Like DeepONet, neural networks are tailored to learn operators by mapping between infinite-dimensional function spaces.

Observability – A system is observable if it is possible to estimate any system state with a time history of the available sensors. Degrees of observability are determined by the observability Gramian.

Observable function – A function that measures some property of the state of a system. Observable functions are typically elements of a Hilbert space.

Off-policy reinforcement learning – Reinforcement learning algorithms, such as Q-learning, where the agent is able to take sub-optimal actions while learning, enabling imitation learning and hindsight replay.

On-policy reinforcement learning – Reinforcement learning algorithms where the agent must take the best action according to its current policy as it learns.

Optimization – Generally a set of algorithms that find the "best available" values of some objective function given a defined domain (or input), including a variety of different types of objective functions and different types of domains. Mathematically, optimization aims to maximize or minimize a real function by systematically choosing input values from within an allowed set and computing the value of the function. The generalization of optimization theory and techniques to other formulations constitutes a large area of applied mathematics.

Over-determined system – A system $\mathbf{A}\mathbf{x} = \mathbf{b}$ where there are more equations than unknowns. Usually, there is no exact solution $\mathbf{x}$ to an over-determined system, unless the vector $\mathbf{b}$ is in the column space of $\mathbf{A}$.

Pareto front – The allocation of resources from which it is impossible to reallocate so as to make any one individual or preference criterion better off without making at least one individual or preference criterion worse off.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Perron–Frobenius operator – The adjoint of the Koopman operator, the Perron–Frobenius operator is an infinite-dimensional operator that advances probability density functions (PDFs) through a dynamical system.

Physics-informed neural network (PINN) – PINNs are a class of machine learning models for sequential data typically generated by a deterministic dynamical system. PINNs enforce governing equations, initial conditions, and boundary conditions in the training loss function. Thus the neural network is trained to solve a supervised learning task while respecting any given laws of physics described by general nonlinear partial differential equations.

Policy iteration – A form of dynamic programming in which the policy function and value function are iteratively updated, while the other is held fixed.

Policy function – A set of rules about what action an agent should take given the current state of the environment.

Power spectrum – The squared magnitude of each coefficient of a Fourier transform of a signal. The power corresponds to the amount of each frequency required to reconstruct a given signal.

Principal component – A spatially correlated mode in a given data set, often computed using the singular value decomposition (SVD) of the data after the mean has been subtracted.

Principal component analysis (PCA) – A decomposition of a data matrix into a hierarchy of principal component vectors that are ordered from most correlated to least correlated with the data. PCA is computed by taking the singular value decomposition (SVD) of the data after subtracting the mean. In this case, each singular value represents the variance of the corresponding principal component (singular vector) in the data.

Proper orthogonal decomposition (POD) – The decomposition of data from a dynamical system into a hierarchical set of orthogonal modes, often using the singular value decomposition (SVD). When the data consists of velocity measurements of a system, such as an incompressible fluid, then the POD orders modes in terms of the amount of energy these modes contain in the given data.

Pseudo-inverse – The pseudo-inverse generalizes the matrix inverse for non-square matrices, and is often used to compute the least-squares solution to a system of equations. The singular value decomposition (SVD) is a common

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

method to compute the pseudo-inverse: given the SVD $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^*$, the pseudo-inverse is $\mathbf{X}^\dagger = \mathbf{V}\mathbf{\Sigma}^{-1}\mathbf{U}^*$.

***Q* learning** – A leading model-free reinforcement learning algorithm based on the quality function $Q(\mathbf{s}, \mathbf{a})$.

Quality function – The joint quality of being in a particular state $\mathbf{s}$ and taking a given action $\mathbf{a}$. The quality function $Q(\mathbf{s}, \mathbf{a})$ extends the value function, bypassing the need to know the next optimal state $\mathbf{s}'$ and providing the basis for Q-learning.

Recurrent neural network (RNN) – RNNs are a class of machine learning models for sequential data, typically a temporal sequence, where connections between nodes form a directed or undirected graph along the sequence. Although similar to feedforward neural networks, RNNs use their internal state (memory) to process variable-length sequences of inputs.

Reduced-order model (ROM) – A model of a high-dimensional system in terms of a low-dimensional state. Typically, a reduced-order model balances accuracy with computational cost of the model.

Regression – A statistical model that represents an outcome variable in terms of indicator variables. Least-squares regression is a linear regression that finds the line of best fit to data; when generalized to higher dimensions and multi-linear regression, this generalizes to principal components regression. Nonlinear regression, dynamic regression, and functional or semantic regression are used in system identification, model reduction, and machine learning.

Reinforcement learning (RL) – A major branch of machine learning that is concerned with how to learn control laws and policies to interact with a complex environment from experience.

Restricted isometry property (RIP) – The property that a matrix acts like a unitary matrix, or an isometry map, on sparse vectors. In other words, the distance between any two sparse vectors is preserved if these vectors are mapped through a matrix that satisfies the restricted isometry property.

Reward – A positive reinforcement signal in reinforcement learning (RL), to be maximized by an agent's policy $\boldsymbol{\pi}(\mathbf{s}, \mathbf{a})$.

Reward shaping – The process of constructing a customized proxy reward signal that may be used to improve the learning rate for systems with sparse re-

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

wards.

Robust control – A field of control that penalizes worst-case scenario control outcomes, thus promoting controllers that are robust to uncertainties, disturbances, and unmodeled dynamics.

Robust statistics – Methods for producing good statistical estimates for data drawn from a wide range of probability distributions, especially for distributions that are not normal and where outliers compromise predictive capabilities.

SARSA – SARSA (state–action–reward–state–action) learning is a form of on-policy temporal difference learning closely related to Q-learning.

Singular value decomposition (SVD) – Given a matrix $\mathbf{X} \in \mathbb{C}^{n \times m}$, the SVD is given by $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^*$, where $\mathbf{U} \in \mathbb{C}^{n \times n}$, $\mathbf{\Sigma} \in \mathbb{C}^{n \times m}$, and $\mathbf{V} \in \mathbb{C}^{m \times m}$. The matrices $\mathbf{U}$ and $\mathbf{V}$ are unitary, so that $\mathbf{U}\mathbf{U}^* = \mathbf{U}^*\mathbf{U} = \mathbf{I}$ and $\mathbf{V}\mathbf{V}^* = \mathbf{V}^*\mathbf{V} = \mathbf{I}$. The matrix $\mathbf{\Sigma}$ has entries along the diagonal corresponding to the singular values that are ordered from largest to smallest. This produces a hierarchical matrix decomposition that splits a matrix into a sum of rank-one matrices given by the outer product of a column vector (left singular vector) with a row vector (conjugate transpose of a right singular vector). These rank-one matrices are ordered by the singular value, so that the first r rank-one matrices form the *best* rank-r matrix approximation of the original matrix in a least-squares sense.

Snapshot – A single high-dimensional measurement of a system at a particular time. A number of snapshots collected at a sequence of times may be arranged as column vectors in a data matrix.

Sparse identification of nonlinear dynamics (SINDy) – A nonlinear system identification framework used to simultaneously identify the nonlinear structure and parameters of a dynamical system from data. Various sparse optimization techniques may be used to determine SINDy models.

Sparsity – A vector is *sparse* if most of its entries are zero or nearly zero. Sparsity refers to the observation that most data are sparse when represented as vectors in an appropriate transformed basis, such as Fourier or proper orthogonal decomposition (POD) bases.

Spectrogram – A short-time Fourier transform computed on a moving window, which results in a time–frequency plot of which frequencies are active at a given time. The spectrogram is useful for characterizing non-periodic signals,

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

where the frequency content evolves over time, as in music.

State space – The set of all possible system states. Often the state space is a vector space, such as $\mathbb{R}^n$, although it may also be a smooth manifold $\mathcal{M}$.

Stochastic gradient descent – Also known as incremental gradient descent, it allows one to approximate the gradient with a single data point instead of all available data. At each step of the gradient descent, a randomly chosen data point is used to compute the gradient direction.

System identification – The process by which a model is constructed for a system from measurement data, possibly after perturbing the system.

Temporal difference error – The difference between the estimated future reward (i.e., the target) and the actual future reward, which is used to update the value or quality function in TD learning.

Temporal difference (TD) learning – A sample-based reinforcement learning strategy, in which the current value or quality function is updated based on the rewards obtained in the subsequent events. TD learning is designed to mimic the learning process in animals.

Temporal difference target – The estimated future reward in TD learning.

Time delay coordinates – An augmented set of coordinates constructed by considering a measurement at the current time along with a number of times in the past at fixed intervals from the current time. Time delay coordinates are often useful in reconstructing attractor dynamics for systems that do not have enough measurements, as in the Takens embedding theorem.

Total least-squares – A least-squares regression algorithm that minimizes the error on both the inputs and the outputs. Geometrically, this corresponds to finding the line that minimizes the sum of squares of the total distance to all points, rather than the sum of squares of the vertical distance to all points.

Uncertainty quantification (UQ) – The principled characterization and management of uncertainty in engineering systems. Uncertainty quantification often involves the application of powerful tools from probability and statistics to dynamical systems.

Under-determined system – A system $\mathbf{A}\mathbf{x} = \mathbf{b}$ where there are fewer equations than unknowns. Generally, the system has infinitely many solutions $\mathbf{x}$ unless $\mathbf{b}$

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

is not in the column space of $\mathbf{A}$.

Unitary matrix – A matrix whose complex conjugate transpose is also its inverse. All eigenvalues of a unitary matrix are on the complex unit circle, and the action of a unitary matrix may be thought of as a change of coordinates that preserves the Euclidean distance between any two vectors.

Value function – A function quantifying the desirability of being in a given state $\mathbf{s}$, as calculated by the discounted sum of future rewards, given an optimal policy starting from this state. The value function is often written in a recursive form, based on Bellman's equation.

Value iteration – A form of dynamic programming that iteratively updates the value function, after which an optimal policy may be extracted.

Wavelet – A generalized function, or family of functions, used to generalize the Fourier transform to approximate more complex and multi-scale signals.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

References

[1] P. Abbeel, A. Coates, and A. Y. Ng. Autonomous helicopter aerobatics through apprenticeship learning. *International Journal of Robotics Research*, 29(13):1608–1639, 2010.

[2] P. Abbeel, A. Coates, M. Quigley, and A. Y. Ng. An application of reinforcement learning to aerobatic helicopter flight. In *Advances in Neural Information Processing Systems, 19*, 8pp., 2007.

[3] R. Abraham and J. E. Marsden. *Foundations of Mechanics*, 2nd edition, volume 36 of Benjamin/Cummings Advanced Book Program. Benjamin/Cummings, 1978.

[4] R. Abraham, J. E. Marsden, and T. Ratiu. *Manifolds, Tensor Analysis, and Applications*, volume 75 of Applied Mathematical Sciences. Springer, 1988.

[5] M. Agrawal, S. Vidyashankar, and K. Huang. On-chip implementation of ECoG signal data decoding in brain–computer interface. In *2016 IEEE 21st International Mixed-Signal Testing Workshop*, pages 1–6. IEEE, 2016.

[6] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases*, pages 487–499. ACM, 1994.

[7] H.-S. Ahn, Y. Chen, and K. L. Moore. Iterative learning control: Brief survey and categorization. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 37(6):1099–1121, 2007.

[8] H. Akaike. Fitting autoregressive models for prediction. *Annals of the Institute of Statistical Mathematics*, 21(1):243–247, 1969.

[9] H. Akaike. A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, 19(6):716–723, 1974.

[10] A. Alla and J. N. Kutz. Nonlinear model order reduction via dynamic mode decomposition. *SIAM Journal on Scientific Computing*, 39(5):B778–B796, 2017.

[11] A. Alla and J. N. Kutz. Randomized model order reduction. *Advances in Computational Mathematics*, 45(3):1251–1271, 2019.

[12] E. P. Alves and F. Fiuza. Data-driven discovery of reduced plasma physics models from fully-kinetic simulations. Preprint arXiv:2011.01927, 2020.

[13] B. Amos, I. D. J. Rodriguez, J. Sacks, B. Boots, and J. Z. Kolter. Differentiable MPC for end-to-end planning and control. In *Advances in Neural Information Processing Systems, 31*, pages 8299–8310, 2018.

[14] W. Amrein and A.-M. Berthier. On support properties of L^p-functions and their Fourier transforms. *Journal of Functional Analysis*, 24(3):258–267, 1977.

[15] D. Amsallem and C. Farhat. An online method for interpolating linear parametric reduced-order models. *SIAM Journal on Scientific Computing*, 33(5):2169–2198, 2011.

[16] D. Amsallem, J. Cortial, and C. Farhat. On-demand CFD-based aeroelastic predictions using a database of reduced-order bases and models. In *47th AIAA Aerospace Sciences Meeting Including The New Horizons Forum and Aerospace Exposition*, page 800, 2009.

[17] D. Amsallem, M. J. Zahr, and K. Washabaugh. Fast local reduced basis updates for the efficient reduction of nonlinear systems with hyper-reduction. *Advances in Computational Mathematics*, 41(5):1187–1230, 2015.

[18] J. Andén and S. Mallat. Deep scattering spectrum. *IEEE Transactions on Signal Processing*, 62(16):4114–4128, 2014.

[19] E. Anderson, Z. Bai, C. Bischof, et al. *LAPACK Users' Guide*, volume 9. SIAM, 1999.

[20] J. L. Anderson. An ensemble adjustment Kalman filter for data assimilation. *Monthly Weather Review*, 129(12):2884–2903, 2001.

[21] C. A. Andersson and R. Bro. The n-way toolbox for Matlab. *Chemometrics and Intelligent Laboratory Systems*, 52(1):1–4, 2000.

[22] M. Andrychowicz, F. Wolski, A. Ray, et al. Hindsight experience replay. Preprint arXiv:1707.01495, 2017.

[23] M. Antonini, M. Barlaud, P. Mathieu, and I. Daubechies. Image coding using wavelet transform. *IEEE Transactions on Image Processing*, 1(2):205–220, 1992.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

[24] A. C. Antoulas. *Approximation of Large-Scale Dynamical Systems*. SIAM, 2005.

[25] H. Arbabi and I. Mezić. Ergodic theory, dynamic mode decomposition and computation of spectral properties of the Koopman operator. *SIAM Journal on Applied Dynamical Systems*, 16(4):2096–2126, 2017.

[26] K. B. Ariyur and M. Krstić. *Real-Time Optimization by Extremum-Seeking Control*. Wiley-Interscience, 2003.

[27] T. Askham and J. N. Kutz. Variable projection methods for an optimized dynamic mode decomposition. *SIAM Journal on Applied Dynamical Systems*, 17(1):380–416, 2018.

[28] T. Askham, P. Zheng, A. Aravkin, and J. N. Kutz. Robust and scalable methods for the dynamic mode decomposition. Preprint arXiv:1712.01883, 2017.

[29] P. Astrid. Fast reduced order modeling technique for large scale LTV systems. In *Proceedings of the 2004 American Control Conference*, volume 1, pages 762–767. IEEE, 2004.

[30] K. J. Aström and R. M. Murray. *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton University Press, 2010.

[31] M. Azeez and A. Vakakis. Proper orthogonal decomposition (POD) of a class of vibroimpact oscillations. *Journal of Sound and Vibration*, 240(5):859–889, 2001.

[32] O. Azencot, W. Yin, and A. Bertozzi. Consistent dynamic mode decomposition. *SIAM Journal on Applied Dynamical Systems*, 18(3):1565–1585, 2019.

[33] K. Bache and M. Lichman. UCI Machine Learning Repository, 2013.

[34] P. J. Baddoo, B. Herrmann, B. J. McKeon, and S. L. Brunton. Kernel learning for robust dynamic mode decomposition: Linear and nonlinear disambiguation optimization (LANDO). Preprint arXiv:2106.01510, 2021.

[35] B. W. Bader and T. G. Kolda. Efficient MATLAB computations with sparse and factored tensors. *SIAM Journal on Scientific Computing*, 30(1):205–231, 2007.

[36] S. Bagheri. Koopman-mode decomposition of the cylinder wake. *Journal of Fluid Mechanics*, 726:596–623, 2013.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

[37] S. Bagheri. Effects of weak noise on oscillating flows: Linking quality factor, Floquet modes, and Koopman spectrum. *Physics of Fluids*, 26(9):094104, 2014.

[38] S. Bagheri, L. Brandt, and D. Henningson. Input–output analysis, model reduction and control of the flat-plate boundary layer. *Journal of Fluid Mechanics*, 620:263–298, 2009.

[39] S. Bagheri, J. Hoepffner, P. J. Schmid, and D. S. Henningson. Input–output analysis and control design applied to a linear model of spatially developing flows. *Applied Mechanics Reviews*, 62(2):020803 (27pp.), 2009.

[40] Z. Bai, S. L. Brunton, B. W. Brunton, et al. Data-driven methods in fluid dynamics: Sparse classification from experimental data. Invited chapter in *Whither Turbulence and Big Data in the 21st Century*, 2015.

[41] Z. Bai, E. Kaiser, J. L. Proctor, J. N. Kutz, and S. L. Brunton. Dynamic mode decomposition for compressive system identification. Preprint arXiv:1710.07737, 2017.

[42] Z. Bai, T. Wimalajeewa, Z. Berger, et al. Low-dimensional approach for reconstruction of airfoil data via compressive sensing. *AIAA Journal*, 53(4):920–933, 2014.

[43] O. Balabanov and A. Nouy. Randomized linear algebra for model reduction. Part I: Galerkin methods and error estimation. *Advances in Computational Mathematics*, 45(5):2969–3019, 2019.

[44] M. J. Balajewicz, E. H. Dowell, and B. R. Noack. Low-dimensional modelling of high-Reynolds-number shear flows incorporating constraints from the Navier–Stokes equation. *Journal of Fluid Mechanics*, 729:285–308, 2013.

[45] M. Balasubramanian, S. Zabic, C. Bowd, et al. A framework for detecting glaucomatous progression in the optic nerve head of an eye using proper orthogonal decomposition. *IEEE Transactions on Information Technology in Biomedicine*, 13(5):781–793, 2009.

[46] P. Baldi and K. Hornik. Neural networks and principal component analysis: Learning from examples without local minima. *Neural Networks*, 2(1):53–58, 1989.

[47] B. Bamieh and L. Giarré. Identification of linear parameter varying models. *International Journal of Robust and Nonlinear Control*, 12:841–853, 2002.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

[48] A. Banaszuk, K. B. Ariyur, M. Krstić, and C. A. Jacobson. An adaptive algorithm for control of combustion instability. *Automatica*, 40(11):1965–1972, 2004.

[49] A. Banaszuk, S. Narayanan, and Y. Zhang. Adaptive control of flow separation in a planar diffuser. AIAA Paper 2003-617, 2003.

[50] A. Banaszuk, Y. Zhang, and C. A. Jacobson. Adaptive control of combustion instability using extremum-seeking. In *Proceedings of the 2000 American Control Conference*, volume 1, pages 416–422. IEEE, 2000.

[51] S. Banks. Infinite-dimensional Carleman linearization, the Lie series and optimal control of non-linear partial differential equations. *International Journal of Systems Science*, 23(5):663–675, 1992.

[52] Y. Bar-Sinai, S. Hoyer, J. Hickey, and M. P. Brenner. Learning data-driven discretizations for partial differential equations. *Proceedings of the National Academy of Sciences, USA*, 116(31):15 344–15 349, 2019.

[53] R. G. Baraniuk. Compressive sensing. *IEEE Signal Processing Magazine*, 24(4):118–120, 2007.

[54] R. G. Baraniuk, V. Cevher, M. F. Duarte, and C. Hegde. Model-based compressive sensing. *IEEE Transactions on Information Theory*, 56(4):1982–2001, 2010.

[55] M. Barrault, Y. Maday, N. C. Nguyen, and A. T. Patera. An empirical interpolation method: Application to efficient reduced-basis discretization of partial differential equations. *Comptes Rendus Mathematique*, 339(9):667–672, 2004.

[56] J. Basley, L. R. Pastur, N. Delprat, and F. Lusseyran. Space–time aspects of a three-dimensional multi-modulated open cavity flow. *Physics of Fluids*, 25(6):064105, 2013.

[57] J. Basley, L. R. Pastur, F. Lusseyran, T. M. Faure, and N. Delprat. Experimental investigation of global structures in an incompressible cavity flow using time-resolved PIV. *Experiments in Fluids*, 50(4):905–918, 2011.

[58] T. Baumeister, S. L. Brunton, and J. N. Kutz. Deep learning and model predictive control for self-tuning mode-locked lasers. *Journal of the Optical Society of America B*, 35(3):617–626, 2018.

[59] W. Baur and V. Strassen. The complexity of partial derivatives. *Theoretical Computer Science*, 22(3):317–330, 1983.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

[60] P. W. Bearman. On vortex shedding from a circular cylinder in the critical Reynolds number regime. *Journal of Fluid Mechanics*, 37(3):577–585, 1969.

[61] J.-F. Beaudoin, O. Cadot, J.-L. Aider, and J.-E. Wesfreid. Bluff-body drag reduction by extremum-seeking control. *Journal of Fluids and Structures*, 22:973–978, 2006.

[62] J.-F. Beaudoin, O. Cadot, J.-L. Aider, and J.-E. Wesfreid. Drag reduction of a bluff body using adaptive control methods. *Physics of Fluids*, 18(8):085107, 2006.

[63] R. Becker, R. King, R. Petz, and W. Nitsche. Adaptive closed-loop control on a high-lift configuration using extremum seeking. *AIAA Journal*, 45(6):1382–92, 2007.

[64] S. Beetham and J. Capecelatro. Formulating turbulence closures using sparse regression with embedded form invariance. *Physical Review Fluids*, 5(8):084611, 2020.

[65] S. Beetham, R. O. Fox, and J. Capecelatro. Sparse identification of multiphase turbulence closures for coupled fluid–particle flows. *Journal of Fluid Mechanics*, 914, 2021.

[66] G. Beintema, A. Corbetta, L. Biferale, and F. Toschi. Controlling Rayleigh–Bénard convection via reinforcement learning. Preprint arXiv:2003.14358, 2020.

[67] P. N. Belhumeur, J. P. Hespanha, and D. J. Kriegman. Eigenfaces vs. Fisherfaces: Recognition using class specific linear projection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):711–720, 1997.

[68] G. Bellani. Experimental studies of complex flows through image-based techniques. Doctoral Thesis, KTH, Stockholm, 2011.

[69] R. Bellman. On the theory of dynamic programming. *Proceedings of the National Academy of Sciences, USA*, 38(8):716–719, 1952.

[70] R. Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966.

[71] B. A. Belson, J. H. Tu, and C. W. Rowley. Algorithm 945: modred – a parallelized model reduction library. *ACM Transactions on Mathematical Software*, 40(4):30, 2014.

[72] M. Benedicks. On Fourier transforms of functions supported on sets of finite Lebesgue measure. *Journal of Mathematical Analysis and Applications*, 106(1):180–183, 1985.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

[73] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. Greedy layer-wise training of deep networks. In *Advances in Neural Information Processing Systems, 20*, pages 153–160, 2007.

[74] P. Benner, A. Cohen, M. Ohlberger, and K. Willcox. *Model Reduction and Approximation: Theory and Algorithms*, volume 15. SIAM, 2017.

[75] P. Benner, S. Gugercin, and K. Willcox. A survey of projection-based model reduction methods for parametric dynamical systems. *SIAM Review*, 57(4):483–531, 2015.

[76] P. Benner, J.-R. Li, and T. Penzl. Numerical solution of large-scale Lyapunov equations, Riccati equations, and linear–quadratic optimal control problems. *Numerical Linear Algebra with Applications*, 15(9):755–777, 2008.

[77] P. Benner, V. Mehrmann, V. Sima, S. Van Huffel, and A. Varga. Slicot – a subroutine library in systems and control theory. In *Applied and Computational Control, Signals, and Circuits*, pages 499–539. Springer, 1999.

[78] E. Berger, M. Sastuba, D. Vogt, B. Jung, and H. B. Amor. Estimation of perturbations in robotic behavior using dynamic mode decomposition. *Journal of Advanced Robotics*, 29(5):331–343, 2015.

[79] G. Berkooz, P. Holmes, and J. Lumley. The proper orthogonal decomposition in the analysis of turbulent flows. *Annual Review of Fluid Mechanics*, 25:539–575, 1993.

[80] D. P. Bertsekas. Nonlinear programming. *Journal of the Operational Research Society*, 48(3):334–334, 1997.

[81] D. P. Bertsekas. *Constrained Optimization and Lagrange Multiplier Methods*. Academic Press, 2014.

[82] D. P. Bertsekas and J. N. Tsitsiklis. Neuro-dynamic programming: an overview. In *Proceedings of 1995 34th IEEE Conference on Decision and Control*, volume 1, pages 560–564. IEEE, 1995.

[83] G. Beylkin, R. Coifman, and V. Rokhlin. Fast wavelet transforms and numerical algorithms I. *Communications on Pure and Applied Mathematics*, 44(2):141–183, 1991.

[84] K. Bieker, S. Peitz, S. L. Brunton, J. N. Kutz, and M. Dellnitz. Deep model predictive flow control with limited sensor data and online learning. *Theoretical and Computational Fluid Dynamics*, 34:577–591, 2020.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

[85] L. Biferale, F. Bonaccorso, M. Buzzicotti, P. Clark Di Leoni, and K. Gustavsson. Zermelo's problem: Optimal point-to-point navigation in 2D turbulent flows using reinforcement learning. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 29(10):103138, 2019.

[86] S. A. Billings. *Nonlinear System Identification: NARMAX Methods in the Time, Frequency, and Spatio-temporal Domains*. John Wiley & Sons, 2013.

[87] P. Binetti, K. B. Ariyur, M. Krstić, and F. Bernelli. Formation flight optimization using extremum seeking feedback. *Journal of Guidance, Control, and Dynamics*, 26(1):132–142, 2003.

[88] G. D. Birkhoff. Proof of the ergodic theorem. *Proceedings of the National Academy of Sciences, USA*, 17(12):656–660, 1931.

[89] G. D. Birkhoff and B. O. Koopman. Recent contributions to the ergodic theory. *Proceedings of the National Academy of Sciences, USA*, 18(3):279–282, 1932.

[90] C. M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

[91] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[92] D. A. Bistrian and I. M. Navon. An improved algorithm for the shallow water equations model reduction: Dynamic mode decomposition vs POD. *International Journal for Numerical Methods in Fluids*, 78(9):552–580, 2015.

[93] D. A. Bistrian and I. M. Navon. Randomized dynamic mode decomposition for nonintrusive reduced order modelling. *International Journal for Numerical Methods in Engineering*, 112(1):3–25, 2017.

[94] P. Bondi, G. Casalino, and L. Gambardella. On the iterative learning control theory for robotic manipulators. *IEEE Journal on Robotics and Automation*, 4(1):14–22, 1988.

[95] J. Bongard and H. Lipson. Automated reverse engineering of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences, USA*, 104(24):9943–9948, 2007.

[96] L. Boninsegna, F. Nüske, and C. Clementi. Sparse learning of stochastic dynamical equations. *Journal of Chemical Physics*, 148(24):241723, 2018.

[97] J. L. Borges. The library of Babel. *Collected Fictions*. Viking, 1998.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

[98] B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, pages 144–152. ACM, 1992.

[99] H. Boulard and Y. Kamp. Autoassociative memory by multilayer perceptron and singular values decomposition. *Biological Cybernetics*, 59:291–294, 1989.

[100] G. E. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung. *Time Series Analysis: Forecasting and Control*. John Wiley & Sons, 2015.

[101] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2009.

[102] S. Boyd, L. O. Chua, and C. A. Desoer. Analytical foundations of volterra series. *IMA Journal of Mathematical Control and Information*, 1(3):243–282, 1984.

[103] S. J. Bradtke and A. G. Barto. Linear least-squares algorithms for temporal difference learning. *Machine Learning*, 22(1):33–57, 1996.

[104] J. J. Bramburger and J. N. Kutz. Poincaré maps for multiscale physics discovery and nonlinear Floquet theory. *Physica D: Nonlinear Phenomena*, 408:132479, 2020.

[105] J. J. Bramburger, J. N. Kutz, and S. L. Brunton. Data-driven stabilization of periodic orbits. *IEEE Access*, 9:43 504–43 521, 2021.

[106] A. I. Bratcu, I. Munteanu, S. Bacha, and B. Raison. Maximum power point tracking of grid-connected photovoltaic arrays by using extremum seeking control. *Journal of Control Engineering and Applied Informatics*, 10(4):3–12, 2008.

[107] L. Breiman. Better subset regression using the nonnegative garrote. *Technometrics*, 37(4):373–384, 1995.

[108] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[109] L. Breiman. Statistical modeling: The two cultures (with comments and a rejoinder by the author). *Statistical Science*, 16(3):199–231, 2001.

[110] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen. *Classification and Regression Trees*. CRC Press, 1984.

[111] M. Brenner, J. Eldredge, and J. Freund. Perspective on machine learning for advancing fluid mechanics. *Physical Review Fluids*, 4(10):100501, 2019.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

[112] I. Bright, G. Lin, and J. N. Kutz. Compressive sensing and machine learning strategies for characterizing the flow around a cylinder with limited pressure measurements. *Physics of Fluids*, 25(12):127102 (15pp.), 2013.

[113] I. Bright, G. Lin, and J. N. Kutz. Classification of spatio-temporal data via asynchronous sparse sampling: Application to flow around a cylinder. *SIAM Multiscale Modeling and Simulation*, 14(2):823–838, 2016.

[114] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, 1998.

[115] D. Bristow, M. Tharayil, and A. G. Alleyne. A survey of iterative learning control. *IEEE Control Systems Magazine*, 26(3):96–114, 2006.

[116] R. Bro. PARAFAC. Tutorial and applications. *Chemometrics and Intelligent Laboratory Systems*, 38(2):149–171, 1997.

[117] A. Broad, T. Murphey, and B. Argall. Learning models for shared control of human–machine systems with unknown dynamics. *Robotics: Science and Systems Proceedings*, 2017.

[118] R. W. Brockett. Volterra series and geometric control theory. *Automatica*, 12(2):167–176, 1976.

[119] G. Brockman, V. Cheung, L. Pettersson, et al. OpenAI gym. Preprint arXiv:1606.01540, 2016.

[120] D. Broomhead and R. Jones. Time-series analysis. *Proceedings of the Royal Society A*, 423:103–121, 1989.

[121] D. S. Broomhead and D. Lowe. Radial basis functions, multi-variable functional interpolation and adaptive networks. Technical Report, Royal Signals and Radar Establishment, Malvern, UK, 1988.

[122] B. W. Brunton, S. L. Brunton, J. L. Proctor, and J. N. Kutz. Sparse sensor placement optimization for classification. *SIAM Journal on Applied Mathematics*, 76(5):2099–2122, 2016.

[123] B. W. Brunton, L. A. Johnson, J. G. Ojemann, and J. N. Kutz. Extracting spatial–temporal coherent patterns in large-scale neural recordings using dynamic mode decomposition. *Journal of Neuroscience Methods*, 258:1–15, 2016.

[124] S. L. Brunton and B. R. Noack. Closed-loop turbulence control: Progress and challenges. *Applied Mechanics Reviews*, 67:050801 (48pp.), 2015.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

[125] S. L. Brunton and C. W. Rowley. Maximum power point tracking for photovoltaic optimization using ripple-based extremum seeking control. *IEEE Transactions on Power Electronics*, 25(10):2531–2540, 2010.

[126] S. L. Brunton, B. W. Brunton, J. L. Proctor, E. Kaiser, and J. N. Kutz. Chaos as an intermittently forced linear system. *Nature Communications*, 8(19):1–9, 2017.

[127] S. L. Brunton, B. W. Brunton, J. L. Proctor, and J. N. Kutz. Koopman invariant subspaces and finite linear representations of nonlinear dynamical systems for control. *PLoS ONE*, 11(2):e0150171, 2016.

[128] S. L. Brunton, M. Budišić, E. Kaiser, and J. N. Kutz. Modern Koopman theory for dynamical systems. Preprint arXiv:2102.12086, 2021 (to appear in *SIAM Review*).

[129] S. L. Brunton, X. Fu, and J. N. Kutz. Extremum-seeking control of a mode-locked laser. *IEEE Journal of Quantum Electronics*, 49(10):852–861, 2013.

[130] S. L. Brunton, X. Fu, and J. N. Kutz. Self-tuning fiber lasers. *IEEE Journal of Selected Topics in Quantum Electronics*, 20(5):464–471, 2014.

[131] S. L. Brunton, B. R. Noack, and P. Koumoutsakos. Machine learning for fluid mechanics. *Annual Review of Fluid Mechanics*, 52:477–508, 2020.

[132] S. L. Brunton, J. L. Proctor, and J. N. Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences, USA*, 113(15):3932–3937, 2016.

[133] S. L. Brunton, J. L. Proctor, and J. N. Kutz. Sparse identification of nonlinear dynamics with control (SINDYc). *IFAC-PapersOnLine*, 49(18):710–715, 2016 (10th IFAC Symposium on Nonlinear Control Systems, NOLCOS).

[134] S. L. Brunton, J. L. Proctor, J. H. Tu, and J. N. Kutz. Compressed sensing and dynamic mode decomposition. *Journal of Computational Dynamics*, 2(2):165–191, 2015.

[135] S. L. Brunton, J. L. Proctor, J. H. Tu, and J. N. Kutz. Compressed sensing and dynamic mode decomposition. *Journal of Computational Dynamics*, 2(2):165, 2015.

[136] S. L. Brunton, J. H. Tu, I. Bright, and J. N. Kutz. Compressive sensing and low-rank libraries for classification of bifurcation regimes in nonlinear dynamical systems. *SIAM Journal on Applied Dynamical Systems*, 13(4):1716–1732, 2014.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

[137] D. Buche, P. Stoll, R. Dornberger, and P. Koumoutsakos. Multiobjective evolutionary algorithm for the optimization of noisy combustion processes. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 32(4):460–473, 2002.

[138] M. Budišić and I. Mezić. An approximate parametrization of the ergodic partition using time averaged observables. In *Proceedings of the 48th IEEE Conference on Decision and Control, held jointly with the 28th Chinese Control Conference*, pages 3162–3168. IEEE, 2009.

[139] M. Budišić and I. Mezić. Geometry of the ergodic quotient reveals coherent structures in flows. *Physica D: Nonlinear Phenomena*, 241(15):1255–1269, 2012.

[140] M. Budišić, R. Mohr, and I. Mezić. Applied Koopmanism. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 22(4):047510, 2012.

[141] A. Buhr and K. Smetana. Randomized local model order reduction. *SIAM Journal on Scientific Computing*, 40(4):A2120–A2151, 2018.

[142] K. P. Burnham and D. R. Anderson. *Model Selection and Multimodel Inference: A Practical Information-Theoretic Approach*. Springer, 2003.

[143] D. Burov, D. Giannakis, K. Manohar, and A. Stuart. Kernel analog forecasting: Multiscale test problems. *Multiscale Modeling and Simulation*, 19(2):1011–1040, 2020.

[144] P. A. Businger and G. H. Golub. Algorithm 358: Singular value decomposition of a complex matrix [F1, 4, 5]. *Communications of the ACM*, 12(10):564–565, 1969.

[145] J. L. Callaham, S. L. Brunton, and J.-C. Loiseau. On the role of nonlinear correlations in reduced-order modeling. Preprint arXiv:2106.02409, 2021.

[146] J. L. Callaham, J.-C. Loiseau, G. Rigas, and S. L. Brunton. Nonlinear stochastic modelling with Langevin regression. *Proceedings of the Royal Society A*, 477(2250):20210092, 2021.

[147] J. L. Callaham, K. Maeda, and S. L. Brunton. Robust flow reconstruction from limited measurements via sparse representation. *Physical Review Fluids*, 4:103907, 2019.

[148] J. L. Callaham, G. Rigas, J.-C. Loiseau, and S. L. Brunton. An empirical mean-field model of symmetry-breaking in a turbulent wake. Preprint arXiv:2105.13990, 2021.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

[149] E. F. Camacho and C. B. Alba. *Model Predictive Control*. Springer, 2013.

[150] E. Cambria, G.-B. Huang, L. L. C. Kasun, et al. Extreme learning machines [trends & controversies]. *IEEE Intelligent Systems*, 28(6):30–59, 2013.

[151] E. J. Candès. Compressive sensing. *Proceedings of the International Congress of Mathematicians*, 2006.

[152] E. J. Candès and T. Tao. Decoding by linear programming. *IEEE Transactions on Information Theory*, 51(12):4203–4215, 2005.

[153] E. J. Candès and T. Tao. Near optimal signal recovery from random projections: Universal encoding strategies? *IEEE Transactions on Information Theory*, 52(12):5406–5425, 2006.

[154] E. J. Candès and M. B. Wakin. An introduction to compressive sampling. *IEEE Signal Processing Magazine*, 25(2):21–30, 2008.

[155] E. J. Candès, X. Li, Y. Ma, and J. Wright. Robust principal component analysis? *Journal of the ACM*, 58(3):11 (37pp.), 2011.

[156] E. J. Candès, J. Romberg, and T. Tao. Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information. *IEEE Transactions on Information Theory*, 52(2):489–509, 2006.

[157] E. J. Candès, J. Romberg, and T. Tao. Stable signal recovery from incomplete and inaccurate measurements. *Communications on Pure and Applied Mathematics*, 59(8):1207–1223, 2006.

[158] Y. Cao, J. Zhu, Z. Luo, and I. Navon. Reduced-order modeling of the upper tropical Pacific Ocean model using proper orthogonal decomposition. *Computers and Mathematics with Applications*, 52(8):1373–1386, 2006.

[159] Y. Cao, J. Zhu, I. M. Navon, and Z. Luo. A reduced-order approach to four-dimensional variational data assimilation using proper orthogonal decomposition. *International Journal for Numerical Methods in Fluids*, 53(10):1571–1583, 2007.

[160] K. Carlberg, M. Barone, and H. Antil. Galerkin v. least-squares Petrov–Galerkin projection in nonlinear model reduction. *Journal of Computational Physics*, 330:693–734, 2017.

[161] K. Carlberg, C. Bou-Mosleh, and C. Farhat. Efficient non-linear model reduction via a least-squares Petrov–Galerkin projection and compressive tensor approximations. *International Journal for Numerical Methods in Engineering*, 86(2):155–181, 2011.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

[162] K. Carlberg, C. Farhat, J. Cortial, and D. Amsallem. The GNAT method for nonlinear model reduction: Effective implementation and application to computational fluid dynamics and turbulent flows. *Journal of Computational Physics*, 242:623–647, 2013.

[163] T. Carleman. Application de la théorie des équations intégrales linéaires aux systèmes d'équations différentielles non linéaires. *Acta Mathematica*, 59(1):63–87, 1932.

[164] T. Carleman. Sur la théorie de l'équation intégrodifférentielle de Boltzmann. *Acta Mathematica*, 60(1):91–146, 1933.

[165] T. Carleman. Sur les systèmes linéaires aux dérivées partielles du premier ordre à deux variables. *Comptes Rendus de l'Académie des Sciences, Paris*, 197:471–474, 1933.

[166] J. D. Carroll and J.-J. Chang. Analysis of individual differences in multidimensional scaling via an N-way generalization of "Eckart–Young" decomposition. *Psychometrika*, 35:283–319, 1970.

[167] K. P. Champion, S. L. Brunton, and J. N. Kutz. Discovery of nonlinear multiscale systems: Sampling strategies and embeddings. *SIAM Journal on Applied Dynamical Systems*, 18(1):312–333, 2019.

[168] K. P. Champion, B. Lusch, J. N. Kutz, and S. L. Brunton. Data-driven discovery of coordinates and governing equations. *Proceedings of the National Academy of Sciences, USA*, 116(45):22 445–22 451, 2019.

[169] R. Chartrand. Numerical differentiation of noisy, nonsmooth data. *ISRN Applied Mathematics*, 2011:164564, 2011.

[170] A. Chatterjee. An introduction to the proper orthogonal decomposition. *Current Science*, 78(7):808–817, 2000.

[171] S. Chaturantabut and D. C. Sorensen. Nonlinear model reduction via discrete empirical interpolation. *SIAM Journal on Scientific Computing*, 32(5):2737–2764, 2010.

[172] K. K. Chen and C. W. Rowley. Normalized coprime robust stability and performance guarantees for reduced-order controllers. *IEEE Transactions on Automatic Control*, 58(4):1068–1073, 2013.

[173] K. K. Chen, J. H. Tu, and C. W. Rowley. Variants of dynamic mode decomposition: Boundary condition, Koopman, and Fourier analyses. *Journal of Nonlinear Science*, 22(6):887–915, 2012.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

[174] T. Chen and H. Chen. Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems. *IEEE Transactions on Neural Networks*, 6(4):911–917, 1995.

[175] Y. Chen, K. L. Moore, and H.-S. Ahn. Iterative learning control. In *Encyclopedia of the Sciences of Learning*, pages 1648–1652. Springer, 2012.

[176] S. Cherry. Singular value decomposition analysis and canonical correlation analysis. *Journal of Climate*, 9(9):2003–2009, 1996.

[177] K. Cho, B. Van Merriënboer, C. Gulcehre, et al. Learning phrase representations using RNN encoder–decoder for statistical machine translation. Preprint arXiv:1406.1078, 2014.

[178] J. Choi, M. Krstić, K. Ariyur, and J. Lee. Extremum seeking control for discrete-time systems. *IEEE Transactions on Automatic Control*, 47(2):318–323, 2002.

[179] Y. Choi, D. Amsallem, and C. Farhat. Gradient-based constrained optimization using a database of linear reduced-order models. Preprint arXiv:1506.07849, 2015.

[180] S. Colabrese, K. Gustavsson, A. Celani, and L. Biferale. Flow navigation by smart microswimmers via reinforcement learning. *Physical Review Letters*, 118(15):158004, 2017.

[181] T. Colonius and K. Taira. A fast immersed boundary method using a nullspace approach and multi-domain far-field boundary conditions. *Computer Methods in Applied Mechanics and Engineering*, 197:2131–2146, 2008.

[182] J. W. Cooley and J. W. Tukey. An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computation*, 19(90):297–301, 1965.

[183] J. W. Cooley, P. A. Lewis, and P. D. Welch. Historical notes on the fast Fourier transform. *Proceedings of the IEEE*, 55(10):1675–1677, 1967.

[184] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[185] M. C. Cross and P. C. Hohenberg. Pattern formation outside of equilibrium. *Reviews of Modern Physics*, 65(3):851, 1993.

[186] J. P. Crutchfield and B. S. McNamara. Equations of motion from a data series. *Complex Systems*, 1:417–452, 1987.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

[187] M. Dam, M. Brøns, J. Juul Rasmussen, V. Naulin, and J. S. Hesthaven. Sparse identification of a predator–prey system from simulation data of a convection model. *Physics of Plasmas*, 24(2):022310, 2017.

[188] B. C. Daniels and I. Nemenman. Automated adaptive inference of phenomenological dynamical models. *Nature Communications*, 6:8133, 2015.

[189] B. C. Daniels and I. Nemenman. Efficient inference of parsimonious phenomenological models of cellular dynamics using S-systems and alternating regression. *PloS ONE*, 10(3):e0119821, 2015.

[190] S. Das and D. Giannakis. Delay-coordinate maps and the spectra of Koopman operators. *Journal of Statistical Physics*, 175(6):1107–1145, 2019.

[191] S. Das and D. Giannakis. Koopman spectra in reproducing kernel Hilbert spaces. *Applied and Computational Harmonic Analysis*, 49(2):573–607, 2020.

[192] I. Daubechies. The wavelet transform, time–frequency localization and signal analysis. *IEEE Transactions on Information Theory*, 36(5):961–1005, 1990.

[193] L. Davis. *Handbook of Genetic Algorithms*, volume 115 of VNR Computer Library. Van Nostrand Reinhold, 1991.

[194] N. D. Daw, J. P. O'Doherty, P. Dayan, B. Seymour, and R. J. Dolan. Cortical substrates for exploratory decisions in humans. *Nature*, 441(7095):876–879, 2006.

[195] S. T. Dawson, M. S. Hemati, M. O. Williams, and C. W. Rowley. Characterizing and correcting for the effect of sensor noise in the dynamic mode decomposition. *Experiments in Fluids*, 57(3):1–19, 2016.

[196] P. Dayan and L. F. Abbott. *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*. In Computational Neuroscience Series. MIT Press, 2001.

[197] P. Dayan and T. J. Sejnowski. TD(λ) converges with probability 1. *Machine Learning*, 14(3):295–301, 1994.

[198] B. M. de Silva, D. M. Higdon, S. L. Brunton, and J. N. Kutz. Discovery of physics from data: Universal laws and discrepancies. *Frontiers in Artificial Intelligence*, 3:25, 2020.

[199] B. M. de Silva, K. Champion, M. Quade, et al. PySINDy: a Python package for the sparse identification of nonlinear dynamics from data. *Journal of Open Source Software*, 5(49):2104, 2020.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

[200] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977.

[201] N. Deng, B. R. Noack, M. Morzyński, and L. R. Pastur. Galerkin force model for transient and post-transient dynamics of the fluidic pinball. *Journal of Fluid Mechanics*, 918:A4, 2021.

[202] Z. Deng, C. He, Y. Liu, and K. C. Kim. Super-resolution reconstruction of turbulent velocity fields using a generative adversarial network-based artificial intelligence framework. *Physics of Fluids*, 31(12):125111, 2019.

[203] S. Devasia, D. Chen, and B. Paden. Nonlinear inversion-based output tracking. *IEEE Transactions on Automatic Control*, 41(7):930–942, 1996.

[204] D. L. Donoho. Compressed sensing. *IEEE Transactions on Information Theory*, 52(4):1289–1306, 2006.

[205] D. L. Donoho. 50 years of data science. *Journal of Computational and Graphical Statistics*, 26(4):745-766, 2017 (Based on a Presentation at the Tukey Centennial Workshop, Princeton, NJ, 2015).

[206] D. L. Donoho and M. Gavish. Code supplement to "The optimal hard threshold for singular values is $4/\sqrt{3}$." `http://purl.stanford.edu/vg705qn9070`, 2014.

[207] D. L. Donoho and I. M. Johnstone. Ideal spatial adaptation by wavelet shrinkage. *Biometrika*, 81(3):425–455, 1994.

[208] D. L. Donoho, I. M. Johnstone, J. C. Hoch, and A. S. Stern. Maximum entropy and the nearly black object. *Journal of the Royal Statistical Society. Series B (Methodological)*, 54(1):41–81, 1992.

[209] J. C. Doyle. Guaranteed margins for LQG regulators. *IEEE Transactions on Automatic Control*, 23(4):756–757, 1978.

[210] J. C. Doyle, B. A. Francis, and A. R. Tannenbaum. *Feedback Control Theory*. Courier Corporation, 2013.

[211] J. C. Doyle, K. Glover, P. P. Khargonekar, and B. A. Francis. State-space solutions to standard H_2 and H_∞ control problems. *IEEE Transactions on Automatic Control*, 34(8):831–847, 1989.

[212] P. Drews, G. Williams, B. Goldfain, E. A. Theodorou, and J. M. Rehg. Vision-based high-speed driving with a deep dynamic observer. *IEEE Robotics and Automation Letters*, 4(2):1564–1571, 2019.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

[213] J. Drgona, K. Kis, A. Tuor, D. Vrabie, and M. Klauco. Differentiable predictive control: An MPC alternative for unknown nonlinear systems using constrained deep learning. Preprint arXiv:2011.03699, 2020.

[214] P. Drineas and M. W. Mahoney. A randomized algorithm for a tensor-based generalization of the singular value decomposition. *Linear Algebra and Its Applications*, 420(2–3):553–571, 2007.

[215] Z. Drmac and S. Gugercin. A new selection operator for the discrete empirical interpolation method – improved *a priori* error bound and extensions. *SIAM Journal on Scientific Computing*, 38(2):A631–A648, 2016.

[216] Q. Du and M. Gunzburger. Model reduction by proper orthogonal decomposition coupled with centroidal Voronoi tessellations. In *ASME 2002 Joint US–European Fluids Engineering Division Conference*, pages 1401–1406. ASME, 2002.

[217] Y. Duan, M. Andrychowicz, B. C. Stadie, et al. One-shot imitation learning. Preprint arXiv:1703.07326, 2017.

[218] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. Wiley-Interscience, 2000.

[219] J. A. Duersch and M. Gu. Randomized QR with column pivoting. *SIAM Journal on Scientific Computing*, 39(4):C263–C291, 2017.

[220] D. Duke, D. Honnery, and J. Soria. Experimental investigation of nonlinear instabilities in annular liquid sheets. *Journal of Fluid Mechanics*, 691:594–604, 2012.

[221] D. Duke, J. Soria, and D. Honnery. An error analysis of the dynamic mode decomposition. *Experiments in Fluids*, 52(2):529–542, 2012.

[222] G. E. Dullerud and F. Paganini. *A Course in Robust Control Theory: A Convex Approach*. In Texts in Applied Mathematics. Springer, 2000.

[223] R. Dunne and B. J. McKeon. Dynamic stall on a pitching and surging airfoil. *Experiments in Fluids*, 56(8):1–15, 2015.

[224] K. Duraisamy, G. Iaccarino, and H. Xiao. Turbulence modeling in the age of data. *Annual Reviews of Fluid Mechanics*, 51:357–377, 2019.

[225] T. Duriez, S. L. Brunton, and B. R. Noack. *Machine Learning Control: Taming Nonlinear Dynamics and Turbulence*. Springer, 2016.

[226] T. Duriez, V. Parezanović, L. Cordier, et al. Closed-loop turbulence control using machine learning. Preprint arXiv:1404.4589, 2014.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

[227] T. Duriez, V. Parezanovic, J.-C. Laurentie, et al. Closed-loop control of experimental shear flows using machine learning. AIAA Paper 2014-2219, 2014 (7th Flow Control Conference).

[228] C. Eckart and G. Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3):211–218, 1936.

[229] J. L. Eftang, A. T. Patera, and E. M. Rønquist. An "hp" certified reduced basis method for parametrized elliptic partial differential equations. *SIAM Journal on Scientific Computing*, 32(6):3170–3200, 2010.

[230] J. L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990.

[231] U. Eren, A. Prach, B. B. Koçer, et al. Model predictive control in aerospace systems: Current state and opportunities. *Journal of Guidance, Control, and Dynamics*, 40(7):1541–1566, 2017.

[232] N. B. Erichson, S. L. Brunton, and J. N. Kutz. Compressed dynamic mode decomposition for background modeling. *Journal of Real-Time Image Processing*, 16:1479–1492, 2019.

[233] N. B. Erichson, K. Manohar, S. L. Brunton, and J. N. Kutz. Randomized CP tensor decomposition. *Machine Learning: Science and Technology*, 1(2):025012, 2020.

[234] N. B. Erichson, L. Mathelin, J. N. Kutz, and S. L. Brunton. Randomized dynamic mode decomposition. *SIAM Journal on Applied Dynamical Systems*, 18(4):1867–1891, 2019.

[235] N. B. Erichson, L. Mathelin, Z. Yao, et al. Shallow neural networks for fluid flow reconstruction with limited sensors. *Proceedings of the Royal Society A*, 476(2238):20200097, 2020.

[236] N. B. Erichson, S. Voronin, S. L. Brunton, and J. N. Kutz. Randomized matrix decompositions using R. *Journal of Statistical Software*, 89(11):1–48, 2019.

[237] T. Esram, J. W. Kimball, P. T Krein, P. L. Chapman, and P. Midya. Dynamic maximum power point tracking of photovoltaic arrays using ripple correlation control. *IEEE Transactions on Power Electronics*, 21(5):1282–1291, 2006.

[238] E. Even-Dar, Y. Mansour, and P. Bartlett. Learning rates for Q-learning. *Journal of Machine Learning Research*, 5:1–25, 2003.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

[239] R. Everson and L. Sirovich. Karhunen–Loeve procedure for gappy data. *Journal of the Optical Society of America A*, 12(8):1657–1664, 1995.

[240] N. Fabbiane, O. Semeraro, S. Bagheri, and D. S. Henningson. Adaptive and model-based control theory applied to convectively unstable flows. *Applied Mechanics Reviews*, 66(6):060801 (20pp.), 2014.

[241] D. Fan, L. Yang, Z. Wang, M. S. Triantafyllou, and G. E. Karniadakis. Reinforcement learning for bluff body active flow control in experiments and simulations. *Proceedings of the National Academy of Sciences, USA*, 117(42):26 091–26 098, 2020.

[242] D. D. Fan, A.-A. Agha-Mohammadi, and E. A. Theodorou. Deep learning tubes for tube MPC. Preprint arXiv:2002.01587, 2020.

[243] B. Feeny. On proper orthogonal co-ordinates as indicators of modal activity. *Journal of Sound and Vibration*, 255(5):805–817, 2002.

[244] R. A. Fisher. On the mathematical foundations of theoretical statistics. *Philosophical Transactions of the Royal Society A*, 222:309–368, 1922.

[245] R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Human Genetics*, 7(2):179–188, 1936.

[246] P. J. Fleming and R. C. Purshouse. Evolutionary algorithms in control systems engineering: A survey. *Control Engineering Practice*, 10:1223–1241, 2002.

[247] N. Fonzi, S. L. Brunton, and U. Fasel. Data-driven nonlinear aeroelastic models of morphing wings for control. *Proceedings of the Royal Society A*, 476(2239):20200079, 2020.

[248] J. Fourier. *Théorie Analytique de la Chaleur*. Firmin Didot, Père et Fils, 1822.

[249] J. B. J. Fourier. *The Analytical Theory of Heat*. Cambridge at The University Press, 1878 (transl. A. Freeman).

[250] J. E. Fowler. Compressive-projection principal component analysis. *IEEE Transactions on Image Processing*, 18(10):2230–2242, 2009.

[251] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

[252] J. H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29(5):1189–1232, 2001.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

[253] A. Frieze, R. Kannan, and S. Vempala. Fast Monte-Carlo algorithms for finding low-rank approximations. *Journal of the ACM*, 51(6):1025–1041, 2004.

[254] G. Froyland, G. A. Gottwald, and A. Hammerlindl. A computational method to extract macroscopic variables and their dynamics in multiscale systems. *SIAM Journal on Applied Dynamical Systems*, 13(4):1816–1846, 2014.

[255] G. Froyland, G. A. Gottwald, and A. Hammerlindl. A trajectory-free framework for analysing multiscale systems. *Physica D: Nonlinear Phenomena*, 328:34–43, 2016.

[256] X. Fu, S. L. Brunton, and J. N. Kutz. Classification of birefringence in mode-locked fiber lasers using machine learning and sparse representation. *Optics Express*, 22(7):8585–8597, 2014.

[257] K. Fujii and Y. Kawahara. Dynamic mode decomposition in vector-valued reproducing kernel Hilbert spaces for extracting dynamical structure among observables. *Neural Networks*, 117:94–103, 2019.

[258] S. Fujimoto, H. Hoof, and D. Meger. Addressing function approximation error in actor–critic methods. In *Proceedings of the 35th International Conference on Machine Learning*, pages 1587–1596. PMLR, 2018.

[259] K. Fukagata, S. Kern, P. Chatelain, P. Koumoutsakos, and N. Kasagi. Evolutionary optimization of an anisotropic compliant surface for turbulent friction drag reduction. *Journal of Turbulence*, 9(35):1–17, 2008.

[260] F. Fukushima. A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36:193–202, 1980.

[261] H. Gao, J. Lam, C. Wang, and Y. Wang. Delay-dependent output-feedback stabilisation of discrete-time systems with time-varying state delay. *IEE Proceedings – Control Theory and Applications*, 151(6):691–698, 2004.

[262] C. E. Garcia, D. M. Prett, and M. Morari. Model predictive control: theory and practice – A survey. *Automatica*, 25(3):335–348, 1989.

[263] J. L. Garriga and M. Soroush. Model predictive control tuning methods: A review. *Industrial and Engineering Chemistry Research*, 49(8):3505–3515, 2010.

[264] C.-F. Gauss. *Theoria Combinationis Observationum Erroribus Minimis Obnoxiae*, volume 1. Henricus Dieterich, 1823.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

[265] C.-F. Gauss. *Theoria Interpolationis Methodo Nova Tractata*. Königliche Gesellschaft der Wissenschaften, Göttingen, 1866.

[266] N. Gautier, J.-L. Aider, T. Duriez, et al. Closed-loop separation control using machine learning. *Journal of Fluid Mechanics*, 770:442–457, 2015.

[267] M. Gavish and D. L. Donoho. The optimal hard threshold for singular values is $4/\sqrt{3}$. *IEEE Transactions on Information Theory*, 60(8):5040–5053, 2014.

[268] M. Gazzola, B. Hejazialhosseini, and P. Koumoutsakos. Reinforcement learning and wavelet adapted vortex methods for simulations of self-propelled swimmers. *SIAM Journal on Scientific Computing*, 36(3):B622–B639, 2014.

[269] M. Gazzola, A. Tchieu, D. Alexeev, A. De Brauer, and P. Koumoutsakos. Learning to school in the presence of hydrodynamic interactions. *Journal of Fluid Mechanics*, 789:726–749, 2016.

[270] M. Gazzola, O. V. Vasilyev, and P. Koumoutsakos. Shape optimization for drag reduction in linked bodies using evolution strategies. *Computers and Structures*, 89(11):1224–1231, 2011.

[271] T. Geijtenbeek, M. Van De Panne, and A. F. Van Der Stappen. Flexible muscle-based locomotion for bipedal creatures. *ACM Transactions on Graphics*, 32(6):1–11, 2013.

[272] G. Gelbert, J. P. Moeck, C. O. Paschereit, and R. King. Advanced algorithms for gradient estimation in one- and two-parameter extremum seeking controllers. *Journal of Process Control*, 22(4):700–709, 2012.

[273] P. Gelß, S. Klus, J. Eisert, and C. Schütte. Multidimensional approximation of nonlinear dynamical systems. *Journal of Computational and Nonlinear Dynamics*, 14(6):061006 (12pp.), 2019.

[274] A. S. Georghiades, P. N. Belhumeur, and D. J. Kriegman. From few to many: Illumination cone models for face recognition under variable lighting and pose. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(6):643–660, 2001.

[275] J. J. Gerbrands. On the relationships between SVD, KLT and PCA. *Pattern Recognition*, 14(1):375–381, 1981.

[276] A. C. Gilbert and P. Indyk. Sparse recovery using sparse matrices. *Proceedings of the IEEE*, 98(6):937–947, 2010.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

[277] A. C. Gilbert, J. Y. Park, and M. B. Wakin. Sketched SVD: Recovering spectral features from compressive measurements. Preprint arXiv:1211.0361, 2012.

[278] A. C. Gilbert, M. J. Strauss, and J. A. Tropp. A tutorial on fast Fourier sampling. *IEEE Signal Processing Magazine*, 25(2):57–66, 2008.

[279] C. Gin, B. Lusch, S. L. Brunton, and J. N. Kutz. Deep learning models for global coordinate transformations that linearise PDEs. *European Journal of Applied Mathematics*, 32(3):515–539, 2021.

[280] C. R. Gin, D. E. Shea, S. L. Brunton, and J. N. Kutz. DeepGreen: Deep learning of Green's functions for nonlinear boundary value problems. *Scientific Reports*, 11:21614, 2021.

[281] B. Glaz, L. Liu, and P. P. Friedmann. Reduced-order nonlinear unsteady aerodynamic modeling using a surrogate-based recurrence framework. *AIAA Journal*, 48(10):2418–2429, 2010.

[282] P. W. Glimcher. Understanding dopamine and reinforcement learning: The dopamine reward prediction error hypothesis. *Proceedings of the National Academy of Sciences, USA*, 108(Supplement 3):15 647–15 654, 2011.

[283] P. J. Goddard and K. Glover. Controller approximation: Approaches for preserving H_∞ performance. *IEEE Transactions on Automatic Control*, 43(7):858–871, 1998.

[284] D. E. Goldberg. *Genetic Algorithms*. Pearson Education India, 2006.

[285] G. H. Golub and W. Kahan. Calculating the singular values and pseudo-inverse of a matrix. *Journal of the Society for Industrial and Applied Mathematics, Series B: Numerical Analysis*, 2(2):205–224, 1965.

[286] G. H. Golub and C. Reinsch. Singular value decomposition and least squares solutions. *Numerical Mathematics*, 14:403–420, 1970.

[287] G. H. Golub and C. F. Van Loan. *Matrix Computations*, 3rd edition. Johns Hopkins University Press, 2012.

[288] G. H. Golub, S. Nash, and C. Van Loan. A Hessenberg–Schur method for the problem $ax+xb = c$. *IEEE Transactions on Automatic Control*, 24(6):909–913, 1979.

[289] R. González-García, R. Rico-Martínez, and I. G. Kevrekidis. Identification of distributed parameter systems: A neural net based approach. *Computers and Chemical Engineering*, 22:S965–S968, 1998.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

[290] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

[291] I. Goodfellow, J. Pouget-Abadie, M. Mirza, et al. Generative adversarial nets. *Advances in Neural Information Processing Systems, 27*, 9pp., 2014.

[292] A. Goza and T. Colonius. Modal decomposition of fluid–structure interaction with application to flag flapping. *Journal of Fluids and Structures*, 81:728–737, 2018.

[293] M. Grant, S. Boyd, and Y. Ye. CVX: Matlab software for disciplined convex programming, `http://cvxr.com/cvx`, 2008.

[294] A. Graves, G. Wayne, and I. Danihelka. Neural Turing machines. Preprint arXiv:1410.5401, 2014.

[295] A. Greenbaum. *Iterative methods for solving linear systems*. SIAM, 1997.

[296] M. S. Grewal. Kalman filtering. In *International Encyclopedia of Statistical Science*, pages 705–708. Springer, 2011.

[297] M. Grilli, P. J. Schmid, S. Hickel, and N. A. Adams. Analysis of unsteady behaviour in shockwave turbulent boundary layer interaction. *Journal of Fluid Mechanics*, 700:16–28, 2012.

[298] J. Grosek and J. N. Kutz. Dynamic mode decomposition for real-time background/foreground separation in video. Preprint arXiv:1404.7592, 2014.

[299] M. Gu. Subspace iteration randomization and singular value problems. *SIAM Journal on Scientific Computing*, 37(3):1139–1173, 2015.

[300] S. Gu, E. Holly, T. Lillicrap, and S. Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *2017 IEEE International Conference on Robotics and Automation*, pages 3389–3396. IEEE, 2017.

[301] Y. Guan, S. L. Brunton, and I. Novosselov. Sparse nonlinear models of chaotic electroconvection. *Royal Society Open Science*, 8(8):202367, 2021.

[302] F. Guéniat, L. Mathelin, and M. Y. Hussaini. A statistical learning strategy for closed-loop control of fluid flows. *Theoretical and Computational Fluid Dynamics*, 30(6):497–510, 2016.

[303] F. Guéniat, L. Mathelin, and L. Pastur. A dynamic mode decomposition approach for large and arbitrarily sampled systems. *Physics of Fluids*, 27(2):025113, 2015.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

[304] P. Gunnarson, I. Mandralis, G. Novati, P. Koumoutsakos, and J. O. Dabiri. Learning efficient navigation in vortical flow fields. Preprint arXiv:2102.10536, 2021.

[305] D. R. Gurevich, P. A. Reinbold, and R. O. Grigoriev. Robust and optimal sparse regression for nonlinear pde models. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 29(10):103113, 2019.

[306] F. Gustafsson, F. Gunnarsson, N. Bergman, et al. Particle filters for positioning, navigation, and tracking. *IEEE Transactions on Signal Processing*, 50(2):425–437, 2002.

[307] A. Haar. Zur Theorie der orthogonalen Funktionensysteme. *Mathematische Annalen*, 69(3):331–371, 1910.

[308] N. Halko, P.-G. Martinsson, Y. Shkolnisky, and M. Tygert. An algorithm for the principal component analysis of large data sets. *SIAM Journal on Scientific Computing*, 33:2580–2594, 2011.

[309] N. Halko, P.-G. Martinsson, and J. A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Review*, 53(2):217–288, 2011.

[310] S. J. Hammarling. Numerical solution of the stable, non-negative definite Lyapunov equation. *IMA Journal of Numerical Analysis*, 2(3):303–323, 1982.

[311] S. Han and B. Feeny. Application of proper orthogonal decomposition to structural vibration analysis. *Mechanical Systems and Signal Processing*, 17(5):989–1001, 2003.

[312] N. Hansen, A. S. Niederberger, L. Guzzella, and P. Koumoutsakos. A method for handling uncertainty in evolutionary optimization with an application to feedback control of combustion. *IEEE Transactions on Evolutionary Computation*, 13(1):180–197, 2009.

[313] D. Harrison Jr. and D. L. Rubinfeld. Hedonic housing prices and the demand for clean air. *Journal of Environmental Economics and Management*, 5(1):81–102, 1978.

[314] R. A. Harshman. Foundations of the PARAFAC procedure: Models and conditions for an "explanatory" multi-modal factor analysis. *UCLA Working Papers in Phonetics*, 16:1–84, 1970. Available at `http://www.psychology.uwo.ca/faculty/harshman/wpppfac0.pdf`.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

[315] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*, 2nd edition. Springer, 2009.

[316] M. Hausknecht and P. Stone. Deep recurrent Q-learning for partially observable MDPs. In *2015 AAAI Fall Symposium Series*, 2015.

[317] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[318] M. Heath, A. Laub, C. Paige, and R. Ward. Computing the singular value decomposition of a product of two matrices. *SIAM Journal on Scientific and Statistical Computing*, 7(4):1147–1159, 1986.

[319] M. Heideman, D. Johnson, and C. Burrus. Gauss and the history of the fast Fourier transform. *IEEE ASSP Magazine*, 1(4):14–21, 1984.

[320] W. Heisenberg. Über den anschaulichen Inhalt der quantentheoretischen Kinematik und Mechanik. In *Werner Heisenberg Gesammelte Werke (Collected Works): Original Scientific Papers (Wissenschaftliche Originalarbeiten)*, pages 478–504. Springer, 1985.

[321] M. S. Hemati, C. W. Rowley, E. A. Deem, and L. N. Cattafesta. De-biasing the dynamic mode decomposition for applied Koopman spectral analysis. *Theoretical and Computational Fluid Dynamics*, 31(4):349–368, 2017.

[322] M. S. Hemati, M. O. Williams, and C. W. Rowley. Dynamic mode decomposition for large and streaming datasets. *Physics of Fluids*, 26(11):111701, 2014.

[323] K. K. Herrity, A. C. Gilbert, and J. A. Tropp. Sparse approximation via iterative thresholding. In *2006 IEEE International Conference on Acoustics, Speech and Signal Processing Proceedings*, volume 3. IEEE, 2006.

[324] B. Herrmann, P. J. Baddoo, R. Semaan, S. L. Brunton, and B. J. McKeon. Data-driven resolvent analysis. *Journal of Fluid Mechanics*, 918:A10, 2021.

[325] J. S. Hesthaven, G. Rozza, and B. Stamm. *Certified Reduced Basis Methods for Parametrized Partial Differential Equations*. In Springer Briefs in Mathematics, Springer, 2015.

[326] T. Hey, S. Tansley, and K. M. Tolle. *The Fourth Paradigm: Data-Intensive Scientific Discovery*. Microsoft Research, 2009.

[327] G. E. Hinton and T. J. Sejnowski. Learning and relearning in Boltzmann machines. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1, *Foundations*, pages 282–317. ACM, 1986.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

[328] S. M. Hirsh, S. M. Ichinaga, S. L. Brunton, J. N. Kutz, and B. W. Brunton. Structured time-delay models for dynamical systems with connections to Frenet–Serret frame. *Proceedings of the Royal Society A*, 477(2254): 20210097, 2021.

[329] B. L. Ho and R. E. Kalman. Effective construction of linear state-variable models from input/output data. In *Proceedings of the 3rd Annual Allerton Conference on Circuit and System Theory*, pages 449–459, 1965.

[330] J. Ho and S. Ermon. Generative adversarial imitation learning. *Advances in Neural Information Processing Systems, 29*, pages 4565–4573, 2016.

[331] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[332] A. E. Hoerl and R. W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.

[333] J. H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence.* University of Michigan Press, 1975.

[334] P. Holmes and J. Guckenheimer. *Nonlinear Oscillations, Dynamical Systems, and Bifurcations of Vector Fields*, volume 42 of Applied Mathematical Sciences. Springer, 1983.

[335] P. Holmes, J. L. Lumley, G. Berkooz, and C. W. Rowley. *Turbulence, Coherent Structures, Dynamical Systems and Symmetry*, 2nd edition. Cambridge University Press, 2012.

[336] E. Hopf. The partial differential equation $u_t + uu_x = \mu u_{xx}$. *Communications on Pure and Applied Mathematics*, 3(3):201–230, 1950.

[337] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences, USA*, 79(8):2554–2558, 1982.

[338] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[339] H. Hotelling. Analysis of a complex of statistical variables into principal components [to be concluded]. *Journal of Educational Psychology*, 24:417–441, 1933.

[340] H. Hotelling. Analysis of a complex of statistical variables into principal components [concluded]. *Journal of Educational Psychology*, 24:498–520, 1933.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

[341] C. Huang, W. E. Anderson, M. E. Harvazinski, and V. Sankaran. Analysis of self-excited combustion instabilities using decomposition techniques. In *51st AIAA Aerospace Sciences Meeting*, pages 1–18, 2013.

[342] D. H. Hubel and T. N. Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *Journal of Physiology*, 160:106–154, 1962.

[343] P. J. Huber. Robust statistics. In *International Encyclopedia of Statistical Science*, pages 1248–1251. Springer, 2011.

[344] A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne. Imitation learning: A survey of learning methods. *ACM Computing Surveys*, 50(2):1–35, 2017.

[345] S. J. Illingworth, A. S. Morgans, and C. W. Rowley. Feedback control of flow resonances using balanced reduced-order models. *Journal of Sound and Vibration*, 330(8):1567–1581, 2010.

[346] E. Jacobsen and R. Lyons. The sliding DFT. *IEEE Signal Processing Magazine*, 20(2):74–80, 2003.

[347] H. Jaeger and H. Haas. Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *Science*, 304(5667):78–80, 2004.

[348] G. James, D. Witten, T. Hastie, and R. Tibshirani. *An Introduction to Statistical Learning*. Springer, 2013.

[349] M. C. Johnson, S. L. Brunton, N. B. Kundtz, and J. N. Kutz. Extremum-seeking control of a beam pattern of a reconfigurable holographic metamaterial antenna. *Journal of the Optical Society of America A*, 33(1):59–68, 2016.

[350] R. A. Johnson and D. Wichern. *Applied Multivariate Statistical Analysis*, 6th edition. Pearson, 2018.

[351] W. B. Johnson and J. Lindenstrauss. Extensions of Lipschitz mappings into a Hilbert space. *Contemporary Mathematics*, 26(189-206):1, 1984.

[352] I. Jolliffe. Principal component analysis. In *Encyclopedia of Statistics in Behavioral Science*. John Wiley & Sons, 2005.

[353] S. Joshi and S. Boyd. Sensor selection via convex optimization. *IEEE Transactions on Signal Processing*, 57(2):451–462, 2009.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

[354] M. R. Jovanović. From bypass transition to flow control and data-driven turbulence modeling: An input–output viewpoint. *Annual Review of Fluid Mechanics*, 53(1):311–345, 2021.

[355] M. R. Jovanović and B. Bamieh. Componentwise energy amplification in channel flows. *Journal of Fluid Mechanics*, 534:145–183, 2005.

[356] M. R. Jovanović, P. J. Schmid, and J. W. Nichols. Sparsity-promoting dynamic mode decomposition. *Physics of Fluids*, 26(2):024103, 2014.

[357] J. N. Juang. *Applied System Identification*. Prentice Hall, 1994.

[358] J. N. Juang and R. S. Pappa. An eigensystem realization algorithm for modal parameter identification and model reduction. *Journal of Guidance, Control, and Dynamics*, 8(5):620–627, 1985.

[359] J. N. Juang, M. Phan, L. G. Horta, and R. W. Longman. Identification of observer/Kalman filter Markov parameters: Theory and experiments. Technical Memorandum 104069, NASA, 1991.

[360] S. J. Julier and J. K. Uhlmann. A new extension of the Kalman filter to nonlinear systems. In *International Symposium on Aerospace/Defense Sensing, Simulation and Controls*, volume 3, pages 182–193, 1997.

[361] S. J. Julier and J. K. Uhlmann. Unscented filtering and nonlinear estimation. *Proceedings of the IEEE*, 92(3):401–422, 2004.

[362] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[363] K. Kaheman, E. Kaiser, B. Strom, J. N. Kutz, and S. L. Brunton. Learning discrepancy models from experimental data. Preprint arXiv:1909.08574, 2019.

[364] K. Kaheman, J. N. Kutz, and S. L. Brunton. SINDy-PI: a robust algorithm for parallel implicit sparse identification of nonlinear dynamics. *Proceedings of the Royal Society A*, 476(2242):20200279, 2020.

[365] E. Kaiser, J. N. Kutz, and S. L. Brunton. Data-driven discovery of Koopman eigenfunctions for control. *Machine Learning: Science and Technology*, 2(3):035023, 2021.

[366] E. Kaiser, J. N. Kutz, and S. L. Brunton. Sparse identification of nonlinear dynamics for model predictive control in the low-data limit. *Proceedings of the Royal Society A*, 474(2219):20180335, 2018.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

[367] E. Kaiser, B. R. Noack, L. Cordier, et al. Cluster-based reduced-order modelling of a mixing layer. *Journal of Fluid Mechanics*, 754:365–414, 2014.

[368] S. M. Kakade. A natural policy gradient. *Advances in Neural Information Processing Systems, 14*, 8pp., 2001.

[369] M. Kalia, S. L. Brunton, H. G. Meijer, C. Brune, and J. N. Kutz. Learning normal form autoencoders for data-driven discovery of universal, parameter-dependent governing equations. Preprint arXiv:2106.05102, 2021.

[370] R. E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Fluids Engineering*, 82(1):35–45, 1960.

[371] M. Kamb, E. Kaiser, S. L. Brunton, and J. N. Kutz. Time-delay observables for Koopman: Theory and applications. *SIAM Journal on Applied Dynamical Systems*, 19(2):886–917, 2020.

[372] A. A. Kaptanoglu, J. L. Callaham, C. J. Hansen, A. Aravkin, and S. L. Brunton. Promoting global stability in data-driven models of quadratic nonlinear dynamics. *Physical Review Fluids*, 6:094401, 2021.

[373] A. A. Kaptanoglu, K. D. Morgan, C. J. Hansen, and S. L. Brunton. Physics-constrained, low-dimensional models for MHD: First-principles and data-driven approaches. *Physical Review E*, 104:015206, 2021.

[374] K. Karhunen. Über lineare Methoden in der Wahrscheinlichkeitsrechnung. Dissertation, Helsinki. *Annales Academiæ Scientiarum Fennicæ, A. I*, 37, 1947.

[375] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, et al. Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440, 2021.

[376] K. Kasper, L. Mathelin, and H. Abou-Kandil. A machine learning approach for constrained sensor placement. In *American Control Conference, 2015*, pages 4479–4484. IEEE, 2015.

[377] A. K. Kassam and L. N. Trefethen. Fourth-order time-stepping for stiff PDEs. *SIAM Journal on Scientific Computing*, 26(4):1214–1233, 2005.

[378] M. Kearns and L. Valiant. Cryptographic limitations on learning Boolean formulae and finite automata. *Journal of the ACM*, 41(1):67–95, 1994.

[379] A. R. Kellems, S. Chaturantabut, D. C. Sorensen, and S. J. Cox. Morphologically accurate reduced order modeling of spiking neurons. *Journal of Computational Neuroscience*, 28(3):477–494, 2010.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

[380] J. Kepler. *Tabulae Rudolphinae, quibus astronomicae scientiae, temporum longinquitate collapsae restauratio continentur*. Jonas Saur, 1627.

[381] G. Kerschen and J.-C. Golinval. Physical interpretation of the proper orthogonal modes using the singular value decomposition. *Journal of Sound and Vibration*, 249(5):849–865, 2002.

[382] G. Kerschen, J.-C. Golinval, A. F. Vakakis, and L. A. Bergman. The method of proper orthogonal decomposition for dynamical characterization and order reduction of mechanical systems: An overview. *Nonlinear Dynamics*, 41(1–3):147–169, 2005.

[383] I. G. Kevrekidis, C. W. Gear, J. M. Hyman, et al. Equation-free, coarse-grained multiscale computation: Enabling microscopic simulators to perform system-level analysis. *Communications in Mathematical Sciences*, 1(4):715–762, 2003.

[384] N. J. Killingsworth and M. Krstc. PID tuning using extremum seeking: Online, model-free performance optimization. *IEEE Control Systems Magazine*, 26(1):70–79, 2006.

[385] H. J. Kim, M. I. Jordan, S. Sastry, and A. Y. Ng. Autonomous helicopter flight via reinforcement learning. In *Advances in Neural Information Processing Systems, 16*, pages 799–806, 2003.

[386] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. Preprint arXiv:1412.6980, 2014.

[387] D. P. Kingma and M. Welling. Auto-encoding variational Bayes. Preprint arXiv:1312.6114, 2013.

[388] M. Kirby and L. Sirovich. Application of the Karhunen–Loève procedure for the characterization of human faces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(1):103–108, 1990.

[389] V. C. Klema and A. J. Laub. The singular value decomposition: Its computation and some applications. *IEEE Transactions on Automatic Control*, 25(2):164–176, 1980.

[390] S. Klus, P. Gelß, S. Peitz, and C. Schütte. Tensor-based dynamic mode decomposition. *Nonlinearity*, 31(7):3359, 2018.

[391] S. Klus, F. Nüske, and B. Hamzi. Kernel-based approximation of the Koopman generator and Schrödinger operator. *Entropy*, 22(7):722, 2020.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

[392] S. Klus, F. Nüske, P. Koltai, et al. Data-driven model reduction and transfer operator approximation. *Journal of Nonlinear Science*, 28:985–1010, 2018.

[393] S. Klus, I. Schuster, and K. Muandet. Eigendecompositions of transfer operators in reproducing kernel Hilbert spaces. *Journal of Nonlinear Science*, 30(1):283–315, 2020.

[394] J. Kober and J. Peters. Reinforcement learning in robotics: A survey. In *Reinforcement Learning*, pages 579–610. Springer, 2012.

[395] R. Koch. *The 80/20 Principle*. Nicholas Brealey, 1997.

[396] R. Koch. *Living the 80/20 Way*. Audio-Tech Business Book Summaries, 2006.

[397] R. Koch. *The 80/20 Principle: The Secret to Achieving More with Less*. Crown Business, 2011.

[398] R. Koch. *The 80/20 Principle and 92 Other Powerful Laws of Nature: The Science of Success*. Nicholas Brealey, 2013.

[399] D. Kochkov, J. A. Smith, A. Alieva, et al. Machine learning accelerated computational fluid dynamics. Preprint arXiv:2102.01010, 2021.

[400] T. Kohonen. The self-organizing map. *Neurocomputing*, 21(1–3):1–6, 1998.

[401] T. G. Kolda and B. W. Bader. Tensor decompositions and applications. *SIAM Review*, 51(3):455–500, 2009.

[402] B. O. Koopman. Hamiltonian systems and transformation in Hilbert space. *Proceedings of the National Academy of Sciences, USA*, 17(5):315–318, 1931.

[403] B. O. Koopman and J. v. Neumann. Dynamical systems of continuous spectra. *Proceedings of the National Academy of Sciences, USA*, 18(3):255, 1932.

[404] M. Korda and I. Mezić. Linear predictors for nonlinear dynamical systems: Koopman operator meets model predictive control. *Automatica*, 93(149–160), 2018.

[405] M. Korda and I. Mezić. On convergence of extended dynamic mode decomposition to the Koopman operator. *Journal of Nonlinear Science*, 28(2):687–710, 2018.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

[406] P. Koumoutsakos, J. Freund, and D. Parekh. Evolution strategies for automatic optimization of jet mixing. *AIAA Journal*, 39(5):967–969, 2001.

[407] N. Kovachki, Z. Li, B. Liu, et al. Neural operator: Learning maps between function spaces. Preprint arXiv:2108.08481, 2021.

[408] K. Kowalski, W.-H. Steeb, and K. Kowalksi. *Nonlinear Dynamical Systems and Carleman Linearization*. World Scientific, 1991.

[409] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[410] J. R. Koza, F. H. Bennett III, and O. Stiffelman. Genetic programming as a Darwinian invention machine. In *Genetic Programming*, pages 93–108. Springer, 1999.

[411] B. Kramer, P. Grover, P. Boufounos, M. Benosman, and S. Nabi. Sparse sensing and DMD based identification of flow regimes and bifurcations in complex flows. *SIAM Journal on Applied Dynamical Systems*, 16(2):1164–1196, 2017.

[412] J. P. Krieger and M. Krstic. Extremum seeking based on atmospheric turbulence for aircraft endurance. *Journal of Guidance, Control, and Dynamics*, 34(6):1876–1885, 2011.

[413] A. Krishnapriyan, A. Gholami, S. Zhe, R. Kirby, and M. W. Mahoney. Characterizing possible failure modes in physics-informed neural networks. Advances in Neural Information Processing Systems, 34, 2021.

[414] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems, 25*, pages 1097–1105, 2012.

[415] M. Krstić and H. Wang. Stability of extremum seeking feedback for general nonlinear dynamic systems. *Automatica*, 36:595–601, 2000.

[416] M. Krstić, A. Krupadanam, and C. Jacobson. Self-tuning control of a nonlinear model of combustion instabilities. *IEEE Transactions on Control Systems Technology*, 7(4):424–436, 1999.

[417] T. D. Kulkarni, W. F. Whitney, P. Kohli, and J. Tenenbaum. Deep convolutional inverse graphics network. In *Advances in Neural Information Processing Systems, 28*, pages 2539–2547, 2015.

[418] S. Kullback and R. A. Leibler. On information and sufficiency. *Annals of Mathematical Statistics*, 22(1):79–86, 1951.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

[419] K. Kunisch and S. Volkwein. Optimal snapshot location for computing POD basis functions. *ESAIM: Mathematical Modelling and Numerical Analysis*, 44(3):509–529, 2010.

[420] J. N. Kutz. *Data-Driven Modeling and Scientific Computation: Methods for Complex Systems and Big Data*. Oxford University Press, 2013.

[421] J. N. Kutz. Deep learning in fluid dynamics. *Journal of Fluid Mechanics*, 814:1–4, 2017.

[422] J. N. Kutz, S. L. Brunton, B. W. Brunton, and J. L. Proctor. *Dynamic Mode Decomposition: Data-Driven Modeling of Complex Systems*. SIAM, 2016.

[423] J. N. Kutz, X. Fu, and S. L. Brunton. Multi-resolution dynamic mode decomposition. *SIAM Journal on Applied Dynamical Systems*, 15(2):713–735, 2016.

[424] J. N. Kutz, S. Sargsyan, and S. L. Brunton. Leveraging sparsity and compressive sensing for reduced order modeling. In *Model Reduction of Parametrized Systems*, pages 301–315. Springer, 2017.

[425] S. Lall, J. E. Marsden, and S. Glavaški. Empirical model reduction of controlled nonlinear systems. In *IFAC World Congress*, volume F, pages 473–478. International Federation of Automatic Control, 1999.

[426] S. Lall, J. E. Marsden, and S. Glavaški. A subspace approach to balanced truncation for model reduction of nonlinear control systems. *International Journal of Robust and Nonlinear Control*, 12(6):519–535, 2002.

[427] Y. Lan and I. Mezić. Linearization in the large of nonlinear systems and Koopman operator spectrum. *Physica D: Nonlinear Phenomena*, 242(1):42–53, 2013.

[428] H. Lange, S. L. Brunton, and J. N. Kutz. From Fourier to Koopman: Spectral methods for long-term time series prediction. *Journal of Machine Learning Research*, 22(41):1–38, 2021.

[429] S. Lanka and T. Wu. ARCHER: Aggressive rewards to counter bias in hindsight experience replay. Preprint arXiv:1809.02070, 2018.

[430] A. Laub. A Schur method for solving algebraic Riccati equations. *IEEE Transactions on Automatic Control*, 24(6):913–921, 1979.

[431] H. Le, C. Voloshin, and Y. Yue. Batch policy learning under constraints. In *International Conference on Machine Learning*, pages 3703–3712. PMLR, 2019.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

[432] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436, 2015.

[433] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[434] J. H. Lee. Model predictive control: Review of the three decades of development. *International Journal of Control, Automation and Systems*, 9(3):415–424, 2011.

[435] K. Lee, J. Ho, and D. Kriegman. Acquiring linear subspaces for face recognition under variable lighting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(5):684–698, 2005.

[436] A. M. Legendre. *Nouvelles méthodes pour la détermination des orbites des comètes*. Didot, 1805.

[437] V. Lenaerts, G. Kerschen, and J.-C. Golinval. Proper orthogonal decomposition for model updating of non-linear mechanical systems. *Mechanical Systems and Signal Processing*, 15(1):31–43, 2001.

[438] I. Lenz, R. A. Knepper, and A. Saxena. DeepMPC: Learning deep latent features for model predictive control. In *Robotics: Science and Systems*, 9pp., 2015.

[439] R. Leyva, C. Alonso, I. Queinnec, et al. MPPT of photovoltaic systems using extremum-seeking control. *IEEE Transactions on Aerospace and Electronic Systems*, 42(1):249–258, 2006.

[440] Q. Li, F. Dietrich, E. M. Bollt, and I. G. Kevrekidis. Extended dynamic mode decomposition with dictionary learning: A data-driven adaptive spectral decomposition of the Koopman operator. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 27(10):103111, 2017.

[441] Z. Li, N. Kovachki, K. Azizzadenesheli, et al. Fourier neural operator for parametric partial differential equations. Preprint arXiv:2010.08895, 2020.

[442] Z. Li, N. Kovachki, K. Azizzadenesheli, et al. Multipole graph neural operator for parametric partial differential equations. Preprint arXiv:2006.09535, 2020.

[443] Z. Li, N. Kovachki, K. Azizzadenesheli, et al. Neural operator: Graph kernel network for partial differential equations. Preprint arXiv:2003.03485, 2020.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

[444] Y. Liang, H. Lee, S. Lim, et al. Proper orthogonal decomposition and its applications – Part I: Theory. *Journal of Sound and Vibration*, 252(3):527–544, 2002.

[445] E. Liberty. Simple and deterministic matrix sketching. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 581–588. ACM, 2013.

[446] E. Liberty, F. Woolfe, P.-G. Martinsson, V. Rokhlin, and M. Tygert. Randomized algorithms for the low-rank approximation of matrices. *Proceedings of the National Academy of Sciences, USA*, 104:20 167–20 172, 2007.

[447] Z. Lin, M. Chen, and Y. Ma. The augmented Lagrange multiplier method for exact recovery of corrupted low-rank matrices. Preprint arXiv:1009.5055, 2010.

[448] J. Ling, A. Kurzawski, and J. Templeton. Reynolds averaged turbulence modelling using deep neural networks with embedded invariance. *Journal of Fluid Mechanics*, 807:155–166, 2016.

[449] Y. Liu, J. N. Kutz, and S. L. Brunton. Hierarchical deep learning of multiscale differential equation time-steppers. Preprint arXiv:2008.09768, 2020.

[450] Y. Liu, C. Ponce, S. L. Brunton, and J. N. Kutz. Multiresolution convolutional autoencoders. Preprint arXiv:2004.04946, 2020.

[451] L. Ljung. *System Identification: Theory for the User*. Prentice Hall, 1999.

[452] S. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.

[453] M. Loeve. *Probability Theory*. Van Nostrand, 1955.

[454] J.-C. Loiseau. Data-driven modeling of the chaotic thermal convection in an annular thermosyphon. *Theoretical and Computational Fluid Dynamics*, 34(4):339–365, 2020.

[455] J.-C. Loiseau and S. L. Brunton. Constrained sparse Galerkin regression. *Journal of Fluid Mechanics*, 838:42–67, 2018.

[456] J.-C. Loiseau, B. R. Noack, and S. L. Brunton. Sparse reduced-order modeling: Sensor-based dynamics to full-state estimation. *Journal of Fluid Mechanics*, 844:459–490, 2018.

[457] R. W. Longman. Iterative learning control and repetitive control for engineering practice. *International Journal of Control*, 73(10):930–954, 2000.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

[458] B. T. Lopez, J.-J. E. Slotine, and J. P. How. Dynamic tube MPC for nonlinear systems. In *2019 American Control Conference*, pages 1655–1662. IEEE, 2019.

[459] E. N. Lorenz. Empirical orthogonal functions and statistical weather prediction. Technical Report, Massachusetts Institute of Technology, 1956.

[460] E. N. Lorenz. Deterministic nonperiodic flow. *Journal of the Atmospheric Sciences*, 20(2):130–141, 1963.

[461] L. Lu, P. Jin, G. Pang, Z. Zhang, and G. E. Karniadakis. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, 2021.

[462] D. M. Luchtenburg and C. W. Rowley. Model reduction using snapshot-based realizations. *Bulletin of the American Physical Society*, 56(18), Abstract, 2011 (64th Annual Meeting of the APS Division of Fluid Dynamics).

[463] J. Lumley. Toward a turbulent constitutive relation. *Journal of Fluid Mechanics*, 41(2):413–434, 1970.

[464] B. Lusch, E. C. Chi, and J. N. Kutz. Shape constrained tensor decompositions using sparse representations in over-complete libraries. Preprint arXiv:1608.04674, 2016.

[465] B. Lusch, J. N. Kutz, and S. L. Brunton. Deep learning for universal linear embeddings of nonlinear dynamics. *Nature Communications*, 9(1):4950, 2018.

[466] F. Lusseyran, F. Gueniat, J. Basley, et al. Flow coherent structures and frequency signature: Application of the dynamic modes decomposition to open cavity flow. *Journal of Physics: Conference Series*, 318: 042036, 2011.

[467] J. Lynch, P. Aughwane, and T. M. Hammond. Video games and surgical ability: A literature review. *Journal of Surgical Education*, 67(3):184–189, 2010.

[468] Z. Ma, S. Ahuja, and C. W. Rowley. Reduced order models for control of fluids using the eigensystem realization algorithm. *Theoretical and Computational Fluid Dynamics*, 25(1):233–247, 2011.

[469] W. Maass, T. Natschläger, and H. Markram. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural Computation*, 14(11):2531–2560, 2002.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

[470] A. Mackey, H. Schaeffer, and S. Osher. On the compressive spectral method. *Multiscale Modeling and Simulation*, 12(4):1800–1827, 2014.

[471] M. W. Mahoney. Randomized algorithms for matrices and data. *Foundations and Trends in Machine Learning*, 3:123–224, 2011.

[472] A. J. Majda and J. Harlim. Physics constrained nonlinear regression models for time series. *Nonlinearity*, 26(1):201, 2012.

[473] A. J. Majda and Y. Lee. Conceptual dynamical models for turbulence. *Proceedings of the National Academy of Sciences, USA*, 111(18):6548–6553, 2014.

[474] S. G. Mallat. A theory for multiresolution signal decomposition: The wavelet representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(7):674–693, 1989.

[475] S. G. Mallat. *A Wavelet Tour of Signal Processing*. Academic Press, 1999.

[476] S. G. Mallat. Understanding deep convolutional networks. *Philosophical Transactions of the Royal Society A*, 374(2065):20150203, 2016.

[477] J. Mandel. Use of the singular value decomposition in regression analysis. *American Statistician*, 36(1):15–24, 1982.

[478] N. M. Mangan, S. L. Brunton, J. L. Proctor, and J. N. Kutz. Inferring biological networks by sparse identification of nonlinear dynamics. *IEEE Transactions on Molecular, Biological, and Multi-Scale Communications*, 2(1):52–63, 2016.

[479] N. M. Mangan, J. N. Kutz, S. L. Brunton, and J. L. Proctor. Model selection for dynamical systems via sparse regression and information criteria. *Proceedings of the Royal Society A*, 473(2204):1–16, 2017.

[480] J. Mann and J. N. Kutz. Dynamic mode decomposition for financial trading strategies. *Quantitative Finance*, 16(11):1643–1655, 2016.

[481] K. Manohar, B. W. Brunton, J. N. Kutz, and S. L. Brunton. Data-driven sparse sensor placement for reconstruction: Demonstrating the benefits of exploiting known patterns. *IEEE Control Systems Magazine*, 38(3):63–86, 2018.

[482] K. Manohar, S. L. Brunton, and J. N. Kutz. Environmental identification in flight using sparse approximation of wing strain. *Journal of Fluids and Structures*, 70:162–180, 2017.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

[483] K. Manohar, E. Kaiser, S. L. Brunton, and J. N. Kutz. Optimized sampling for multiscale dynamics. *SIAM Multiscale Modeling and Simulation*, 17(1):117–136, 2019.

[484] K. Manohar, J. N. Kutz, and S. L. Brunton. Optimized sensor and actuator placement for balanced models. Preprint arXiv:1812.01574, 2018.

[485] A. Mardt, L. Pasquali, H. Wu, and F. Noé. VAMPnets for deep learning of molecular kinetics. *Nature Communications*, 9:5, 2018.

[486] J. E. Marsden and T. S. Ratiu. *Introduction to Mechanics and Symmetry*, 2nd edition. Springer, 1999.

[487] P.-G. Martinsson. Randomized methods for matrix computations and analysis of high dimensional data. Preprint arXiv:1607.01649, 2016.

[488] P.-G. Martinsson, V. Rokhlin, and M. Tygert. A randomized algorithm for the decomposition of matrices. *Applied and Computational Harmonic Analysis*, 30:47–68, 2011.

[489] J. L. Maryak, J. C. Spall, and B. D. Heydon. Use of the Kalman filter for inference in state-space models with unknown noise distributions. *IEEE Transactions on Automatic Control*, 49(1):87–90, 2004.

[490] L. Massa, R. Kumar, and P. Ravindran. Dynamic mode decomposition analysis of detonation waves. *Physics of Fluids*, 24(6):066101, 2012.

[491] L. Mathelin, K. Kasper, and H. Abou-Kandil. Observable dictionary learning for high-dimensional statistical inference. *Archives of Computational Methods in Engineering*, 25(1):103–120, 2018.

[492] R. Maulik, O. San, A. Rasheed, and P. Vedula. Subgrid modelling for two-dimensional turbulence using neural networks. *Journal of Fluid Mechanics*, 858:122–144, 2019.

[493] R. Maury, M. Keonig, L. Cattafesta, P. Jordan, and J. Delville. Extremum-seeking control of jet noise. *Aeroacoustics*, 11(3–4):459–474, 2012.

[494] S. F. McCormick. *Multigrid Methods*. SIAM, 1987.

[495] B. J. McKeon and A. S. Sharma. A critical-layer framework for turbulent pipe flow. *Journal of Fluid Mechanics*, 658:336–382, 2010.

[496] X. Meng, Z. Li, D. Zhang, and G. E. Karniadakis. PPINN: Parareal physics-informed neural network for time-dependent PDEs. *Computer Methods in Applied Mechanics and Engineering*, 370:113250, 2020.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

[497] I. Mezić. Spectral properties of dynamical systems, model reduction and decompositions. *Nonlinear Dynamics*, 41(1–3):309–325, 2005.

[498] I. Mezić. Analysis of fluid flows via spectral properties of the Koopman operator. *Annual Review of Fluid Mechanics*, 45:357–378, 2013.

[499] I. Mezić. *Spectral Operator Methods in Dynamical Systems: Theory and Applications*. Springer, 2017.

[500] I. Mezić and A. Banaszuk. Comparison of systems with complex behavior. *Physica D: Nonlinear Phenomena*, 197(1):101–133, 2004.

[501] I. Mezić and S. Wiggins. A method for visualization of invariant sets of dynamical systems based on the ergodic partition. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 9(1):213–218, 1999.

[502] M. Milano and P. Koumoutsakos. Neural network modeling for near wall turbulent flow. *Journal of Computational Physics*, 182(1):1–26, 2002.

[503] M. Minsky. Steps toward artificial intelligence. *Proceedings of the IRE*, 49(1):8–30, 1961.

[504] T. M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[505] Y. Mizuno, D. Duke, C. Atkinson, and J. Soria. Investigation of wall-bounded turbulent flow using dynamic mode decomposition. *Journal of Physics: Conference Series*, 318:042040, 2011.

[506] V. Mnih, K. Kavukcuoglu, D. Silver, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[507] J. P. Moeck, J.-F. Bourgouin, D. Durox, T. Schuller, and S. Candel. Tomographic reconstruction of heat release rate perturbations induced by helical modes in turbulent swirl flames. *Experiments in Fluids*, 54(4):1–17, 2013.

[508] P. R. Montague, P. Dayan, and T. J. Sejnowski. A framework for mesencephalic dopamine systems based on predictive Hebbian learning. *Journal of Neuroscience*, 16(5):1936–1947, 1996.

[509] B. C. Moore. Principal component analysis in linear systems: Controllability, observability, and model reduction. *IEEE Transactions on Automatic Control*, 26(1):17–32, 1981.

[510] C. C. Moore. Ergodic theorem, ergodic theory, and statistical mechanics. *Proceedings of the National Academy of Sciences, USA*, 112(7):1907–1911, 2015.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

[511] K. L. Moore. *Iterative Learning Control for Deterministic Systems*. Springer, 2012.

[512] M. Morari and J. H. Lee. Model predictive control: Past, present and future. *Computers and Chemical Engineering*, 23(4):667–682, 1999.

[513] J. Morton, A. Jameson, M. J. Kochenderfer, and F. D. Witherden. Deep dynamical modeling and control of unsteady fluid flows. In *Advances in Neural Information Processing Systems, 31*, 11pp., 2018.

[514] T. W. Muld, G. Efraimsson, and D. S. Henningson. Flow structures around a high-speed train extracted using proper orthogonal decomposition and dynamic mode decomposition. *Computers and Fluids*, 57:87–97, 2012.

[515] T. W. Muld, G. Efraimsson, and D. S. Henningson. Mode decomposition on surface-mounted cube. *Flow, Turbulence and Combustion*, 88(3):279–310, 2012.

[516] S. Müller, M. Milano, and P. Koumoutsakos. Application of machine learning algorithms to flow modeling and optimization. *Annual Research Briefs*, pages 169–178, Center for Turbulence Research, Stanford University, 1999.

[517] I. Munteanu, A. I. Bratcu, and E. Ceanga. Wind turbulence used as searching signal for MPPT in variable-speed wind energy conversion systems. *Renewable Energy*, 34(1):322–327, 2009.

[518] K. P. Murphy. *Machine Learning: A Probabilistic Perspective*. MIT Press, 2012.

[519] V. Nair and G. E. Hinton. Rectified linear units improve restricted Boltzmann machines. In *Proceedings of the 27th international Conference on Machine Learning*, pages 807–814, 2010.

[520] D. Needell and J. A. Tropp. CoSaMP: Iterative signal recovery from incomplete and inaccurate samples. *Communications of the ACM*, 53(12):93–100, 2010.

[521] J. v. Neumann. Proof of the quasi-ergodic hypothesis. *Proceedings of the National Academy of Sciences, USA*, 18(1):70–82, 1932.

[522] N. Nguyen, A. Patera, and J. Peraire. A best points interpolation method for efficient approximation of parametrized functions. *International Journal for Numerical Methods in Engineering*, 73(4):521–543, 2008.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

[523] Y. Nievergelt. *Wavelets Made Easy*, volume 174 of Modern Birkhäuser Classics. Springer, 1999.

[524] B. R. Noack, K. Afanasiev, M. Morzynski, G. Tadmor, and F. Thiele. A hierarchy of low-dimensional models for the transient and post-transient cylinder wake. *Journal of Fluid Mechanics*, 497:335–363, 2003.

[525] B. R. Noack, T. Duriez, L. Cordier, et al. Closed-loop turbulence control with machine learning methods. *Bulletin of the American Physical Society*, 58(18):M25.0009, page 418, 2013.

[526] B. R. Noack, M. Morzynski, and G. Tadmor. *Reduced-Order Modelling for Flow Control*, volume 528 of CISM Courses and Lectures. Springer, 2011.

[527] B. R. Noack, W. Stankiewicz, M. Morzynski, and P. J. Schmid. Recursive dynamic mode decomposition of a transient cylinder wake. *Journal of Fluid Mechanics*, 809:843–872, 2016.

[528] F. Noé and F. Nuske. A variational approach to modeling slow processes in stochastic dynamical systems. *Multiscale Modeling and Simulation*, 11(2):635–655, 2013.

[529] E. Noether. Invariante variationsprobleme. *Nachrichten von der Gesellschaft der Wissenschaften zu Göttingen, Mathematisch-Physikalische Klasse*, 235–257, 1918. (Engl. transl.: arXiv:physics/0503066; http://dx.doi.org/10.1080/00411457108231446).

[530] G. Novati, H. L. de Laroussilhe, and P. Koumoutsakos. Automating turbulence modelling by multi-agent reinforcement learning. *Nature Machine Intelligence*, 3(1):87–96, 2021.

[531] G. Novati, L. Mahadevan, and P. Koumoutsakos. Controlled gliding and perching through deep-reinforcement-learning. *Physical Review Fluids*, 4(9):093902, 2019.

[532] G. Novati, S. Verma, D. Alexeev, et al. Synchronisation through learning for two self-propelled swimmers. *Bioinspiration and Biomimetics*, 12(3):aa6311, 2017.

[533] F. Nüske, P. Gelß, S. Klus, and C. Clementi. Tensor based EDMD for the Koopman analysis of high-dimensional systems. Preprint arXiv:1908.04741, 2019.

[534] F. Nüske, B. G. Keller, G. Pérez-Hernández, A. S. Mey, and F. Noé. Variational approach to molecular kinetics. *Journal of Chemical Theory and Computation*, 10(4):1739–1752, 2014.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

[535] F. Nüske, R. Schneider, F. Vitalini, and F. Noé. Variational tensor approach for approximating the rare-event kinetics of macromolecular systems. *Journal of Chemical Physics*, 144(5):054105, 2016.

[536] H. Nyquist. Certain topics in telegraph transmission theory. *Transactions of the AIEE*, 47(2):617–644, 1928.

[537] G. Obinata and B. D. Anderson. *Model Reduction for Control System Design*. Springer, 2012.

[538] M. Ornik, A. Israel, and U. Topcu. Control-oriented learning on the fly. Preprint arXiv:1709.04889, 2017.

[539] C. M. Ostoich, D. J. Bodony, and P. H. Geubelle. Interaction of a Mach 2.25 turbulent boundary layer with a fluttering panel using direct numerical simulation. *Physics of Fluids*, 25(11):110806, 2013.

[540] S. E. Otto and C. W. Rowley. Linearly-recurrent autoencoder networks for learning dynamics. SIAM Journal on Applied Dynamical Systems, 18(1):558–593, 2019.

[541] Y. Ou, C. Xu, E. Schuster, et al. Design and simulation of extremum-seeking open-loop optimal control of current profile in the DIII-D tokamak. *Plasma Physics and Controlled Fusion*, 50:115001 (24pp.), 2008.

[542] V. Ozoliņš, R. Lai, R. Caflisch, and S. Osher. Compressed modes for variational problems in mathematics and physics. *Proceedings of the National Academy of Sciences, USA*, 110(46):18 368–18 373, 2013.

[543] C. Pan, D. Yu, and J. Wang. Dynamical mode decomposition of Gurney flap wake flow. *Theoretical and Applied Mechanics Letters*, 1(1):012002, 2011.

[544] S. Pan and K. Duraisamy. Physics-informed probabilistic learning of linear embeddings of nonlinear dynamics with guaranteed stability. *SIAM Journal on Applied Dynamical Systems*, 19(1):480–509, 2020.

[545] X. Pan, Y. You, Z. Wang, and C. Lu. Virtual to real reinforcement learning for autonomous driving. Preprint arXiv:1704.03952, 2017.

[546] V. Parezanović, T. Duriez, L. Cordier, et al. Closed-loop control of an experimental mixing layer using machine learning control. Preprint arXiv:1408.3259, 2014.

[547] V. Parezanović, J.-C. Laurentie, T. Duriez, et al. Mixing layer manipulation experiment – from periodic forcing to machine learning closed-loop control. *Journal of Flow Turbulence and Combustion*, 94(1):155–173, 2015.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

[548] E. J. Parish and K. T. Carlberg. Time-series machine-learning error models for approximate solutions to parameterized dynamical systems. *Computer Methods in Applied Mechanics and Engineering*, 365:112990, 2020.

[549] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell. Curiosity-driven exploration by self-supervised prediction. In *International Conference on Machine Learning*, pages 2778–2787. PMLR, 2017.

[550] J. Pathak, Z. Lu, B. R. Hunt, M. Girvan, and E. Ott. Using machine learning to replicate chaotic attractors and calculate Lyapunov exponents from data. Chaos: An Interdisciplinary Journal of Nonlinear Science, 27(12):121102, 2017.

[551] P. I. Pavlov. *Conditioned Reflexes: An Investigation of the Physiological Activity of the Cerebral Cortex*. Oxford University Press, 1927.

[552] K. Pearson. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, 2(7–12):559–572, 1901.

[553] B. Peherstorfer and K. Willcox. Detecting and adapting to parameter changes for reduced models of dynamic data-driven application systems. *Procedia Computer Science*, 51:2553–2562, 2015.

[554] B. Peherstorfer and K. Willcox. Dynamic data-driven reduced-order models. *Computer Methods in Applied Mechanics and Engineering*, 291:21–41, 2015.

[555] B. Peherstorfer and K. Willcox. Online adaptive model reduction for nonlinear systems via low-rank updates. *SIAM Journal on Scientific Computing*, 37(4):A2123–A2150, 2015.

[556] B. Peherstorfer, D. Butnaru, K. Willcox, and H.-J. Bungartz. Localized discrete empirical interpolation method. *SIAM Journal on Scientific Computing*, 36(1):A168–A192, 2014.

[557] B. Peherstorfer, Z. Drmac, and S. Gugercin. Stability of discrete empirical interpolation and gappy proper orthogonal decomposition with randomized and deterministic sampling points. *SIAM Journal on Scientific Computing*, 42(5):A2837–A2864, 2020.

[558] S. Peitz and S. Klus. Koopman operator-based model reduction for switched-system control of PDEs. Preprint arXiv:1710.06759, 2017.

[559] S. D. Pendergrass, J. N. Kutz, and S. L. Brunton. Streaming GPU singular value and dynamic mode decompositions. Preprint arXiv:1612.07875, 2016.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

[560] R. Penrose. A generalized inverse for matrices. *Mathematical Proceedings of the Cambridge Philosophical Society*, 51:406–413, 1955.

[561] R. Penrose and J. A. Todd. On best approximate solutions of linear matrix equations. *Mathematical Proceedings of the Cambridge Philosophical Society*, 52:17–19, 1956.

[562] L. Perko. *Differential Equations and Dynamical Systems*, volume 7 of Texts in Applied Mathematics. Springer, 2013.

[563] M. Phan, L. G. Horta, J. N. Juang, and R. W. Longman. Linear system identification via an asymptotically stable observer. *Journal of Optimization Theory and Applications*, 79:59–86, 1993.

[564] M. A. Pinsky. *Introduction to Fourier Analysis and Wavelets*, volume 102 of Graduate Studies in Mathematics. American Mathematical Society, 2002.

[565] C. Pivot, L. Mathelin, L. Cordier, F. Guéniat, and B. R. Noack. A continuous reinforcement learning strategy for closed-loop control in fluid dynamics. In *35th AIAA Applied Aerodynamics Conference*, page 3566. AIAA, 2017.

[566] T. Poggio. Deep learning: mathematics and neuroscience. *Views & Reviews, McGovern Center for Brains, Minds and Machines*, 7pp., 2016.

[567] P. Poncet, G.-H. Cottet, and P. Koumoutsakos. Control of three-dimensional wakes using evolution strategies. *Comptes Rendus Mécanique*, 333(1):65–77, 2005.

[568] J. L. Proctor and P. A. Eckhoff. Discovering dynamic patterns from infectious disease data using dynamic mode decomposition. *International Health*, 7(2):139–145, 2015.

[569] J. L. Proctor, S. L. Brunton, B. W. Brunton, and J. N. Kutz. Exploiting sparsity and equation-free architectures in complex systems. *European Physical Journal Special Topics*, 223(13):2665–2684, 2014.

[570] J. L. Proctor, S. L. Brunton, and J. N. Kutz. Dynamic mode decomposition with control. *SIAM Journal on Applied Dynamical Systems*, 15(1):142–161, 2016.

[571] H. Qi and S. M. Hughes. Invariance of principal components under low-dimensional random projection of the data. *19th IEEE International Conference on Image Processing*. IEEE, 2012.

[572] S. Qian and D. Chen. Discrete Gabor transform. *IEEE Transactions on Signal Processing*, 41(7):2429–2438, 1993.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

[573] S. J. Qin and T. A. Badgwell. An overview of industrial model predictive control technology. *Chemical Process Control – 5*, volume 93 of AIChE Symposium Series, pages 232–256. AIChE, 1997.

[574] S. J. Qin and T. A. Badgwell. A survey of industrial model predictive control technology. *Control Engineering Practice*, 11(7):733–764, 2003.

[575] T. Qin, K. Wu, and D. Xiu. Data driven governing equations approximation using deep neural networks. *Journal of Computational Physics*, 395:620–635, 2019.

[576] Q. Qu, J. Sun, and J. Wright. Finding a sparse vector in a subspace: Linear sparsity using alternating directions. In *Advances in Neural Information Processing Systems, 27*, pages 3401–3409, 2014.

[577] A. Quarteroni and G. Rozza. *Reduced Order Methods for Modeling and Computational Reduction*, volume 9 of Modeling, Simulation and Applications. Springer, 2013.

[578] A. Quarteroni, A. Manzoni, and F. Negri. *Reduced Basis Methods for Partial Differential Equations: An Introduction*, volume 92 of UNITEXT book series. Springer, 2015.

[579] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[580] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Elsevier, 2014.

[581] J. Rabault and A. Kuhnle. Deep reinforcement learning applied to active flow control. Preprint, 2020.

[582] J. Rabault, M. Kuchta, A. Jensen, U. Réglade, and N. Cerardi. Artificial neural networks trained through deep reinforcement learning discover control strategies for active flow control. *Journal of Fluid Mechanics*, 865:281–302, 2019.

[583] M. Raissi and G. E. Karniadakis. Hidden physics models: Machine learning of nonlinear partial differential equations. *Journal of Computational Physics*, 357:125–141, 2018.

[584] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

[585] M.'A. Ranzato, C. Poultney, S. Chopra, and Y. LeCun. Efficient learning of sparse representations with an energy-based model. In *Advances in Neural Information Processing Systems, 19*, pages 1137–1144, 2006.

[586] C. R. Rao. The utilization of multiple measurements in problems of biological classification. *Journal of the Royal Statistical Society. Series B (Methodological)*, 10(2):159–203, 1948.

[587] J. B. Rawlings. Tutorial overview of model predictive control. *IEEE Control Systems*, 20(3):38–52, 2000.

[588] S. Raychaudhuri, J. M. Stuart, and R. B. Altman. Principal components analysis to summarize microarray experiments: application to sporulation time series. In *Pacific Symposium on Biocomputing 2000*, pages 455–466. World Scientific, 2000.

[589] B. Recht. A tour of reinforcement learning: The view from continuous control. *Annual Review of Control, Robotics, and Autonomous Systems*, 2:253–279, 2019.

[590] G. Reddy, A. Celani, T. J. Sejnowski, and M. Vergassola. Learning to soar in turbulent environments. *Proceedings of the National Academy of Sciences, USA*, 113(33):E4877–E4884, 2016.

[591] G. Reddy, J. Wong-Ng, A. Celani, T. J. Sejnowski, and M. Vergassola. Glider soaring via reinforcement learning in the field. *Nature*, 562(7726):236–239, 2018.

[592] S. Reddy, A. D. Dragan, and S. Levine. Shared autonomy via deep reinforcement learning. Preprint arXiv:1802.01744, 2018.

[593] A. D. Redish. Addiction as a computational process gone awry. *Science*, 306(5703):1944–1947, 2004.

[594] W. T. Redman. On Koopman mode decomposition and tensor component analysis. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 31(5):051101, 2021.

[595] F. Regazzoni, L. Dede, and A. Quarteroni. Machine learning for fast and reliable solution of time-dependent differential equations. *Journal of Computational Physics*, 397:108852, 2019.

[596] R. H. Reichle, D. B. McLaughlin, and D. Entekhabi. Hydrologic data assimilation with the ensemble Kalman filter. *Monthly Weather Review*, 130(1):103–114, 2002.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

[597] P. A. Reinbold, D. R. Gurevich, and R. O. Grigoriev. Using noisy or incomplete data to discover models of spatiotemporal dynamics. *Physical Review E*, 101(1):010203, 2020.

[598] P. A. Reinbold, L. M. Kageorge, M. F. Schatz, and R. O. Grigoriev. Robust learning from noisy, incomplete, high-dimensional experimental data via physically constrained symbolic regression. *Nature Communications*, 12(1):1–8, 2021.

[599] B. Ren, P. Frihauf, R. J. Rafac, and M. Krstić. Laser pulse shaping via extremum seeking. *Control Engineering Practice*, 20:674–683, 2012.

[600] A. Richards and J. How. Decentralized model predictive control of cooperating UAVs. In *2004 43rd IEEE Conference on Decision and Control*, volume 4, pages 4286–4291. IEEE, 2004.

[601] A. Richards and J. P. How. Robust distributed model predictive control. *International Journal of Control*, 80(9):1517–1531, 2007.

[602] B. Ristic, S. Arulampalam, and N. J. Gordon. *Beyond the Kalman filter: Particle filters for tracking applications*. Artech House, 2004.

[603] A. J. Roberts. *Model Emergent Dynamics in Complex Systems*. SIAM, 2014.

[604] C. A. Rohde. Generalized inverses of partitioned matrices. *Journal of the Society for Industrial and Applied Mathematics*, 13(4):1033–1035, 1965.

[605] V. Rokhlin, A. Szlam, and M. Tygert. A randomized algorithm for principal component analysis. *SIAM Journal on Matrix Analysis and Applications*, 31:1100–1124, 2009.

[606] S. M. Ross. *Introduction to Stochastic Dynamic Programming*. Academic Press, 2014.

[607] J. C. Rosser, P. J. Lynch, L. Cuddihy, et al. The impact of video games on training surgeons in the 21st century. *Archives of Surgery*, 142(2):181–186, 2007.

[608] C. Rowley. Model reduction for fluids using balanced proper orthogonal decomposition. *International Journal of Bifurcation and Chaos*, 15(3):997–1013, 2005.

[609] C. W. Rowley and J. E. Marsden. Reconstruction equations and the Karhunen–Loève expansion for systems with symmetry. *Physica D: Nonlinear Phenomena*, 142(1):1–19, 2000.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

[610] C. W. Rowley, T. Colonius, and R. M. Murray. Model reduction for compressible flows using POD and Galerkin projection. *Physica D*, 189:115–129, 2004.

[611] C. W. Rowley, I. Mezić, S. Bagheri, P. Schlatter, and D. Henningson. Spectral analysis of nonlinear flows. *Journal of Fluid Mechanics*, 645:115–127, 2009.

[612] S. Roy, J.-C. Hua, W. Barnhill, G. H. Gunaratne, and J. R. Gord. Deconvolution of reacting-flow dynamics using proper orthogonal and dynamic mode decompositions. *Physical Review E*, 91(1):013001, 2015.

[613] S. H. Rudy, S. L. Brunton, J. L. Proctor, and J. N. Kutz. Data-driven discovery of partial differential equations. *Science Advances*, 3:e1602614, 2017.

[614] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[615] A. E. Sallab, M. Abdou, E. Perot, and S. Yogamani. Deep reinforcement learning framework for autonomous driving. *Electronic Imaging*, 2017(19):70–76, 2017.

[616] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–229, 1959.

[617] A. Sanchez-Gonzalez, J. Godwin, T. Pfaff, et al. Learning to simulate complex physics with graph networks. In *International Conference on Machine Learning*, pages 8459–8468. PMLR, 2020.

[618] T. P. Sapsis and A. J. Majda. Statistically accurate low-order models for uncertainty quantification in turbulent dynamical systems. *Proceedings of the National Academy of Sciences, USA*, 110(34):13 705–13 710, 2013.

[619] S. Sargsyan, S. L. Brunton, and J. N. Kutz. Nonlinear model reduction for dynamical systems using sparse sensor locations from learned libraries. *Physical Review E*, 92:033304, 2015.

[620] S. Sarkar, S. Ganguly, A. Dalal, P. Saha, and S. Chakraborty. Mixed convective flow stability of nanofluids past a square cylinder by dynamic mode decomposition. *International Journal of Heat and Fluid Flow*, 44:624–634, 2013.

[621] T. Sarlos. Improved approximation algorithms for large matrices via random projections. In *47th Annual IEEE Symposium on Foundations of Computer Science*, pages 143–152, 2006.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

[622] D. Sashidhar and J. N. Kutz. Bagging, optimized dynamic mode decomposition (BOP-DMD) for robust, stable forecasting with spatial and temporal uncertainty-quantification. Preprint arXiv:2107.10878, 2021.

[623] T. Sayadi and P. J. Schmid. Parallel data-driven decomposition algorithm for large-scale datasets: With application to transitional boundary layers. *Theoretical and Computational Fluid Dynamics*, 30:415–428, 2016.

[624] T. Sayadi, P. J. Schmid, J. W. Nichols, and P. Moin. Reduced-order representation of near-wall structures in the late transitional boundary layer. *Journal of Fluid Mechanics*, 748:278–301, 2014.

[625] S. Schaal. Is imitation learning the route to humanoid robots? *Trends in Cognitive Sciences*, 3(6):233–242, 1999.

[626] H. Schaeffer. Learning partial differential equations via data discovery and sparse optimization. *Proceedings of the Royal Society A*, 473:20160446, 2017.

[627] H. Schaeffer and S. G. McCalla. Sparse model selection via integral terms. *Physical Review E*, 96(2):023302, 2017.

[628] H. Schaeffer, R. Caflisch, C. D. Hauck, and S. Osher. Sparse dynamics for partial differential equations. *Proceedings of the National Academy of Sciences USA*, 110(17):6634–6639, 2013.

[629] R. E. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, 1990.

[630] T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay. Preprint arXiv:1511.05952, 2015.

[631] I. Scherl, B. Strom, J. K. Shang, et al. Robust principal component analysis for particle image velocimetry. *Physical Review Fluids*, 5:054401, 2020.

[632] M. Schlegel and B. R. Noack. On long-term boundedness of Galerkin models. *Journal of Fluid Mechanics*, 765:325–352, 2015.

[633] M. Schlegel, B. R. Noack, and G. Tadmor. Low-dimensional Galerkin models and control of transitional channel flow. Technical Report 01/2004, Hermann-Föttinger-Institut für Strömungsmechanik, Technische Universität Berlin, 2004.

[634] M. Schmelzer, R. P. Dwight, and P. Cinnella. Discovery of algebraic Reynolds-stress models using sparse symbolic regression. *Flow, Turbulence and Combustion*, 104(2):579–603, 2020.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

[635] P. J. Schmid. Dynamic mode decomposition for numerical and experimental data. *Journal of Fluid Mechanics*, 656:5–28, 2010.

[636] P. J. Schmid and J. Sesterhenn. Dynamic mode decomposition of numerical and experimental data. In *61st Annual Meeting of the APS Division of Fluid Dynamics*. American Physical Society, 2008.

[637] P. J. Schmid, L. Li, M. P. Juniper, and O. Pust. Applications of the dynamic mode decomposition. *Theoretical and Computational Fluid Dynamics*, 25(1–4):249–259, 2011.

[638] P. J. Schmid, D. Violato, and F. Scarano. Decomposition of time-resolved tomographic PIV. *Experiments in Fluids*, 52:1567–1579, 2012.

[639] E. Schmidt. Zur Theorie der linearen und nichtlinearen Integralgleichungen. I. Teil: Entwicklung willkürlicher Funktionen nach Systemen vorgeschriebener. *Mathematische Annalen*, 63:433–476, 1907.

[640] M. Schmidt and H. Lipson. Distilling free-form natural laws from experimental data. *Science*, 324(5923):81–85, 2009.

[641] M. D. Schmidt, R. R. Vallabhajosyula, J. W. Jenkins, et al. Automated refinement and inference of analytical models for metabolic networks. *Physical Biology*, 8(5):055011, 2011.

[642] O. T. Schmidt and T. Colonius. Guide to spectral proper orthogonal decomposition. *AIAA Journal*, 58(3):1023–1033, 2020.

[643] B. Schölkopf and A. J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, 2002.

[644] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897. PMLR, 2015.

[645] W. Schultz, P. Dayan, and P. R. Montague. A neural substrate of prediction and reward. *Science*, 275(5306):1593–1599, 1997.

[646] G. Schwarz. Estimating the dimension of a model. *Annals of Statistics*, 6(2):461–464, 1978.

[647] A. Seena and H. J. Sung. Dynamic mode decomposition of turbulent cavity flows for self-sustained oscillations. *International Journal of Heat and Fluid Flow*, 32(6):1098–1110, 2011.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

[648] E. Sejdić, I. Djurović, and J. Jiang. Time–frequency feature representation using energy concentration: An overview of recent advances. *Digital Signal Processing*, 19(1):153–183, 2009.

[649] O. Semeraro, G. Bellani, and F. Lundell. Analysis of time-resolved PIV measurements of a confined turbulent jet using POD and Koopman modes. *Experiments in Fluids*, 53(5):1203–1220, 2012.

[650] O. Semeraro, F. Lusseyran, L. Pastur, and P. Jordan. Qualitative dynamics of wavepackets in turbulent jets. *Physical Review Fluids*, 2:094605, 2017.

[651] G. Shabat, Y. Shmueli, Y. Aizenbud, and A. Averbuch. Randomized LU decomposition. *Applied and Computational Harmonic Analysis*, 44(2): 246–272, 2018.

[652] S. Shalev-Shwartz, S. Shammah, and A. Shashua. Safe, multi-agent, reinforcement learning for autonomous driving. Preprint arXiv:1610.03295, 2016.

[653] C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(3):379–423, 1948.

[654] C. E. Shannon. XXII. Programming a computer for playing chess. *London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 41(314):256–275, 1950.

[655] A. S. Sharma, I. Mezić, and B. J. McKeon. Correspondence between Koopman mode decomposition, resolvent mode decomposition, and invariant solutions of the Navier–Stokes equations. *Physical Review Fluids*, 1(3):032402, 2016.

[656] D. E. Shea, S. L. Brunton, and J. N. Kutz. SINDy-BVP: Sparse identification of nonlinear dynamics for boundary value problems. *Physical Review Research*, 3(2):023255, 2021.

[657] E. Shlizerman, E. Ding, M. O. Williams, and J. N. Kutz. The proper orthogonal decomposition for dimensionality reduction in mode-locked lasers and optical systems. *International Journal of Optics*, 2012:831604, 2011.

[658] D. Silver, A. Huang, C. J. Maddison, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[659] D. Silver, T. Hubert, J. Schrittwieser, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

[660] D. Silver, G. Lever, N. Heess, et al. Deterministic policy gradient algorithms. In *International Conference on Machine Learning*, pages 387–395. PMLR, 2014.

[661] D. Silver, J. Schrittwieser, K. Simonyan, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.

[662] V. Simoncini. A new iterative method for solving large-scale Lyapunov matrix equations. *SIAM Journal on Scientific Computing*, 29(3):1268–1288, 2007.

[663] L. Sirovich. Turbulence and the dynamics of coherent structures. I–III. *Quarterly of Applied Mathematics*, 45(3):561–590, 1987.

[664] L. Sirovich and M. Kirby. A low-dimensional procedure for the characterization of human faces. *Journal of the Optical Society of America A*, 4(3):519–524, 1987.

[665] S. Skogestad and I. Postlethwaite. *Multivariable Feedback Control*. John Wiley & Sons, 1996.

[666] P. Smolensky. Information processing in dynamical systems: Foundations of harmony theory. Technical Report, Colorado University, Boulder, Department of Computer Science, 1986.

[667] G. Solari, L. Carassale, and F. Tubino. Proper orthogonal decomposition in wind engineering. Part 1: A state-of-the-art and some prospects. *Wind and Structures*, 10(2):153–176, 2007.

[668] G. Song, F. Alizard, J.-C. Robinet, and X. Gloerfelt. Global and Koopman modes analysis of sound generation in mixing layers. *Physics of Fluids*, 25(12):124101, 2013.

[669] D. C. Sorensen and Y. Zhou. Direct methods for matrix Sylvester and Lyapunov equations. *Journal of Applied Mathematics*, 2003(6):277–303, 2003.

[670] M. Sorokina, S. Sygletos, and S. Turitsyn. Sparse identification for nonlinear optical communication systems: SINO method. *Optics Express*, 24(26):30 433–30 443, 2016.

[671] J. C. Spall. The Kantorovich inequality for error analysis of the Kalman filter with unknown noise distributions. *Automatica*, 31(10):1513–1517, 1995.

[672] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

[673] I. Stakgold and M. J. Holst. *Green's Functions and Boundary Value Problems*, 3rd edition, volume 99 of Pure and Applied Mathematics. John Wiley & Sons, 2011.

[674] W.-H. Steeb and F. Wilhelm. Non-linear autonomous systems of differential equations and Carleman linearization procedure. *Journal of Mathematical Analysis and Applications*, 77(2):601–611, 1980.

[675] R. F. Stengel. *Optimal Control and Estimation*. Courier Corporation, 2012.

[676] G. W. Stewart. On the early history of the singular value decomposition. *SIAM Review*, 35(4):551–566, 1993.

[677] G. Sugihara, R. May, H. Ye, et al. Detecting causality in complex ecosystems. *Science*, 338(6106):496–500, 2012.

[678] C. Sun, E. Kaiser, S. L. Brunton, and J. N. Kutz. Deep reinforcement learning for optical systems: A case study of mode-locked lasers. *Machine Learning: Science and Technology*, 1(4):045013, 2020.

[679] A. Surana. Koopman operator based observer synthesis for control-affine nonlinear systems. In *55th IEEE Conference on Decision and Control*, pages 6492–6499, 2016.

[680] A. Surana and A. Banaszuk. Linear observer synthesis for nonlinear systems using Koopman operator framework. *IFAC-PapersOnLine*, 49(18):716–723, 2016 (10th IFAC Symposium on Nonlinear Control Systems, NOLCOS).

[681] Y. Susuki and I. Mezić. A prony approximation of Koopman mode decomposition. In *2015 IEEE 54th Annual Conference on Decision and Control*, pages 7022–7027. IEEE, 2015.

[682] R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44, 1988.

[683] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[684] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems, 12*, pages 1057–1063, 1999.

[685] A. Svenkeson, B. Glaz, S. Stanton, and B. J. West. Spectral decomposition of nonlinear systems with memory. *Physical Review E*, 93:022211, 2016.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

[686] S. Svoronos, D. Papageorgiou, and C. Tsiligiannis. Discretization of nonlinear control systems via the Carleman linearization. *Chemical Engineering Science*, 49(19):3263–3267, 1994.

[687] D. L. Swets and J. Weng. Using discriminant eigenfeatures for image retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(8):831–836, 1996.

[688] K. Taira and T. Colonius. The immersed boundary method: A projection approach. *Journal of Computational Physics*, 225(2):2118–2137, 2007.

[689] K. Taira, S. L. Brunton, S. T. Dawson, et al. Modal analysis of fluid flows: An overview. *AIAA Journal*, 55(12):4013–4041, 2017.

[690] K. Taira, M. S. Hemati, S. L. Brunton, et al. Modal analysis of fluid flows: Applications and outlook. *AIAA Journal*, 58(3):998–1022, 2020.

[691] N. Takeishi, Y. Kawahara, Y. Tabei, and T. Yairi. Bayesian dynamic mode decomposition. *Twenty-Sixth International Joint Conference on Artificial Intelligence*, 2017.

[692] N. Takeishi, Y. Kawahara, and T. Yairi. Learning Koopman invariant subspaces for dynamic mode decomposition. In *Advances in Neural Information Processing Systems, 30*, pages 1130–1140, 2017.

[693] N. Takeishi, Y. Kawahara, and T. Yairi. Subspace dynamic mode decomposition for stochastic Koopman analysis. *Physical Review E*, 96(033310), 2017.

[694] F. Takens. Detecting strange attractors in turbulence. In *Dynamical Systems and Turbulence*, volume 898 of Lecture Notes in Mathematics, pages 366–381. Springer, 1981.

[695] Z. Q. Tang and N. Jiang. Dynamic mode decomposition of hairpin vortices generated by a hemisphere protuberance. *Science China: Physics, Mechanics and Astronomy*, 55(1):118–124, 2012.

[696] A. Taylor, A. Singletary, Y. Yue, and A. Ames. Learning for safety-critical control with control barrier functions. In *Learning for Dynamics and Control*, pages 708–717. PMLR, 2020.

[697] R. Taylor, J. N. Kutz, K. Morgan, and B. Nelson. Dynamic mode decomposition for plasma diagnostics and validation. *Review of Scientific Instruments*, 89:053501, 2018.

[698] R. Tedrake, Z. Jackowski, R. Cory, J. W. Roberts, and W. Hoburg. Learning to fly like a bird. In *14th International Symposium on Robotics Research*, 2009.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

[699] G. Tesauro. Practical issues in temporal difference learning. *Machine Learning*, 8(3):257–277, 1992.

[700] G. Tesauro. Temporal difference learning and TD-Gammon. *Communications of the ACM*, 38(3):58–68, 1995.

[701] S. Thaler, L. Paehler, and N. A. Adams. Sparse identification of truncation errors. *Journal of Computational Physics*, 397:108851, 2019.

[702] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 58(1):267–288, 1996.

[703] Z. Ting and J. Hui. EEG signal processing based on proper orthogonal decomposition. In *2012 International Conference on Audio, Language and Image Processing*, pages 636–640. IEEE, 2012.

[704] S. Tirunagari, N. Poh, K. Wells, et al. Movement correction in DCE-MRI through windowed and reconstruction dynamic mode decomposition. *Machine Vision and Applications*, 28(3–4):393–407, 2017.

[705] J. Tithof, B. Suri, R. K. Pallantla, R. O. Grigoriev, and M. F. Schatz. Bifurcations in a quasi-two-dimensional Kolmogorov-like flow. *Journal of Fluid Mechanics*, 828:837–866, 2017.

[706] E. Todorov, T. Erez, and Y. Tassa. MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

[707] C. Torrence and G. P. Compo. A practical guide to wavelet analysis. *Bulletin of the American Meteorological Society*, 79(1):61–78, 1998.

[708] A. Towne, O. T. Schmidt, and T. Colonius. Spectral proper orthogonal decomposition and its relationship to dynamic mode decomposition and resolvent analysis. *Journal of Fluid Mechanics*, 847:821–867, 2018.

[709] G. Tran and R. Ward. Exact recovery of chaotic systems from highly corrupted data. *SIAM Multiscale Modeling and Simulation*, 15(3):1108–1129, 2017.

[710] L. N. Trefethen. *Spectral Methods in MATLAB*. SIAM, 2000.

[711] L. N. Trefethen and D. Bau III. *Numerical Linear Algebra*, volume 50 of Other Titles in Applied Mathematics. SIAM, 1997.

[712] L. N. Trefethen, A. E. Trefethen, S. C. Reddy, and T. A. Driscoll. Hydrodynamic stability without eigenvalues. *Science*, 261(5121):578–584, 1993.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

[713] J. A. Tropp. Greed is good: Algorithmic results for sparse approximation. *IEEE Transactions on Information Theory*, 50(10):2231–2242, 2004.

[714] J. A. Tropp. Recovery of short, complex linear combinations via l_1 minimization. *IEEE Transactions on Information Theory*, 51(4):1568–1570, 2005.

[715] J. A. Tropp. Algorithms for simultaneous sparse approximation. Part II: Convex relaxation. *Signal Processing*, 86(3):589–602, 2006.

[716] J. A. Tropp. Just relax: Convex programming methods for identifying sparse signals in noise. *IEEE Transactions on Information Theory*, 52(3):1030–1051, 2006.

[717] J. A. Tropp and A. C. Gilbert. Signal recovery from random measurements via orthogonal matching pursuit. *IEEE Transactions on Information Theory*, 53(12):4655–4666, 2007.

[718] J. A. Tropp, A. C. Gilbert, and M. J. Strauss. Algorithms for simultaneous sparse approximation. Part I: Greedy pursuit. *Signal Processing*, 86(3):572–588, 2006.

[719] J. A. Tropp, J. N. Laska, M. F. Duarte, J. K. Romberg, and R. G. Baraniuk. Beyond Nyquist: Efficient sampling of sparse bandlimited signals. *IEEE Transactions on Information Theory*, 56(1):520–544, 2010.

[720] J. A. Tropp, A. Yurtsever, M. Udell, and V. Cevher. Randomized single-view algorithms for low-rank matrix approximation. Preprint arXiv:1609.00048, 2016.

[721] U. Trottenberg, C. W. Oosterlee, and A. Schuller. *Multigrid*. Elsevier, 2000.

[722] J. N. Tsitsiklis. Asynchronous stochastic approximation and Q-learning. *Machine Learning*, 16(3):185–202, 1994.

[723] J. N. Tsitsiklis. Efficient algorithms for globally optimal trajectories. *IEEE Transactions on Automatic Control*, 40(9):1528–1538, 1995.

[724] J. H. Tu and C. W. Rowley. An improved algorithm for balanced POD through an analytic treatment of impulse response tails. *Journal of Computational Physics*, 231(16):5317–5333, 2012.

[725] J. H. Tu, C. W. Rowley, E. Aram, and R. Mittal. Koopman spectral analysis of separated flow over a finite-thickness flat plate with elliptical leading edge. AIAA Paper 2011-2864, 2011.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

[726] J. H. Tu, C. W. Rowley, J. N. Kutz, and J. K. Shang. Spectral analysis of fluid flows using sub-Nyquist-rate PIV data. *Experiments in Fluids*, 55(9):1–13, 2014.

[727] J. H. Tu, C. W. Rowley, D. M. Luchtenburg, S. L. Brunton, and J. N. Kutz. On dynamic mode decomposition: Theory and applications. *Journal of Computational Dynamics*, 1(2):391–421, 2014.

[728] M. Turk and A. Pentland. Eigenfaces for recognition. *Journal of Cognitive Neuroscience*, 3(1):71–86, 1991.

[729] R. Van Der Merwe. Sigma-point Kalman filters for probabilistic inference in dynamic state-space models. Dissertation, Oregon Health & Science University, 2004.

[730] H. Van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double Q-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

[731] C. Van Loan. *Computational Frameworks for the Fast Fourier Transform*. SIAM, 1992.

[732] D. Venturi and G. E. Karniadakis. Gappy data and reconstruction procedures for flow past a cylinder. *Journal of Fluid Mechanics*, 519:315–336, 2004.

[733] S. Verma, G. Novati, and P. Koumoutsakos. Efficient collective swimming by harnessing vortices through deep reinforcement learning. *Proceedings of the National Academy of Sciences, USA*, 115(23):5849–5854, 2018.

[734] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th International Conference on Machine Learning*, pages 1096–1103. ACM, 2008.

[735] O. Vinyals, I. Babuschkin, W. M. Czarnecki, et al. Grandmaster level in Starcraft II using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

[736] P. R. Vlachas, W. Byeon, Z. Y. Wan, T. P. Sapsis, and P. Koumoutsakos. Data-driven forecasting of high-dimensional chaotic systems with long short-term memory networks. Proceedings of the Royal Society A, 474(2213):20170844, 2018.

[737] S. Volkwein. Model reduction using proper orthogonal decomposition. Lecture Notes, Institute of Mathematics and Scientific Computing, University of Graz, 2011.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

[738] S. Volkwein. Proper orthogonal decomposition: Theory and reduced-order modelling. Lecture Notes, Department of Mathematics and Statistics, University of Konstanz, 2013.

[739] S. Voronin and P.-G. Martinsson. RSVDPACK: Subroutines for computing partial singular value decompositions via randomized sampling on single core, multi core, and GPU architectures. Preprint arXiv:1502.05366, 2015.

[740] A. L.-C. Wang. An industrial strength audio search algorithm. In *Proceedings of the 4th International Conference on Music Information Retrieval*, pages 7–13, 2003.

[741] H. H. Wang, M. Krstić, and G. Bastin. Optimizing bioreactors by extremum seeking. *Adaptive Control and Signal Processing*, 13(8):651–669, 1999.

[742] H. H. Wang, S. Yeung, and M. Krstić. Experimental application of extremum seeking on an axial-flow compressor. *IEEE Transactions on Control Systems Technology*, 8(2):300–309, 2000.

[743] W. X. Wang, R. Yang, Y. C. Lai, V. Kovanis, and C. Grebogi. Predicting catastrophes in nonlinear dynamical systems by compressive sensing. *Physical Review Letters*, 106:154101 (4pp.), 2011.

[744] Z. Wang, I. Akhtar, J. Borggaard, and T. Iliescu. Proper orthogonal decomposition closure models for turbulent flows: A numerical comparison. *Computer Methods in Applied Mechanics and Engineering*, 237:10–26, 2012.

[745] Z. Wang, T. Schaul, M. Hessel, et al. Dueling network architectures for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1995–2003. PMLR, 2016.

[746] C. J. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3–4):279–292, 1992.

[747] C. Wehmeyer and F. Noé. Time-lagged autoencoders: Deep learning of slow collective variables for molecular kinetics. *Journal of Chemical Physics*, 148:241703, 2018.

[748] E. Weinan. *Principles of Multiscale Modeling*. Cambridge University Press, 2011.

[749] E. Weinan and B. Engquist. The heterogeneous multiscale methods. *Communications in Mathematical Sciences*, 1(1):87–132, 2003.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

[750] G. Welch and G. Bishop. An introduction to the Kalman filter. Technical Report, University of North Carolina at Chapel Hill, 1995.

[751] P. Whittle. *Hypothesis Testing in Time Series Analysis*, volume 4 of Statistics. Almqvist & Wiksells, 1951.

[752] O. Wiederhold, R. King, B. R. Noack, et al. Extensions of extremum-seeking control to improve the aerodynamic performance of axial turbomachines. AIAA Paper 092407, 2009 (39th AIAA Fluid Dynamics Conference).

[753] S. Wiggins, S. Wiggins, and M. Golubitsky. *Introduction to Applied Nonlinear Dynamical Systems and Chaos*, volume 2 of Texts in Applied Mathematics. Springer, 1990.

[754] K. Willcox. Unsteady flow sensing and estimation via the gappy proper orthogonal decomposition. *Computers and Fluids*, 35(2):208–226, 2006.

[755] K. Willcox and J. Peraire. Balanced model reduction via the proper orthogonal decomposition. *AIAA Journal*, 40(11):2323–2330, 2002.

[756] G. Williams, N. Wagener, B. Goldfain, et al. Information theoretic MPC for model-based reinforcement learning. In *2017 IEEE International Conference on Robotics and Automation*, pages 1714–1721. IEEE, 2017.

[757] M. O. Williams, I. G. Kevrekidis, and C. W. Rowley. A data-driven approximation of the Koopman operator: Extending dynamic mode decomposition. *Journal of Nonlinear Science*, 6:1307–1346, 2015.

[758] M. O. Williams, C. W. Rowley, and I. G. Kevrekidis. A kernel approach to data-driven Koopman spectral analysis. *Journal of Computational Dynamics*, 2(2):247–265, 2015.

[759] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256, 1992.

[760] D. M. Witten and R. Tibshirani. Penalized classification using Fisher's linear discriminant. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 73(5):753–772, 2011.

[761] F. Woolfe, E. Liberty, V. Rokhlin, and M. Tygert. A fast randomized algorithm for the approximation of matrices. *Journal of Applied and Computational Harmonic Analysis*, 25:335–366, 2008.

[762] J. Wright, A. Yang, A. Ganesh, S. Sastry, and Y. Ma. Robust face recognition via sparse representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(2):210–227, 2009.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

[763] C. F. J. Wu. On the convergence properties of the EM algorithm. *Annals of Statistics*, 11(1):95–103, 1983.

[764] X. Wu, V. Kumar, J. R. Quinlan, et al. Top 10 algorithms in data mining. *Knowledge and Information Systems*, 14(1):1–37, 2008.

[765] H. Ye, R. J. Beamish, S. M. Glaser, et al. Equation-free mechanistic ecosystem forecasting using empirical dynamic modeling. *Proceedings of the National Academy of Sciences, USA*, 112(13):E1569–E1576, 2015.

[766] E. Yeung, S. Kundu, and N. Hodas. Learning deep neural network representations for Koopman operators of nonlinear dynamical systems. Preprint arXiv:1708.06850, 2017.

[767] B. Yildirim, C. Chryssostomidis, and G. Karniadakis. Efficient sensor placement for ocean measurements using low-dimensional concepts. *Ocean Modelling*, 27(3):160–173, 2009.

[768] X. Yuan and J. Yang. Sparse and low-rank matrix decomposition via alternating direction methods. Preprint, 2009.

[769] I. Zamora, N. G. Lopez, V. M. Vilches, and A. H. Cordero. Extending the OpenAI Gym for robotics: A toolkit for reinforcement learning using ROS and Gazebo. Preprint arXiv:1608.05742, 2016.

[770] M. D. Zeiler, D. Krishnan, G. W. Taylor, and R. Fergus. Deconvolutional networks. In *IEEE Computer Vision and Pattern Recognition*, pages 2528–2535, 2010.

[771] C. Zhang and R. Ordóñez. Numerical optimization-based extremum seeking control with application to ABS design. *IEEE Transactions on Automatic Control*, 52(3):454–467, 2007.

[772] H. Zhang, C. W. Rowley, E. A. Deem, and L. N. Cattafesta. Online dynamic mode decomposition for time-varying systems. Preprint arXiv:1707.02876, 2017.

[773] T. Zhang, G. Kahn, S. Levine, and P. Abbeel. Learning deep control policies for autonomous aerial vehicles with MPC-guided policy search. In *2016 IEEE International Conference on Robotics and Automation*, pages 528–535. IEEE, 2016.

[774] W. Zhang, B. Wang, Z. Ye, and J. Quan. Efficient method for limit cycle flutter analysis based on nonlinear aerodynamic reduced-order models. *AIAA Journal*, 50(5):1019–1028, 2012.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

[775] P. Zheng, T. Askham, S. L. Brunton, J. N. Kutz, and A. Y. Aravkin. Sparse relaxed regularized regression: SR3. *IEEE Access*, 7(1):1404–1423, 2019.

[776] S. Zlobec. An explicit form of the Moore–Penrose inverse of an arbitrary complex matrix. *SIAM Review*, 12(1):132–134, 1970.

[777] H. Zou and T. Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(2):301–320, 2005.

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Index

N-way arrays, 55, 56
Q-learning, 513, 517, 518, 522, 523, 526
k-fold cross-validation, 191–193
k-means, 215, 216, 219, 245
k-nearest neighbors (kNN), 246

activation function, 257–259, 267, 271, 272
actor–critic, 527, 528
actuator, 391, 393
adjoint, 396, 447–451, 454, 455, 462
agent, 505–507, 510–512, 523, 529, 531, 533
Akaike information criterion (AIC), 197, 228, 344
alternating descent method, 173
autoencoder, 280, 285, 286, 289, 295, 365–367, 530, 629, 632–634, 654

backpropagation (backprop), 260, 262–264, 267, 268
balanced input–output model, 453
Bayesian information criterion (BIC), 197, 198
Bellman optimality, 509, 514, 536–538
boosting, 244, 247

classification, 34, 40, 95, 142–145, 244–246, 252–254, 256, 258–260, 263, 267, 275, 326, 496
classification trees, 239, 244
closed-loop control, 377, 381, 382
clustering, 32, 33, 48, 239, 245, 246
coherent structure, 4, 120, 313, 326, 396, 436, 437
compressed sensing, 21, 123–128, 132, 135, 314
compression, 12, 13, 79, 102, 105, 109, 118–120, 123, 132, 325
control, 8, 64, 75, 124, 204, 310, 326, 345, 357, 396, 397, 437, 438, 469
control theory, 124, 314, 376, 430, 436, 474, 477
controllability, 64, 389–392, 394, 395, 438, 440, 441, 445, 450
convex optimization, 128, 177, 336, 398, 484
convolutional neural network (CNN), 245, 271
cost function, 378, 399, 402, 411, 428, 469, 481, 488, 496
CP tensor decomposition, 56–59
cross-validation, 139, 140, 188, 190, 191, 203, 214, 216, 225, 231, 262, 327, 358
curve fitting, 160, 162, 168

data arrays, 55
data labels, 203, 210, 211, 213, 266, 274
data matrix, 42, 47, 56, 58, 145, 217, 256, 316, 328, 475, 562
deep convolutional neural network (DCNN), 271–273
deep learning, 252, 253, 271
deep reinforcement learning, 505, 507, 525, 527
DeepONet, 639, 642, 643
dendrogram, 219, 220

dimensionality reduction, 4, 8, 229, 314, 316, 376, 475, 542, 547
discrete cosine transform (DCT), 130
discrete empirical interpolation method (DEIM), 586, 603
discrete wavelet transform, 103, 104, 109
DMD eigenvalue, 317, 319
DMD mode (also *dynamic mode*), 314, 316, 325, 326
dynamic mode decomposition (DMD), 313, 324
dynamic programming, 513–515, 519, 520, 528, 535
dynamical system, 9, 64, 275, 305–314, 331, 338, 344, 376–378, 402, 437, 586

eigensystem realization algorithm (ERA), 326, 361, 451–453
ensemble learning, 244, 247
environment, 505–507, 510, 511, 513, 518, 525
expectation-maximization algorithm (EM), 223–226, 245
extremum-seeking control (ESC), 491–493, 497

fast Fourier transform, 3, 64, 76, 79, 119, 545
feature engineering, 204, 210, 272, 619
feedback control, 363, 376, 378, 380, 435
feedforward control, 377, 426
Fourier transform, 18, 65, 72, 74, 356, 418, 556, 588, 610

Gabor transform, 91–93
Galerkin projection, 332, 337, 436, 545, 549, 550, 605
gappy POD, 550, 585, 605
gated recurrent unit (GRU), 283, 294
Gaussian mixture models (GMM), 223, 225, 227
generative adversarial network (GAN), 289, 297
gradient descent, 160, 167–169, 235, 255, 259, 266
Gramian, 395, 438, 440, 443, 445

Hankel matrix, 361, 448, 450, 454
Hilbert space, 64, 344–346, 348, 360
hindsight experience replay, 528, 529

imitation learning, 523
incoherent measurements, 133, 135
information theory, 195
inner product, 47, 55, 65, 133, 349, 356, 435, 440, 550, 552, 561, 587, 589

Johnson–Lindenstrauss (JL), 136

Kalman filter, 401, 403, 404, 413, 457, 458, 501
kernel methods, 237, 327, 358
Koopman eigenfunction, 359, 361
Koopman operator, 349–351, 357

Laplace transform, 75, 98, 100–102, 418–420, 422
LASSO, 138, 139, 184, 191, 257, 264
least-squares fit, 137, 162, 190
linear discriminant, 228, 229
linear system, 19, 128, 166, 256, 310, 327, 339, 398, 419, 427, 548
linear–quadratic regulator (LQR), 396–398, 477, 488
long short-term memory (LSTM), 283, 294, 299, 574, 576
low rank, 10, 146, 148, 203, 204, 436, 438, 549, 550, 553, 554, 560, 567, 585, 586

machine learning, 26, 32, 105, 162, 188, 206, 208, 216, 232, 237, 239, 244, 252, 266, 451

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

Markov decision process (MDP), 507, 508, 510, 514, 517, 522, 524, 535
Markov parameters, 458
max pooling, 273
mixture models, 223, 225, 227, 245
model predictive control (MPC), 473, 474, 477, 478
model reduction, 395, 435–437, 607, 608
model selection, 160, 163, 184, 190, 195
Moore's law, 124
multi-resolution analysis (MRA), 102
multi-scale, 77, 327, 376

naive Bayes, 245, 246
neural networks, 169, 245, 252, 253, 268, 365
neural operator, 639, 642, 643, 652
noise, 128

observability, 64, 345, 389, 391, 396, 412, 438, 440, 441, 445, 450
observable function, 345, 346, 356
off-policy reinforcement learning, 513, 522, 523, 527
on-policy reinforcement learning, 513, 521, 522
open loop, 485
optimization
  gradient descent, 170
  steepest descent, 170
outliers, 26, 137, 145, 164, 184
over-determined system, 4, 20, 139, 160, 178, 333

pagerank, 244, 247
Pareto front, 186–188
Perron–Frobenius operator, 313, 356
physics-informed neural network (PINN), 644–646
policy function, 513–515, 517, 518, 524, 525, 527
policy iteration, 513–515, 517, 523
polynomial basis functions, 236, 238
principal component, 29, 33
principal component analysis (PCA), 3, 8, 27, 49, 136, 145, 252, 450
principal components, 28, 31, 145
proper orthogonal decomposition (POD), 4, 136, 313, 329, 435, 436, 438, 541, 548, 604
pseudo-inverse, 4, 19, 21–23, 129, 190, 191, 256–258, 315–317, 459

quality function, 513, 517–519, 524–528, 530

radial basis function (RBF), 238, 294
random forest, 239, 244
randomized algorithms, 49, 51
randomized SVD, 49, 50
rank truncation, 204, 354
recurrent neural network (RNN), 271, 281, 294, 574
reduced-order model (ROM), 326, 566, 603
reference tracking, 379, 382, 426, 487
regression, 19, 23, 137, 160, 162, 165, 211, 214, 264, 326, 331, 333, 334, 338, 452, 580
reinforcement learning (RL), 203, 211, 469, 473, 482, 505–507, 509, 510, 512, 514, 517, 521, 525, 527, 528, 530, 535
restricted isometry property (RIP), 126, 134
reward, 505, 508–512, 515, 516, 518–521, 524, 529, 531
reward shaping, 528
ridge regression, 184
robust control, 406, 410, 418, 427
robust fit, 184
robust principal component analysis (rPCA), 145
robust statistics, 184

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.

sampling rate, 126, 130
SARSA, 521–523
sensor, 328, 363, 409, 412, 416, 430, 453, 454, 481, 492, 494, 499
singular value decomposition (SVD), 3, 4, 48, 51, 135, 136, 206, 313, 315, 316, 356, 437, 447, 549, 553, 606, 609
snapshot, 55, 315, 339, 396, 447–450, 475, 549, 554, 560, 605, 608
sparse identification of nonlinear dynamics (SINDy), 331, 332
sparse regression, 137, 332–334, 338, 344, 580
sparse representation, 118, 123, 142, 325, 326
sparsity, 128
spectrogram, 91, 92, 94
state space, 360, 435, 452
stochastic gradient descent (SGD), 260, 262, 268
supervised learning, 210–212
support vector machine (SVM), 232, 245
system identification (system ID), 314, 326, 338, 362, 435, 436, 438, 451, 452, 473, 474, 482

temporal difference (TD) learning, 511, 512, 519–521, 530
temporal difference error, 528
temporal difference target, 527
tensors, 55
test data, 216, 219, 246
training data, 228, 230, 233, 271, 274

uncertainty principle, 95, 97
under-determined system, 4, 20, 124, 128, 143, 160, 181, 253, 255
unitary matrix, 18
unsupervised learning, 203, 210, 211, 244, 245

value function, 509, 513–521, 524, 525, 527, 528, 530, 536, 538
value iteration, 513, 514, 516, 517, 523

wavelet, 102, 135
windowed FFT (Gabor transform), 92
withhold data, 228

Copyright © 2021 Brunton & Kutz, Cambridge University Press. All Rights Reserved.