Exercises 06

CMU 15-462/662

1 An Exponentially Better Representation of Rotations

How can you smoothly interpolate rotations? And what does it mean to “average” a bunch of rotations? Many problems in visual computing require you to answer these kinds of questions. For instance, suppose you have many noisy estimates of a camera pose, and want to average these out to get a more reliable estimate. Or suppose you want to interpolate between known coordinate frames at several points on a surface, to get a smoothly-varying frame. These tasks are hard to perform directly because rotation matrices don’t form a vector space—for instance, adding two rotation matrices doesn’t give you another rotation matrix!

However, we can convert rotation matrices into an axis-angle form where rotations are represented by ordinary vectors. The direction of the vector gives the axis of rotation, and the magnitude of the vector gives the angle of the rotation. These vectors can be added, scaled, and averaged just like normal vectors—and then we can convert back to matrices in order to perform rotations.

$$\begin{tikzcd}
\boxed{\begin{matrix}\theta \in \mathbb{R} \\ \mathbf{u} \in \mathbb{R}^n\end{matrix}} \arrow[r, shift left, "\exp"] & \boxed{R \in \mathbb{R}^{n\times n}} \arrow[l, shift left, "\log"] \\
\textbf{angle \& axis} & \textbf{rotation matrix}
\end{tikzcd}$$

More specifically,

- the **exponential map** can be used to turn an angle θ and unit-length axis u into a rotation matrix R, and
- the **logarithmic map** can be used to turn a rotation matrix R into an axis and angle.

As we’ll see, these maps generalize the usual exp and log functions that you know and love[1]. Beyond being very useful for manipulating rotations, they help connect the dots on some of the rotation representations we’ve already seen in lecture (Euler angles, complex numbers, quaternions, ...).

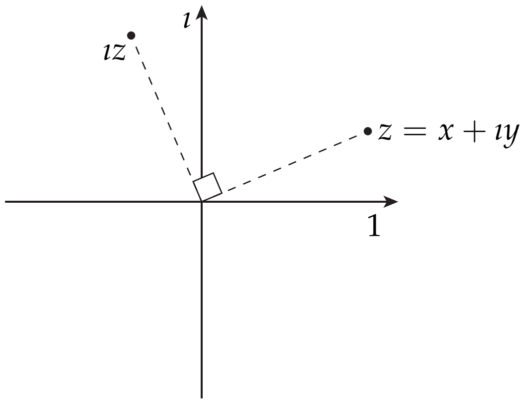
[1]By the way, the super fancy terminology people often use to talk about all this stuff is that the rotation matrices form a *Lie group* $\mathrm{SO}(n)$, and the associated vector space is the *Lie algebra* $\mathfrak{so}(3)$. The exponential map is then a map $\exp : \mathfrak{so}(3) \to \mathrm{SO}(3)$, and the logarithmic map is $\log : \mathrm{SO}(3) \to \mathfrak{so}(3)$. You **definitely won’t need to know these terms for this class**—but if you start working a lot with rotations, it’s a perspective well worth understanding!

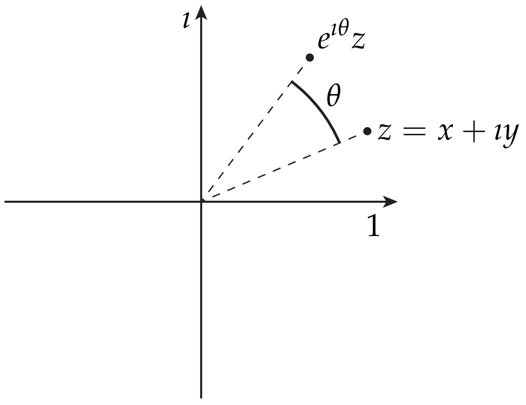
2 Rotations in 2D

Let's start out in 2D, where we already started to see how rotations are connected to the exponential map. In particular, we said that we can rotate a point $z \in \mathbb{C}$ by an angle θ via the map

$$z \mapsto e^{\imath\theta} z.$$

(If you like, you can think of the "axis" in this case as the direction pointing *out of the plane*.) Let's try to understand this idea in a bit more depth.





1. First of all, suppose we have a complex number $z = x + \imath y$. If $\imath$ denotes the imaginary unit, how can we represent the product $\imath z$ as a matrix-vector product between a real 2×2 matrix $J \in \mathbb{R}^{2\times 2}$ (representing $\imath$) and a column vector $[x\ y]^\mathsf{T}$ (respresenting z)?

2. As a sanity check, let's now compute J^2 by taking a matrix-matrix product. What do we get?

3. Now suppose we want to represent the complex product

$$(\cos(\theta) + \imath \sin(\theta))z$$

as a matrix-vector product. Which real 2×2 matrix should we multiply the column vector $[x\ y]^\mathsf{T}$ by? (Hint: start by writing everything in terms of the matrices $I, J \in \mathbb{R}^{2\times 2}$.)

Taylor series expansions

$\cos(x) = \sum_{k=0}^{\infty} \frac{(-1)^k}{(2k)!} x^{2k}$	$= 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \cdots$	$\sin(x) = \sum_{k=0}^{\infty} \frac{(-1)^k}{(2k+1)!} x^{2k+1}$	$= x - \frac{x^3}{3!} + \frac{x^5}{5!} - \cdots$
$\exp(x) = \sum_{k=0}^{\infty} \frac{1}{k!} x^k$	$= 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$	$\log(1-x) = -\sum_{k=1}^{\infty} \frac{1}{k} x^k$	$= -x - \frac{x^2}{2} - \frac{x^2}{3} - \cdots$

From here, we'll finally make sense of the relationship to exponentiation. To do so we'll make use of the *Taylor series*, which expresses a given function $f(x)$ around a fixed point x_0 as

$$f(x) = \sum_{k=0}^{\infty} \frac{1}{k!} f^{(k)}(x_0)(x - x_0)^k,$$

where $f^{(k)}$ denotes the kth derivative of f. In the special case where $x_0 = 0$, this approximation is called a *Mclaurin series*. Several common examples are given above (though in principle you could just derive these yourself by applying the general formula!).

One way to generalize common functions to matrices is to apply the same Taylor series to a matrix $A \in \mathbb{R}^{n\times n}$ rather than a scalar variable $x \in \mathbb{R}$. In this case, the power A^k just means we multiply together k copies of the matrix A.

4. Let $A = \theta J$ for some angle θ and the matrix J derived above[2]. Write out the Taylor series for $\exp(A)$, truncated to the first four terms.

[2]Side note: A is called an *antisymmetric matrix* or *skew-symmetric matrix*, since $A^\mathsf{T} = -A$.

5. Re-write this same truncated series using *Horner's rule*, i.e., factor out multiples of A so that you get a nested expression in terms of just A. For example, if you just had an ordinary polynomial $a + bx + cx^2 + dx^3$, Horner's rule would re-write this polynomial as $a + x(b + x(c + dx))$.

6. Now work out the entries of this matrix by performing the additions and multiplications from the "inside out." (Notice that, even for a computer, doing it this way is a *lot* more efficient than repeating all those powers!)

7. Inspect the entries of your solution from the previous question. What does this matrix remind you of? What do you think will happen if you use more terms of the Taylor series?

8. For ordinary numbers, the logarithm is the inverse of the exponential and vice-versa. I.e.,

$$\log(e^x) = x \qquad \text{and} \qquad e^{\log(x)} = x.$$

Suppose you are given a matrix

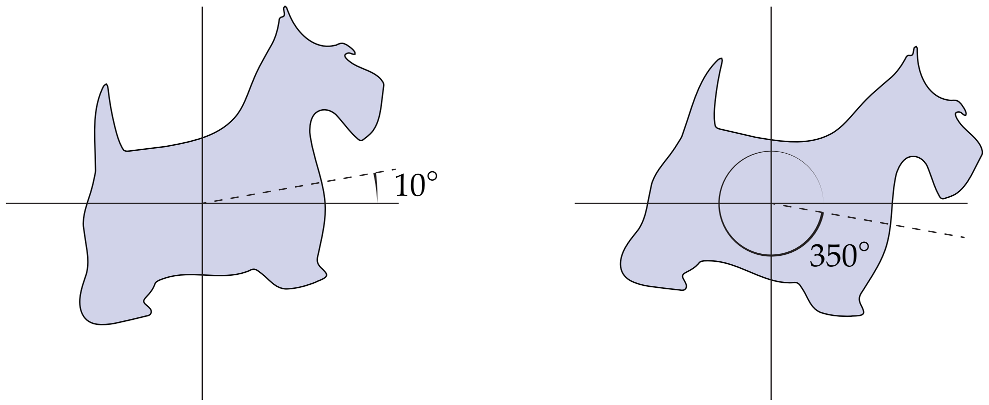
$$R = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

and are told to compute $log(R)$, i.e., you are told to find some matrix $A \in \mathbb{R}^{2\times 2}$ such that $\exp(A) = R$. What matrix A would you use? How could you compute the entries of this matrix, given only the four entries of R?

9. Is the matrix exponential $\exp(A)$ an injective (i.e., 1-to-1) map? Why or why not? What does this fact tell you about representing rotations as angles θ versus rotation matrices R?

10. In light of the previous question, is the solution you gave for $\log(R)$ unique? Or are there other matrices A such that $\exp(A) = R$? If the solution is not unique, what is special about the matrix you computed in the previous part?

At this point we start to see how exponentiation is related to rotation. But why is this approach *useful*? Let's start by playing around with interpolation of rotations.



11. Suppose that in 2D we start out with Scotty at an angle $\theta_0 = 10^\circ$ to the horizontal, and want to rotate him to a pose where he's making an angle 350° to the horizontal over a time interval $t \in [0, 1]$. What's a very simple way to write down a family of rotation matrices $R(t)$ that performs this animation? Do *not* use the exp/log map for this problem.

12. Wouldn't it be much easier to just average the rotation matrices themselves? Give one example where interpolating the matrices directly turns out *very badly*.

13. If you think of each rotation R_θ as a point $(\cos\theta, \sin\theta)$ on the unit circle, does the family of matrices $R(t)$ from your previous solution describe the shortest path of rotations between the inital and final poses? If not, how might we get a shorter path? This time you can and should use the exp/log maps! (Hint: which rotation takes you directly from the initial pose to the final pose?)

3 Rotations in 3D

In 3D[3], we have a similar setup:

- Rotations are represented by 3×3 matrices R that are orthogonal ($R^\mathsf{T} R = RR^\mathsf{T} = I$) and have determinant $+1$.
- A rotation by θ around a unit-length axis $u \in \mathbb{R}^3$ is encoded by a 3×3 skew-symmetric matrix $A = \theta\hat{u}$, where
$$\hat{u} = \begin{bmatrix} 0 & -u_z & u_y \\ u_z & 0 & -u_x \\ -u_y & u_x & 0 \end{bmatrix}.$$
- To go from axis-angle form to rotation matrix form, we can evaluate the exponential map $R = \exp(A)$.
- To go from a rotation matrix to axis-angle form, we can evaluate the log map $A = \log(R)$.

But don't take my word for it! Let's make sure this machinery still works as expected in 3D. First, a couple little warm-ups to get us more comfortable with this setup.

14. In 3D, do rotation matrices form a vector space? Do skew-symmetric matrices form a vector space? Give a little argument for why/why not in each case.

[3]In fact, this same setup applies for n-dimensional rotations—just replace "3" with "n".

Ok, so rotations don't form a vector space. But what does the set of all rotations look like? Just like it helps to visualize vectors as "little arrows," visualizing rotations geometrically is super helpful whenever we face problems like how to interpolate or average rotations. In general, we will call the set of all n-dimensional rotations the *special orthogonal group* $\mathrm{SO}(n)$. "Orthogonal" because rotation matrices are orthogonal; "special" because they are not just any orthogonal matrices—they must also have positive determinant. In other words,

$$\mathrm{SO}(n) := \{R \in \mathbb{R}^{n\times n} \mid R^T R = I,\ \det(R) > 0\}.$$

We call this set a "group" because composition of rotations (via matrix multiplication) satisfies some very natural properties: the product of two rotations is a rotation; every rotation R can be reversed by some inverse rotation R^{-1}, there is an "identity" rotation that does nothing ($R = I$), and the way we group rotations doesn't matter: $(R_1R_2)R_3 = R_1(R_2R_3)$. But what is the *shape* of the rotation group?

15. In 2D, we said that all rotations could be expressed as complex numbers $e^{\imath\theta} = \cos(\theta) + \imath\sin(\theta)$, for any angle $\theta \in [-\pi, \pi)$. Equivalently, we could say that rotations are just complex numbers $z \in \mathbb{C}$ with unit norm: $|z| = 1$. What shape does this set describe?

16. In general, for any function f we can consider the set of points x such that $f(x) = 0$. For instance, in the previous question the function $f(x) = |x|^2 - 1$ describes the circle. In general, if we start out in n dimensions, then each scalar condition reduces the dimension of the set by one. For instance, consider the functions $f(x) = |x|^2 - 1$ and $g(x) = x_1$ (where x_1 is the first coordinate of x). Describe the set

$$S := \{x \in \mathbb{R}^2 | f(x) = 0, g(x) = 0\}.$$

What points does it contain? What dimension is it?

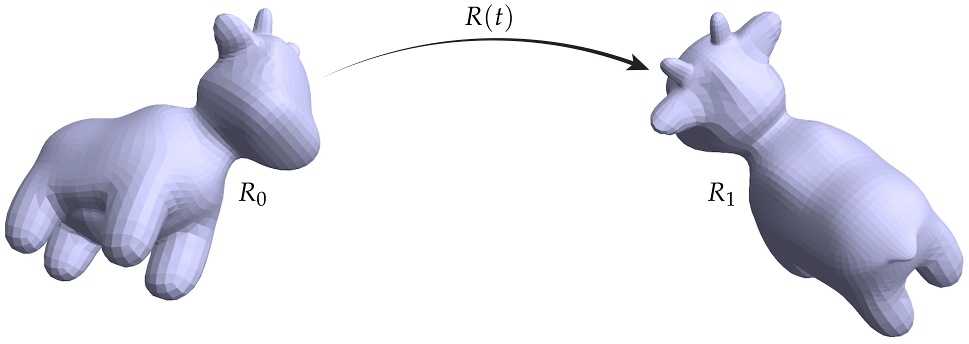
17. Another way of getting our head around the shape of the rotation group is to consider the relationship $A^T A = I$ characterizing orthogonal matrices, or rather, the function $f(A) = A^T A - I$. Importantly, the equation $f(A) = 0$ is not a scalar equation: it relates all n^2 entries of the matrix $A^T A$ to the n^2 entries of the identity matrix I. If we let

$$\mathrm{O}(n) := \{A \in \mathbb{R}^{n\times n} | A^T A - I = 0\}$$

be the set of orthogonal matrices, what dimension does this set have for $n = 2$? For $n = 3$? For general n?

18. Remember that rotations are not just orthogonal matrices—they're orthogonal matrices with positive determinant. However, this additional *inequality* doesn't reduce the dimension of the set any further: instead, it splits the orthogonal matrices $\mathrm{O}(n)$ into two pieces: those with positive determinant (rotations) and those with negative determinant. What do the negative-determinant orthogonal matrices correspond to?

19. Let's get back to the conversion between rotation matrices and the axis-angle encoding. In particular, given a 3×3 skew-symmetric matrix $A^{\mathsf{T}} = -A$ encoding a rotation in axis-angle form, how can we recover the axis u and angle θ of rotation?



20. Suppose we have two rotation matrices given in rotation angles:

$$\begin{aligned} R_0 &= R^x(\alpha_0)R^y(\beta_0)R^z(\gamma_0), \\ R_1 &= R^x(\alpha_1)R^y(\beta_1)R^z(\gamma_1), \end{aligned}$$

where $R^x(\theta)$ denotes a rotation by θ around the x-axis (and similarly for y, z). What's a simple way to interpolate between these two rotation matrices using Euler angles? Give an expression for a family of rotation matrices $R(t)$ with $t \in [0, 1]$ such that $R(0) = R_0$ and $R(1) = R_1$. For now, do *not* use the exp/log map.

21. Based on your intuition from the 2D case, how do you think you might interpolate the rotations using the exp/log map for 3D rotations?

22. If you were paying close attention in our lecture on vector calculus, you may recall that $\hat{u}$ represents a cross product with the vector u, i.e., $\hat{u}x = u \times x$ for any vector $x \in \mathbb{R}^3$. Using your knowledge of the cross product, make a *geometric* argument that $\hat{u}^3 = -\hat{u}$, and hence $\hat{u}^{k+2} = -\hat{u}^k$ for $k \geq 1$.

23. Derive Rodrigues' formula $\exp(\theta\hat{u}) = I + \sin(\theta)\hat{u} + (1 - \cos(\theta))\hat{u}^2$ by writing out the left-hand side via the Taylor series for the matrix exponential.

24. Is the matrix $\exp(\theta u)$ provided by Rodrigues' formula a rotation matrix for all vectors u and angles θ? How could you check?

25. In 3D, is the exponential map injective (1-to-1), or not? At a high level, then, what do you think the 3D log map should give us?

26. Let's define the 3D log map explicitly. In particular, given a 3×3 rotation matrix R, the log map should produce a matrix $A = \log(R)$ such that $\exp(A) = R$. Give an explicit formula for $\log(R)$. (Hint: what happens when you take the trace of Rodrigues' formula for R? What happens when you antisymmetrize this formula, i.e., evaluate $R - R^{\mathsf{T}}$?)

27. For ordinary numbers, we have the useful identity

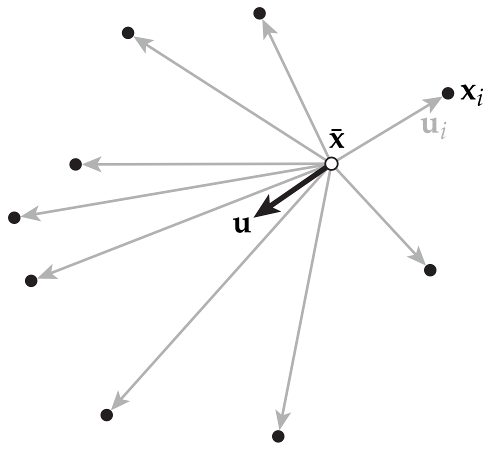
$$e^{a+b} = e^a e^b.$$

Do you think this same identity holds for matrices, i.e., that

$$\exp(A_1 + A_2) = \exp(A_1)\exp(A_2)$$

for any two skew-symmetric matrices A_1, A_2? In 2D? In 3D? Why or why not? Give an argument that makes an appeal to the way rotations behave *geometrically*—not an algebraic argument.

28. Finally, if all we wanted to do with the log/exp map was interpolate rotations, this might feel like much ado about nothing! But now that we can treat rotations like vectors, there's other useful stuff we can do. For instance, given a collection of rotation matrices $R_1, \ldots, R_n \in \mathbb{R}^{3\times 3}$, we can define a meaningful notion of the *average rotation*. As discussed above, we can't just take an average of the matrices $\frac{1}{n}\sum_{i=1}^n R_i$, since the average of rotation matrices will not in general be a rotation matrix.



Instead, let's first think about a "funny" way to average a bunch of points $\mathbf{x}_1, \ldots, \mathbf{x}_n$ in the plane. Instead of just directly taking the average (which we can't do with rotations), we'll pick some initial guess $\bar{\mathbf{x}}$ for the average. We'll then compute, for each point, the smallest vector $\mathbf{u}_i$ that takes us from $\bar{\mathbf{x}}$ to $\mathbf{x}_i$. In the case of points in the plane, this vector is just $\mathbf{u}_i = \mathbf{x}_i - \mathbf{x}$. We'll then compute the average vector $\mathbf{u} := \frac{1}{n}\sum_{i=1}^{n} \mathbf{u}_i$ that tries to pull us toward all the points, and take a little step in this direction of size $\tau \in [0,1]$. If we reach a point where $\mathbf{u} = 0$—or at least, below some very small $\epsilon > 0$—then the algorithm stops and we know we've reached the average. This algorithm can be summarized as follows:

- Pick an initial guess $\bar{\mathbf{x}} \in \mathbb{R}^2$.
- Do
  - $\mathbf{u}_i \leftarrow \mathbf{x_i} - \mathbf{x}$
  - $\mathbf{u} \leftarrow \frac{1}{n}\sum_{i=1}^{n} \mathbf{u}_i$
  - $\bar{\mathbf{x}} \leftarrow \bar{\mathbf{x}} + \tau\mathbf{u}$
- While $|\mathbf{u}| > \epsilon$

Your task is to re-write this same algorithm for rotations. For instance, given a current guess $\bar{R}$ for the average rotation, how do you find the smallest rotation from $\bar{R}$ to each of the R_i? Where and how do you compute the average vector? How do you take a small step in this direction? How do you pick a valid initial guess?

29. Implement some code (in your favorite language) that computes the average of a given list of rotations. You should now have all the tools and formulas you need to really do this! (Note: you may need to be a bit more careful about corner cases when you implement the log map...)