PSYCH-GA.2211/NEURL-GA.2201 – Fall 2024
Mathematical Tools for Neural and Cognitive Science

Homework 1

Due: 26 Sept 2024
(late homeworks penalized 10% per day)

Please: don't wait until the day before the due date... start *now*! [Click here for submission instructions]

Important! Unless we specify otherwise, do not use the [MATLAB Linear Algebra Library] or Python `np.linalg` library or equivalent functions from other built-in libraries for this homework (for example, the `norm` and `inv` functions). In general, write things to express the underlying math, using the multiplication operator (which may require transposing some matrices or vectors). You are welcome (encouraged, even!) to write your own functions to carry out more complex computations.

1. **Inner product with a unit vector.** Given a unit N-dimensional vector $\hat{u}$, and an arbitrary N-dimensional vector $\vec{v}$, write the following (MATLAB or Python) functions:

   (a) A function named `projection` that returns a vector that is the component of $\vec{v}$ lying along the direction $\hat{u}$.

   (b) A function named `ortho` that returns a vector that is the component of $\vec{v}$ that is orthogonal (perpendicular) to $\hat{u}$, and

   (c) A function named `distance` that returns the distance from $\vec{v}$ to the line that lies along direction specified by $\hat{u}$.

   $\hat{u}$, Verify that your code is working by testing it on random vectors $\hat{u}$ and $\vec{v}$ (generate these using `randn` in MATLAB or `np.random.randn` in Python. Remember to re-scale $\hat{u}$ so that it has unit length). First, do this visually with 2-dimensional vectors, by plotting $\hat{u}$, $\vec{v}$, and the two components described in (a) and (b). (hint: execute `axis equal` in MATLAB or `plt.axis('equal')` in Python to ensure that the horizontal and vertical axes have the same units). Then test it numerically in higher dimensions (e.g., 4) by writing expressions to verify each of the following, and executing them on a few randomly drawn vectors $\vec{v}$:

   - the vector in (a) points in the same (or opposite) direction as $\hat{u}$.
   - the vector in (a) is orthogonal to the vector in (b).
   - the sum of the vectors in (a) and (b) is equal to $\vec{v}$.
   - the value returned by (c) is equal to the magnitude (norm) of the vector in (b).
   - the sum of squared lengths of the vectors in (a) and (b) is equal to $||\vec{v}||^2$.

2. **Experimentally testing a system for (non)linearity.** There is no coding required for this part (though you are welcome to use code or math to help explain your thought process). Please create a separate text or markdown cell for each subproblem.

Consider each of the systems below. We observe a set of input-output pairs in the form of *input* $\rightarrow$ *output*. For each system, determine whether the system could be a *linear system*. If not, explain your reasoning.

If yes, provide a matrix M that is consistent with the examples provided, and state whether M is unique (i.e., the *only* matrix that could explain these observations).

(a) System 1:

0 $\longrightarrow$ [2, -3]

(b) System 2:

[2, 3] $\longrightarrow$ 9
[-1, 2] $\longrightarrow$ 4

(c) System 3:

[-1, 2] $\longrightarrow$ [0, 2]
[3, 1] $\longrightarrow$ [-1, 1]
[0, 7] $\longrightarrow$ [-2, 7]

(d) System 4:

[3, 2] $\longrightarrow$ [-4, 1]
[-6, -4] $\longrightarrow$ [8, -2]

3. **Geometry of linear transformations.** The files `sysN.p` (where `N=1,2,3,4`) each provide a function that implements a linear system whose input and output are both 2-dimensional vectors. These are "pre-compiled" MATLAB files where you can apply the function to inputs and see the result, but you cannot see the source code. In Python, download `linear_systems2020.py` and import the mystery functions into your workspace using `from linear_systems2020 import sys1, sys2, sys3, sys4`. For each of these:

   (a) Generate 30 random 2D inputs using `randn` or Python equivalent. Compute the corresponding outputs. Plot 30 line segments from each input to output, labeling input and output with different symbols or colors. Use `hold on` in MATLAB (no need in Python), and plot points at start and end of each segment. Describe, in words, what the system is doing to the input space.

   (b) Characterize the system, by measuring its response to impulses, and embedding these in a matrix M. Compute the SVD of this matrix, and explain how the components of the SVD relate to the description you provided in the previous part.

   (c) Generate a data matrix P with 8 columns corresponding to the vertices and faces of a square of size 2x2 centered at the origin. If you start on the horizontal axis, and work counter-clockwise, the first few vectors are $(0,1),(1,1),(1,0),(1,-1)$.... Plot a single blue curve through these points, and a red star (asterisk) at the location of one of the vertices (e.g., second point in example above). Now consider and discuss the action of

the matrix M on this set of points. In particular, apply the transformations derived from the SVD of the matrix M one at a time to the full set of points (again, think of a way to do this without using a for loop!), plot each of the transformations, and describe what geometric changes you see (and why).

4. **A simple visual neuron.** There is no coding required for this part. Please create a separate text or markdown cell for each subproblem.

   You are a biologist exploring a previously unexplored island in the South Pacific. You come across a new species of salamander and decide to characterize how its visual system works.

   You find a retinal neuron that responds *only* to the intensity of light at 5 different localized regions on the retina - we can represent the intensity at these 5 regions as a vector $\vec{v} = [v_1, v_2, v_3, v_4, v_5]$, where the components are all *non-negative*. The output r of the retina is a weighted sum of the intensities at each location; specifically, $r = v_1 + 2.3v_2 + 1.5v_3 + v_4 + 6v_5$.

   (a) Is this system linear? If so, express the response as a matrix multiplied by the input intensity vector v. If not, explain why not.

   (b) What unit-length stimulus vector (i.e., vector of light intensities) elicits the largest response in this neuron? Write out your reasoning and explain how you know this is the largest possible response. [hint: try to reason through what this looks like geometrically in 2D! What kind of relationship must exist between r and the weights for the response to be large?]

   (c) What physically-realizable unit-length stimulus vector produces the *smallest* response in this neuron? Explain your reasoning. [hint: try to reason through what this would look like geometrically in 2D! What does it mean for a vector to be physically realizable?]

5. **Gram-Schmidt.** A classic method for iterative construction of an orthonormal basis is known as *Gram-Schmidt orthogonalization*. First, one generates an arbitrary unit vector (typically, by normalizing a vector created with `randn` in MATLAB or `np.random.normal` in Python). Each subsequent basis vector is created by generating another arbitrary vector, subtracting off the projections of that vector along each of the previously created basis vectors, and normalizing the remaining vector. The output should be a matrix of random unit vectors all of which are orthogonal to each of the other vectors in the matrix.

   Write a function `gramschmidt` that takes a single argument, `n`, specifying the dimensionality of the basis. It should then generate an `n`×`n` matrix whose columns contain a set of orthogonal normalized unit vectors. Try your function for `n` = 3, and plot the basis vectors (you can use `rotate3d` in MATLAB or see footnote[1] in Python to interactively examine these). Check your function numerically by calling it for an `n` much larger than 3 (e.g. 1000) and verifying that the resulting matrix is orthonormal (hint: you should be able to do this without using loops). Check that both the columns *and* the rows separately satisfy the conditions of being orthonormal. **Extra credit:** Make your function *recursive* – instead of using a `for` loop, have the function call itself, each time adding a new column to the matrix of previously created orthogonal columns. To do this, you'll probably need to write two functions (a main function that initializes the problem, and a helper function that is called with a matrix containing the

[1] Make sure you run `from mpl_toolkits.mplot3d import Axes3D` and `%matplotlib notebook` at some point. Then run `fig = plt.figure(); ax = fig.add_subplot(111, projection('3d')); ax.plot(''whatever you want'')`. Note that this does *not* work in Colab, and you need to have Jupyter notebook on your own computer for interactive 3D plots.

current set of orthogonal columns and adds a new column until the number of column equals the number of rows).

6. **Null and Range spaces.** Imagine you have a linear system characterized by matrix M, which takes as input a vector, $\vec{v}$, and outputs a vector, $\vec{y}$, such that $\vec{y} = M\vec{v}$. For this question, you may use `svd` or `np.linalg.svd`.

   (a) Explain in a few sentences what the null and range spaces of the matrix are.

   (b) Imagine that a creature has a linear tactile system: it takes a vector input (of pressure measurements) and produces a vector of neural responses. If the system has a non-zero null space, what does this tell you about the creature's perceptual capabilities?

   (c) Load the file `mtxExamples-2024.mat` into your MATLAB world (use `scipy.io.loadmat` in Python). You'll find a set of matrices named `mtxN`, with $N = 1, 2...5$. For each matrix, **use the SVD** to: (1) determine if there are non-trivial (i.e., non-zero) vectors in the input space that the matrix maps to zero (i.e., determine if there's a nullspace). If so, write a MATLAB or Python expression that generates a random example of such a vector, and verify that the matrix maps it to the zero vector; (2) generate a random vector y that lies in the range space of the matrix, and then verify that it's in the range space by finding an input vector, x, such that $Mx = y$. Please create a separate code cell for each matrix.