PSYCH-GA.2211/NEURL-GA.2201 – Fall 2023
Mathematical Tools for Neural and Cognitive Science

Homework 1

Due: 3 Oct 2023
(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. This includes functions `norm` and `inv`. You are welcome (encouraged, even!) to write your own functions to carry out these 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 component that lies along direction $\hat{u}$.

   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 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:

$$0 \longrightarrow [1, -5]$$

(b) System 2:

$$\begin{array}{rcl} [3, 2] & \longrightarrow & 15 \\ [-2, 2] & \longrightarrow & 6 \end{array}$$

(c) System 3:

$$\begin{array}{rcl} [2, 6] & \longrightarrow & [3, 7] \\ [1, -2] & \longrightarrow & [-1, 3] \\ [5, 0] & \longrightarrow & [1, 4] \end{array}$$

(d) System 4:

$$\begin{array}{rcl} [3, 1.5] & \longrightarrow & [-3, -6] \\ [-8, -4] & \longrightarrow & [8, 16] \end{array}$$

3. **Geometry of linear transformations**
For this question, you may use `svd` or `np.linalg.svd`.

(a) Write a function `plotVec2` that takes as an argument a matrix with 2 rows and n columns, and plots each column vector from this matrix on 2-dimensional axes. It should check that the matrix argument has two rows, signaling an error if not. Vectors should be plotted as a line from the origin (0,0) to the vector position, using a circle or other symbol to denote the "head" (see help for function 'plot'). It should also draw the x and y axes, extending from -1 to 1. The two axes should be equal size, so that horizontal units are equal to vertical units (read the help for the function 'axis'). Test this function by plotting three random vectors.

(b) Write a second function `vecLenAngle` that takes two vectors as arguments and returns the length (magnitude, or Euclidean-norm, not *dimensionality*) of each vector, as well as the angle (in radians) between them. Decide how you would like to handle cases when one (or both) vectors have zero length. Print the output of this function from two random vectors.

(c) Generate a random 2x2 matrix M, and decompose it using the SVD, $M = USV^T$. Now examine the action of this sequence of transformations on the two "standard basis" vectors, $\{\hat{e}_1, \hat{e}_2\}$. Specifically, use `vecLenAngle` to examine the lengths and angle between two basis vectors $\hat{e}_n$, the two vectors $V^T\hat{e}_n$, the two vectors $SV^T\hat{e}_n$, and the two vectors $USV^T\hat{e}_n$. Do these values change, and if so, after which transformation? Verify this is consistent with their visual appearance by plotting each pair using `plotVec2`.

(d) Generate a data matrix P with 65 columns containing 2-dimensional unit-vectors $\hat{u}_n = [cos(\theta_n); sin(\theta_n)]$, and $\theta_n = 2\pi n/64, n = 0, 1, \ldots, 64$. [Hint: Don't use a `for` loop! Create a vector containing the values of θ_n.] Plot a single blue curve through these points, and a red star (asterisk) at the location of the first point. Consider and discuss the action of the matrix M from the previous problem on this set of points. In particular, apply the SVD transformations 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 light at these 5 regions as a (nonnegative-valued) vector $\vec{v} = [v_1, v_2, v_3, v_4, v_5]$. The output r of the retina is a weighted sum of the intensities at each location; specifically, $r = 6v_1 + 8.2v_2 + v_3 + 3v_4 + v_5$.

   (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

[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.

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 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 `Hw1Q6_MtxExamples.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.