PSYCH-GA.2211/NEURL-GA.2201 – Fall 2021
Mathematical Tools for Neural and Cognitive Science

Homework 1

Due: 24 Sep 2021
(late homeworks penalized 10% per day)

See the course web site for submission details. Please: don't wait until the day before the due date... start *now*!

1. **Inner product with a unit vector.** Given unit vector $\hat{u}$, and an arbitrary vector $\vec{v}$, write (MATLAB or Python) expressions for computing:
   (a) the component of $\vec{v}$ lying along the direction $\hat{u}$,
   (b) the component of $\vec{v}$ that is orthogonal (perpendicular) to $\hat{u}$, and
   (c) 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, and don't forget 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) lies 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. **Testing for (non)linearity.** Suppose, for each of the systems below, you observe some example input/output pairs of vectors (or scalars). Determine whether each system could possibly be a *linear* system. If not, explain why. If so, provide an example of a matrix that is consistent with the observed input/output pairs, and state whether you think that matrix is unique (i.e., the *only* matrix that is consistent with the observations).

   System 1:
   [3, 2] $\longrightarrow$ 13
   [-1, 1] $\longrightarrow$ 3

   System 2:
   0 $\longrightarrow$ [3, -3]

   System 3:
   [5, 2.5] $\longrightarrow$ [-5, -10]
   [-1, -0.5] $\longrightarrow$ [1, 2]

   System 4:
   [1, 3] $\longrightarrow$ [3, 1]
   [1, -1] $\longrightarrow$ [-2, 2]
   [4, 0] $\longrightarrow$ [1, 6]

3. **Geometry of linear transformations**

   (a) Write a function `plotVec2` that takes as an argument a matrix of height 2, and plots each column vector from this matrix on 2-dimensional axes. It should check that the matrix argument has height two, signaling an error if not. Vectors should be plotted as a line from the origin to the vector position, using 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').

   (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.

   (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 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 full set of points (again, think of a way to do this without using a for loop!), plot them, and describe what geometric changes you see (and why).

4. **A simple visual neuron.** Suppose an experiment on a retinal neuron in a particular species of toad reveals that the responses are a weighted sum of the (positive-valued) intensities of light that is sensed at 5 localized regions of the retina. The weight vector is $[1, 4, 3.5, 2, 1]$. (a) Is this system linear? If so, express the response as a matrix multiplied by the input intensity vector. If not, provide a counterexample. (b) What unit-length stimulus vector (i.e., vector of light intensities) elicits the largest response in this neuron? Write a piece of code to compute this, and explain how you arrived at your answer. (c) What physically-realizable unit-length stimulus vector produces the smallest response in this neuron? Explain your reasoning. [hint: visualize a simpler version of the problem, in 2 dimensions]

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` 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.

   Write a MATLAB function `gramSchmidt` that takes a single argument, N, specifying the dimensionality of the basis. It should then generate an $N \times N$ 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). **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 $\vec{y} = M\vec{v}$. Explain in a few sentences what the null and range spaces of the matrix are. Now imagine a creature that 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?

   Load the file `mtxExamples.mat` into your MATLAB world (use `scipy.io.loadmat` in Python). You'll find a set of matrices named `mtxN`, with $N = 1, 2....$ For each matrix, **use the SVD** to: (a) 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; (b) 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 $Mx = y$.

[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.