STAT 339: HOMEWORK 0 (CODING, CALCULUS, AND MATRIX ALGEBRA WARMUP)

OPTIONAL, BUT AIM FOR WEDNESDAY 10/13

Instructions. This assignment is optional and will not be graded, but I will post solutions. You should definitely do it if you are rusty with programming, multivariable calculus, and/or matrix algebra, or could use a refresher (linear algebra is not a prerequisite for this class, but knowing a few basics will be helpful. There is a reference packet on the course website that includes some of the most important basic definitions.

You can do the whole thing if you want, or just pick and choose the parts you think will be useful to you (e.g., just the coding parts, just the math parts, or a bit of each).

You may use any language you like to do this assignment (and all assignments in this class) — the tasks are stated in a language-neutral way. I recommend Python if possible, but R would work fine too.

There are some resources for Python on the course webpage (under "Resources"). You may want to install the Anaconda environment to help keep your stuff organized, as well as an IDE such as Spyder or PyCharm. You will want the `numpy`, `scipy` and `matplotlib` libraries, and possibly also `pandas` which will most likely need to be installed separately (via Anaconda if you're using that).

Whatever language you are using, **you will certainly need to look things up** from time to time. Reference manuals are available, but I would start by just Googling: for example Google `python thingyouwanttodo`, where `thingyouwanttodo` might be `multiply matrices` or `histogram`. If you know the name of the function you want (e.g., `hist`) you can type `help hist` at the command prompt to see documentation on how to use the function.

You may also use any typesetting software to prepare your writeups, but final documents should be PDFs. LaTeXis highly encouraged.

For Python code, I prefer to receive `.py` script files rather than a format like a Jupyter notebook, though you are more than welcome to develop code in Jupyter and then export it to `.py`. Name your files something informative, and use modular design

Date: Last Revised: October 6, 2021.

where it makes sense (e.g., define helper functions in a separate file that you import from your main file).

0. **Create a project directory** for this homework to keep your files organized, and **set it to be your working directory**.

1. **Creating some simple plots.** Create each of the following plots and embed them in your writeup

   (a) Produce a plot of the floor function, $f(x) = \lfloor x \rfloor$ for x values in the range 0 to 10. (The floor function "rounds down" to the nearest integer.) Include axis labels. If using Python, I suggest the `matplotlib` library for plots.

   (b) Plot three different lines on the same plot (pick your own intercepts and slopes). Distinguish them by color and include a legend.

   (c) Produce a plot of the unit circle. (Hint: Create the $y > 0$ and $y < 0$ parts separately)

2. **Matrix manipulation and loops**

   (a) **Get data** for this exercise from my website: `http://colindawson.net/data/big_matrix.txt`. You can either read it directly into your program from the web, or first download the file into your working directory and read it in locally. One of these may be easier than the other depending on the language you are using. Save it as a matrix-valued variable called `A` (in Python, use the `numpy` matrix data type).

   (b) **Create integer-valued variables** `R` and `C` that contain the **number of rows and columns in** `A` (calculate this in their definition; don't hard code the values)

   (c) Normalize (shift and rescale) `A` so that the largest value becomes $1 - \varepsilon$ and the smallest becomes $0 + \varepsilon$ for some small positive constant ε. This can be accomplished by elementwise subtraction and division by a constant.

   (d) **Produce a grayscale image** whose pixels correspond to the entries of `A`, where 0 is black and 1 is white, and values in between are in between. You may find that most of the image shows up as black; if so, **apply a nonlinear transformation** to the individual entries in `A` to exaggerate differences near 0 and reduce them near 1. A **concave** function will have this behavior, but ensure that 0 stays 0 and 1 stays 1.

(e) Create a copy of `A` (e.g., `B`), and **write a loop** to replace the (i, j) entry of `B` by ε (or whatever your transformation returns for 0) whenever i and j are both evenly divisible by 5. **Visualize** `B` (you should see a lattice of black pixels that occupies 1/25 of the image in total).

(f) Make the entries that were previously ε (or transformed ε) into $1 - \varepsilon$ (or transformed $1 - \varepsilon$). Do this **without writing any explicit loops** (Hint: select the entries of interest by assigning directly to particular indices).

(g) **Write the new matrix to a file**.

3. **Creating and running a script.** Create a script file in your working directory and place in it all the code you need to read in and write out the matrices in problem 2. Verify that it runs by running it from a fresh workspace (or from the shell command line if not using an IDE).

4. **Creating a function.** In a separate file, define a function that does what you did in problem 2 with some more flexibility. Your function should take **three parameters**: the **name of the file to read** data from, the **name of a file to write** output to, and the **step size** for the lattice. It should also contain a clear **docstring** (a special comment inside the function definition the text of which displays if a user asks for help). Create a script file that gets the definition of your function from the first file and calls it with two different step sizes, writing to two different output files.

5. **Random numbers and Counting.** Define a function that simulates throwing two d-sided dice n times and returns the proportion of the time that the values on the dice sum to y over the n rolls. (In Python, use `scipy` or `numpy` random number generation functions) Your function should take d, n, and y as parameters, and return the sample proportion. (Optional: have your function provide default values for d and/or y) In a script, first set the seed of the random number generator to 42, then call your function 10 times in a row with $d = 6$, $n = 100$ and $y = 12$. How often did you get the same value? Set the seed to 42 again and repeat the 10 calls. How did the 10 values you got compare to the 10 values you got the first time? Explain why it might be useful to be able to control the random seed when testing and debugging probabilistic machine learning code.

6. **Vector Arithmetic.** Define a function that generates two 3×1 vectors (use `numpy` arrays in Python), called `a` and `b`, with random entries (your choice of distribution), and returns a list/tuple/structure consisting of the following quantities: a, b, $a + b$, $a \circ b$, and $a \cdot b$ ($\circ$ denotes the elementwise product and $\cdot$ denotes the dot product, defined as $a \cdot b = a^\top b := \sum_{i=1}^{3} a_i b_i$). Set the random seed to 37 and call your function, including the five output values in your writeup in monospace text (use a `verbatim` environment if using LaTeX) and in formatted mathematical notation.

7. **Matrix Arithmetic.** Create a modified version of your function from the last problem that also creates a random 3×3 matrix, `X`, and returns a list/tuple/struct consisting of a, b, X, $a^\top X$, $a^\top X b$, X^{-1} and $(X^\top X)^{-1}$. Set the seed to 37 again and call your function, including the output values in your writeup in two ways as above. Verify for one representative element of the output that matrix multiplication worked as it should; namely that $(AB)_{ij} = \sum_{k=1}^{3} A_{ik} B_{kj}$, where A_{ij} is the entry in row i and column j of the matrix A.

8. **Calculus Warmup.** (No coding for this problem) Consider the following function of the three variables x, μ and σ:

$$f(x, \mu, \sigma) = (2\pi\sigma^2)^{-1/2} \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right)$$

   (a) Find an expression for the partial derivative $\frac{\partial f}{\partial x}$.

   (b) For fixed values of $\mu \in \mathbb{R}$ and $\sigma > 0$, find the value of x that maximizes f.

   (c) Define $h = \frac{1}{\sigma^2}$ and find an expression for the partial derivative $\frac{\partial f}{\partial h}$.

   (d) Define a new function g as follows:

$$g(\mu, \sigma; x_1, x_2, x_3) = \sum_{i=1}^{3} \log f(x_i, \mu, \sigma)$$

   where x_1, x_2 and x_3 are treated as constants and log is the natural logarithm. Find the gradient $\nabla g = (\frac{\partial g}{\partial \mu}, \frac{\partial g}{\partial \sigma})$. (The gradient is just a vector containing the partial derivatives with respect to each variable)

(e) Using the gradient you found in the previous step, find the vector (μ^*, σ^*) that maximizes g. (Hint: set the gradient to the zero vector and solve resulting the pair of equations for μ and σ)

9. **Matrix Algebra Warmup.** (No coding for this problem)

(a) (FCML Exercise 1.3) Show that it follows from the definition of matrix multiplication that

$$\mathbf{w}^\top \mathbf{X}^\top \mathbf{X}\mathbf{w} = w_0^2 \left(\sum_{n=1}^{N} x_{n1}^2 \right) + 2w_0 w_1 \left(\sum_{n=1}^{N} x_{n1} x_{n2} \right) + w_1^2 \left(\sum_{n=1}^{N} x_{n2}^2 \right),$$

where

$$\mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}, \mathbf{X} = \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \\ x_{31} & x_{32} \\ \vdots & \vdots \\ x_{N1} & x_{N2} \end{bmatrix}.$$

See the packet of linear algebra definitions on the webpage if you don't know or have forgotten how to multiply matrices. You can do this problem mechanically by just carrying out the multiplications one at a time and keeping track of the resulting matrices. Even if you have not studied any linear algebra!

(Hint – it's probably easiest to do the $\mathbf{X}^\top \mathbf{X}$ first, which will allow you to work with smaller matrices! Matrix products are associative but not commutative!)

(b) Using $\mathbf{w}$ and $\mathbf{X}$ as defined in the previous exercise, verify that $(\mathbf{X}\mathbf{w})^\top = \mathbf{w}^\top \mathbf{X}^\top$ by multiplying out both sides.