CS2810 Mathematics of Data Models	Gold
Spring 2019	February 6, 2019

Homework 04

Due: February 13, 9PM
Point total: 60

Instructions:

- Submit your PDF and/or .py file to Blackboard by the due date and time. Please do not zip your files together, as this interferes with Blackboard's preview functionality. Always show all your work, and for full credit, you must use the method that the problem instructs you to use (unless none is mentioned). Handwritten or typeset solutions are both acceptable, but unreadable submissions will be penalized. You may discuss problems with other students, but you may not write up solutions together, copy solutions from a common whiteboard, or otherwise share your written work or code. Do not use code or language that is copied from the Internet or other students; attribute the ideas *and* rephrase in your own words.

Problem 1 (12 points [6,4,2])

You are programming logic for a video game character in a 2D space, and want the character to shoot at targets only when the shot would hit some part of the target. If the targets are circles, we can do this using projections; the line of fire is a line, and the point where it comes closest to the targets center can be found by projecting the vector to the target's center onto the line of fire. If this distance is less than the radius of the target, the shot will hit.

i. Suppose the player is at x, y coordinates (1,2), with a unit vector describing their facing (and firing direction) of $\begin{bmatrix} 1/2 \\ \sqrt{3}/2 \end{bmatrix}$ (that is, a 60 degree angle from parallel to the x-axis). The target's center is at coordinates (11, 22). If a shot is fired at the stationary target, what point in the bullet's trajectory will be closest to the target's center? (You may find it useful to think of the player as the origin when performing the projection, but don't forget to switch back to the original coordinate system when answering.)

Solution: The vector from player to target is $\vec{v} = \begin{bmatrix} 10 \\ 20 \end{bmatrix}$. Projecting this onto the facing vector gives $\begin{bmatrix} 5/2 + 5\sqrt{3} \\ 5\sqrt{3}/2 + 15 \end{bmatrix}$ which is roughly $\begin{bmatrix} 11.2 \\ 19.3 \end{bmatrix}$ Adding this to the original location of the player gives the location (12.2, 21.3).

ii. Calculate the distance of the point you found from the target's center. If the radius of the target is 2, will the shot hit the target?

Solution: $\sqrt{(12.2-11)^2 + (21.3-22)^2}$ = roughly 1.4, less than the radius of 2. The shot will hit the target.

iii. Would it work equally well to program an agent to fire only when the angle between facing and target is less than a fixed threshold? (You can assume all targets are the same size, but not always at the same distance.) You don't need to prove your answer, but provide either an explanation or counterexample.

Solution: No, the threshold angle should logically change with the distance to the target. When it looms very close, many angles would do, while when it's far away, only a tiny angle would work from the player's perspective.

Problem 2 (12 points [8,4])

i. Project the vector $\vec{v} = \begin{bmatrix}10\\20\\30\end{bmatrix}$ onto the plane defined by the basis vectors $\vec{\beta_1} = \begin{bmatrix}1\\1\\0\end{bmatrix}, \vec{\beta_2} = \begin{bmatrix}0\\1\\1\end{bmatrix}$.

Solution: We have $A = \begin{bmatrix}1 & 0\\1 & 1\\0 & 1\end{bmatrix}, A^T = \begin{bmatrix}1 & 1 & 0\\0 & 1 & 1\end{bmatrix}$.

This means $A^TA = \begin{bmatrix}2 & 1\\1 & 2\end{bmatrix}$. Using Gauss-Jordan to solve for $(A^TA)^{-1}$ gives $\begin{bmatrix}2/3 & -1/3\\-1/3 & 2/3\end{bmatrix}$.

Multiplying out all the matrices to get $A(A^TA)^{-1}A^T$ results in the matrix $\begin{bmatrix}2/3 & 1/3 & -1/3\\1/3 & 2/3 & 1/3\\-1/3 & 1/3 & 2/3\end{bmatrix}$

and multiplying this by the original vector $\vec{v}$ gives the projection $\begin{bmatrix}10/3\\80/3\\70/3\end{bmatrix}$

ii. Demonstrate that your answer to the previous part is correct by showing that the vector is a linear sum of the basis vectors, and showing that the difference from $\vec{v}$ is orthogonal to both basis vectors.

Solution: It's easy to check that the vector is the sum of $(10/3)\vec{\beta_1} + (70/3)\vec{\beta_2}$. The difference from $\vec{v}$ is $\begin{bmatrix}20/3\\-20/3\\20/3\end{bmatrix}$ and we can show that the dot product with either basis vector is $-20/3 + 20/3 + 0 = 0$.

Problem 3 (12 points)

You are a manager at a software firm trying to make the case that hiring more senior developers makes sense financially. Alice, at 4 years experience, had only three bug bug reports filed against her last year. Bob, at 2 years experience, had twenty bug reports filed against him last year. Cid, at 1 year experience, had 29 bug reports filed against his code. Dalia, at 3 years, had 7 bugs filed. Fit a linear model to this data of the form $b = my + z$, where b is the number of yearly bugs, y is the programmer's experience in years, and m and z are the constants you find.

For this problem, feel free to show your work by multiplying and inverting matrices in Python instead of doing this by hand, turning in either a function or a log of an interactive session; but, do not call a method that simply solves the least squares problem for you.

Report coefficients to 1 decimal place.

Solution: We want to create a matrix $A = \begin{bmatrix} 4 & 1 \\ 2 & 1 \\ 1 & 1 \\ 3 & 1 \end{bmatrix}$ then calculate $(A^T A)^{-1} A^T \vec{v}$ where $\vec{v} = \begin{bmatrix} 3 \\ 20 \\ 29 \\ 7 \end{bmatrix}$.

We can do this in a few lines of Python at the interactive prompt:

```
>>> A = [[4, 1], [2,1], [1,1], [3,1]]
>>> Am = np.array(A)
>>> v = np.array([3, 20, 29, 7])
>>> c = np.matmul(np.matmul(np.linalg.inv(np.matmul(Am.T,Am)), Am.T), v)
>>> c
array([-9.1, 37.5])
```

Thus the linear model is $b = -9.1y + 37.5$.

Problem 4 (12 points, 6 each)

i. Using the determinant and Gauss-Jordan elimination, find the eigenvalues and eigenvectors of the matrix $\begin{bmatrix} 1 & 3 \\ 2 & 2 \end{bmatrix}$.

Solution: Using the fact that the determinant of $\begin{bmatrix} 1-\lambda & 3 \\ 2 & 2-\lambda \end{bmatrix}$ must be 0, we can solve to get $\lambda = 4$ or $\lambda = -1$. Plugging these into the matrix and applying Gauss-Jordan, we get that the eigenvector for $\lambda = 4$ is $\begin{bmatrix} 1 \\ 1 \end{bmatrix}$ and the eigenvector for $\lambda = -1$ is $\begin{bmatrix} 3 \\ -2 \end{bmatrix}$.

ii. Suppose we know that a character in a game can be in 3 states, Angry, Boastful, or Charismatic, and the chance of the character being in each state the next day depends on which state they are in for the current day. Given a vector of probabilities p_A, p_B, p_C that the character is in a particular state on day i, the probabilities of being in each state the next day are given by

$p'_A = 0.6p_A + 0.1p_B + 0.2p_C$
$p'_B = 0.1p_A + 0.7p_B + 0.2p_C$
$p'_C = 0.3p_A + 0.2p_B + 0.6p_C$

You know that the "stationary distribution" is the vector of probabilities that remains unchanged when passed through these equations; this is also how much of the time the character spends in each state overall.

i. What eigenvalue can we assume for the vector representing the stationary distribution?

Solution: 1.

ii. Solve for the corresponding eigenvector direction.

Solution: Using Gauss-Jordan elimination on the matrix $\begin{bmatrix} -0.4 & 0.1 & 0.2 \\ 0.1 & -0.3 & 0.2 \\ 0.3 & 0.2 & -0.4 \end{bmatrix}$ gives the family of vectors $\begin{bmatrix} 8/11 \\ 10/11 \\ 1 \end{bmatrix} z$.

iii. Use the fact that the probabilities must sum to 1 to find the stationary distribution.

Solution: Solving $(8/11)z + (9/11)z + z = 1$ gives $\begin{bmatrix} 8/29 \\ 10/29 \\ 11/29 \end{bmatrix}$.

Problem 5 (12 points [8, 4])

For this part, download `leastsq.py` and the datasets `uspop.txt` and `uspop1700_1900.txt`.

The main thing you'll be doing here is fitting an exponential model to the data in `uspop1700_1900.txt`, which records the U.S. population from census results between the years 1700 and 1900. The other file, `uspop.txt`, contains data that extends further back into the 1600's and all the way to 2010, but a single exponential curve won't fit that data well. So you will mostly be working with `uspop1700_1900.txt`, except at the end.

`leastsq.py` contains example code for performing a least-squares regression where the model is a line, $y = mx + b$. As you will see if you try this code on our dataset, a straight line is not a great fit for the U.S. population growth between 1700 and 1900. Your job is to fill in the additional functions, `exponential_fit()` and `plotcurve()`, so that you instead fit and visualize a model that takes the form $y = Ce^{ax}$. (Take the log of both sides, and now you are fitting $\ln y = \ln C + ax$. Treat the natural logs of your observations as your observations, find $\ln C$ and a, and finally find the constant $C = e^{\ln C}$.)

i. Complete the `exponential_fit()` function and `plotcurve()` function, and include in your assignment PDF the curve you have plotted to fit the data in `uspop1700_1900.txt`. (This should look like a pretty reasonable fit.) Also report the coefficients you found to at least two significant digits.

ii. Plot years versus log population using the data from `uspop.txt`, and include this figure as well in your submission. (You can either pass log values to the provided linear fit functions, or plot the values yourself.) How can we tell 1700-1900 is a particularly good fit for an exponential model of the form $y = Ce^{ax}$, compared to times near the beginning or end of the range?

Appendix: Python Tips for Problems 3 and 5

Some Python you might want to know or recall:

i. `years = data[:,0]` grabs a column of the array "data"; the colon in the first position indicates that we want all rows. (This is called "slicing.")

ii. In numpy, `mymatrix.T` is the transpose of mymatrix. It won't work on a list; you need to convert to an array with `np.array()`.

iii. `np.vstack()` vertically stacks vectors, and can build a matrix row by row - or column by column, if the transpose is found immediately after.

iv. `m, b = np.linalg.lstsq(A, pops, rcond=None)[0]` is an example of solving a least squares problem. The matrix A is the same as the matrix used by hand when finding the projection; it doesn't add the ones column for you, for example, if solving for $y = mx + b$. `pops` here is the vector to be predicted, the population vector. The [0] indicates that we care only about the coefficients of the model, which together are the zeroth return value, and not the other things this function could return such as the error. `rcond` is a parameter related to handling numerical imprecision, and we're just not going to touch it.

v. Recall that `**` can be used to perform exponentiation.

vi. Recall that operations like `np.log()` can often be used to apply to the whole input array at once.