CS 111: Homework 5: Due by 11:59 pm Sunday, February 19, 2023

Submit your paper as one PDF file, and tell GradeScope which page(s) each problem is on. If you worked with a partner, you must each turn in your own homework paper, and report the name and perm number of your partner. No groups of more than two allowed.

1. This problem is based on Problem 10.14 in the NCM book. Please read that problem for background. The origin of the problem is a curious application of singular value decomposition to cryptography that Moler and Morrison described in a 1983 paper, which is on the course GitHub site under `Homework/h05` if you're interested. The first part of this problem is to duplicate the plot shown in NCM Problem 10.14 by writing a python/numpy version of NCM's `digraph.m` Matlab code (also on the course GitHub).

The goal is to (approximately) divide the letters in a piece of English text into vowels and consonants. The idea is to form a matrix called the "digraph matrix" from the input text, and compute its SVD. The digraph matrix A is 26-by-26, with one row and column for each letter of the English alphabet, where we think of the letters A through Z as being numbered from 0 through 25. The matrix entry `A[i,j]` is the count of the number of times letter i is followed immediately by letter j in the text. The SVD of A lets us compute low-rank approximations to A. As explained in NCM Problem 10.14, we look at the first two terms in the low-rank approximation, which are $\sigma_0 u_0 v_0^T$ and $\sigma_1 u_1 v_1^T$. The σ_0 term mostly depends on how often each letter of the alphabet occurs in the text.

Moler and Morrison's curious discovery was that the σ_1 term captures the *alternation* of vowels and consonants in English—the fact that, more often than not, a vowel is followed by a consonant and a consonant is followed by a vowel. In particular, the signs of the entries in the *second* singular vectors u_1 and v_1 express the alternation. If you plot each letter of the alphabet in the u-v plane, using one element of u_1 as the first coordinate and one element of v_1 as the second coordinate, you will see (as in the NCM figure) that most of the vowels (A, E, I O) show up in the upper left quadrant, and most of the consonants show up in the lower right quadrant. The NCM problem has more discussion of this, and explains why the division into vowels and consonants isn't quite perfect.

A couple of details: When you read in the text to analyze it, you will ignore all whitespace and punctuation, so you'll convert it to just a single long string of letters. You'll also convert all the letters to upper case, so E and e are treated as the same. Finally, you'll consider the whole text to be cyclic, so the last letter in the text is followed by the first letter.

1.1. Write Python code to duplicate the plot given in NCM Problem 10.14, using as input the text of Lincoln's Gettysburg address from the file `gettysburg.txt`, which you can find on the course GitHub in directory `Homework/h05`. That directory also has the original Moler/Morrison paper and a Jupyter notebook with examples of how to do some things you'll need in python, numpy, and matplotlib. Turn in your plots and code.

1.2. Find another English text, longer than `gettysburg.txt`, and use your code to make a corresponding plot. Do you see similar or different behavior? Turn in your plots and code, and your conclusions.

1.3. Find a text in another language, at least as long as `gettysburg.txt`, and use your code to make a corresponding plot. You may have to modify the number of letters in the alphabet. Say what language you chose, and whether the behavior is similar to English or different. Also turn in your plots and code. (The NCM problem suggests trying Hawaiian and Finnish, which have somewhat different statistics than English, but you can use any language you want.)

2. This problem is about using the least squares method for fitting a function of a given form to data from measurements. The two main goals here are (1) to learn to use the numpy routine `npla.lstsq()`, and (2) to learn to make well-designed plots of the data and the least-squares fits to the data. We'll talk about the algorithms behind `npla.lstsq()` in class later this week, and you'll experiment with them on next week's homework.

2.1 The data we'll use is NASA's annual measurements of the so-called "temperature anomaly," which is how much the mean global temperature for a given year differs from the average for the reference years 1951-1980. You can read about this data at `https://data.giss.nasa.gov/gistemp/`. For our purposes, I've downloaded the data as a JSON file called `annual_temps.json` that's with the homework on our GitHub site.

Use the routine `cs111.get_gistemp()` to read the temperature data. It returns a numpy array of years (from 1880 to 2016), and an array of Celsius temperatures. Make a graph that plots the data as dots, one per year, with the years on the horizontal axis extending as far as 2040. Label the axes with a description of what they represent (year and temperature anomaly); put a good descriptive title on the plot; and include a legend (using `plt.legend()`) indicating that the dots are the measured data. Turn in your plot and the python code you used to produce it.

2.2 Here we'll fit a straight line to the data, of the form

$$f(y) = x_0 + x_1 y,$$

where y is the year. We want to choose x_0 and x_1 so that, if the measured temperature in year y_i was t_i, we have $f(y_i) = t_i$. However, the 137 points (y_i, t_i) don't all lie on a straight line, so we can't satisfy this exactly for all i. We will find the values of x_0 and x_1 that are best in the *least squares* sense, meaning that they make the value of the squared residual norm

$$\sum_i (f(y_i) - t_i)^2$$

as small as possible.

Let's say that in terms of matrices and vectors. Let t be the column vector $(t_0, \ldots, t_{136})^T$ of measured temperatures, one per year. Let A be a matrix with two columns, with one row for each year. In row i of A, the first entry is a 1 and the second entry is the year y_i. Then if $x = (x_0, x_1)^T$ is the 2-vector of unknowns, the product Ax is the vector of the 137 values $f(y_i)$, which we want to be as close to t as possible. Thus the problem we need to solve is to find the vector x that minimizes the 2-norm of the residual,

$$\min_x ||Ax - t||_2.$$

This is exactly the problem that `npla.lstsq(A, t, rcond = None)` solves. (The `rcond = None` just suppresses a warning message about different versions of the routine.)

Use `npla.lstsq()` to find the minimizing x for the NASA data from 1880 to 2016. Make another plot, which has the same data points as your plot from (1.1), and also has the straight line representing $t = f(y)$. Plot the straight line all the way from $y = 1880$ to $y = 2040$. Update the legend to describe what the line is in addition to the dots. Turn in your plot and the python code you used to produce it. Report the value of x. Finally, report what the temperature anomaly would be in 2040 if it were on the least-squares fit line.

2.3. You might notice that linear extrapolation along the least squares line badly undershoots the actual temperature anomalies in recent years, so the predicted 2040 value you computed above is not very credible. Make a third plot that adds two more straight lines to your plot from (2.2), one of which fits only the data since 1970 and one of which fits only the data since 2010. Plot both of the new lines all the way out to 2040, and add descriptions of them to the legend. (This plot should still show all the data points from 1880 on, as well as the original line you computed in (2.2).) Turn in your plot and the python code you used to produce it. Report the values of x for each new line, and report the predicted 2040 temperature anomaly according to each line.

2.4. Although these latest predictions look somewhat better than the first one, it really doesn't seem like the data follows a linear trend. Let's try fitting a quadratic (a parabola) to the data, of the form

$$f(y) = x_0 + x_1 y + x_2 y^2.$$

At first glance this doesn't look like a "linear" least squares problem, but it is, because the unknowns x_j appear linearly. Thus we can solve it in a matrix formulation as before, by finding the minimizer

$$\min_x ||Ax - t||_2.$$

This time x is a 3-vector $(x_0, x_1, x_2)^T$, and A is a matrix with three columns. What's in the third column of A? The squares y_i^2, which are the coefficients of x_2 in $f(x) = Ax$.

Use `npla.lstsq()` to find the 3-vector x that describes the quadratic least-squares fit to the data, using all the years from 1880 to 2016. Make one more plot that shows the data (as dots), the 1880-2016 linear fit (as a line), and the 1880-2016 quadratic fit (as a parabola). (To make the graph less noisy, you can omit the other two linear fits you found in (2.3).)

Hint: To draw the parabola $f(y) = x_0 + x_1 y + x_2 y^2$ over the range 1880-2040, you can plot a piecewise linear curve that goes through the values of $f(y)$ for a suitably chosen set of y values. You can use `np.linspace()` (see its docstring) to generate evenly spaced y values as close together as you want.

Turn in your suitably labelled plot and the python code that produced it. What are the values of the quadratic coefficients x? What does the quadratic fit predict as the temperature anomaly in 2040?