CS 111: Homework 5: Due by 11:59 pm Sunday, October 31, 2021

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 separately write up and turn in your own homework paper, and report the name of your partner. No groups of more than two.

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, slightly 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.

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?

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. (The NCM problem specifically suggests trying Hawaiian and Finnish, which have somewhat different statistics than English, but you can use any language you want.)

2. Consider each of the following python loops. For each loop, answer: How many iterations does it do before halting? What are the last two values of x it prints (both as decimals printed by python, and as IEEE standard 16-hex-digit representations as printed by `cs111.print_float64`)? For each loop, explain in one English sentence what property of the floating-point system the loop's behavior demonstrates.

2.1.

```
x = 1.0
while 1.0 + x > 1.0:
    x = x / 2.0
    print(x)
```

2.2.

```
x = 1.0
while x + x > x:
    x = 2.0 * x
    print(x)
```

2.3.

```
x = 1.0
while x + x > x:
    x = x / 2.0
    print(x)
```