The University of Texas at Austin

COE 301 Introduction to Computer Programming Fall, 2024

Homework 10 – Due: 11/13/2024 11:59 pm

Problem 1. (35 points) *Grade calculator II.* Gives the following vectors that contains your COE 301 records:

```
homework = [96; 105; 72; 85; 85; 90; 97; 95; 92; 80; 98; 91];
quiz = [5; 7; 5; 8; 3; 5];
exam =[76; 86]; % midterm is the first score
weight1 = [0.40; 0.1; 0.15; 0.35];
weight2 = [0.40; 0.1; 0.05; 0.45];
```

(1) The first vector **homework** is a m-by-1 vector that represents scores of m homework sets. The lowest score is excluded in the average homework score. if this number is larger than 100, set the average quiz score to be 100.
(2) The second vector **quiz** is a (n+1)-by-1 vector that represents the points you have earned from n quizzes and the total bonus points you have earned from class surveys (this is the last element in the vector). The average quiz score is calculated as: first calculate the **totalPoints** you have earned from your highest (n-1) quizzes plus bonus points, then calculate the average quiz score as **totalPoints/(n-1)/8*100**. if this number is larger than 100, set the average quiz score to be 100.
(3) The third vectror **exam** is a 2-by-1 vector that represents the scores of your mid-term and your final on a 100-piont scale.
(4) The fourth vector **weight1** is a 4-by-1 vector the represents the weight on homework, quiz, mid-term exam and final exam based on our original weighting strategy.
(5) The fifth vector **weight1** is a 4-by-1 vector the represents the weight on homework, quiz, mid-term exam and final exam based on our alternative weighting strategy.

You may use the MATLAB function sort as:
https://www.mathworks.com/help/matlab/ref/sort.html

Write a MATLAB script to calculate the weighted score based on the two weighting strategies **without any loops**. Do not hard-code the length of these vectors (assume m and n can change).

Submit your .m file and report the two weighted scores in the write up.

Problem 2. (35 points) We have learned in the class that the inner product of a vector with itself gives the square of its magnitude:

$$|x| = \sqrt{x^T x}$$

We can define a similarity measure of two column vectors x and y as:

$$\text{similarity}(x, y) = \frac{(x - \overline{x})^T (y - \overline{y})}{\sqrt{(x - \overline{x})^T (x - \overline{x})} \cdot \sqrt{(y - \overline{y})^T (y - \overline{y})}}$$

,where $\bar{x}$ and $\bar{y}$ are the mean of vector x and y. The similarity number ranges from -1 to 1. 1 means the vectors x and y are 100% correlated, 0 means the vectors x and y are not correlated at all and -1 means the vector are anti-correlated (when one goes up, the other goes down and vice-versa).

Use the similarity measure to discover which stock has very similar price movement with AAPL. The historical daily stock price for AAPL, MSFT, QCOM and WFC can be downloaded under the folder 'Lab10_data'. You may load the text file that contains the daily stock prices into a vector using the MATLAB command:

```
            aapl = load('aapl.txt');
```

Plot the price time series of all four stocks on the same figure and comment on whether you can observe the similarity from the plot easily.

Submit your .m file as "yourLastName hw10 prob2.m". Report your figure (save your figure as a .tif or .jpeg file and insert the figure in the write-up. A screenshot of the figure is ok as well), your similarity scores, and your comment in the write-up.

Problem 3. [30 points] MATLAB is a powerful tool for engineers. In this problem, we will show you how to manipulate music data (a vector!) using MATLAB. Let's first read in an audio file 'testaudio.wav' under the folder 'Lab10_data' and put the data in a vector x. The sampling rate (44 kHz) is in the variable Fs.

```
      [x, Fs] = audioread('testaudio.wav');
```

(1) Plot a subset of the data

```
      plot(x(100000:200000))
```

You may zoom in and drag the figure using the icon 🔍 🔍 ✋ above the figure. Find an interesting portion, report the **zoomed-in** figure in the write-up.

(2) Looking at audio data in this way is not very useful. A common type of plot used by audio engineers is a spectrogram (https://en.wikipedia.org/wiki/Spectrogram). This is a 2D plot with time on the x axis and frequency on the y axis. Bass sounds are near the bottom and high-pitched sounds are near the top. Time progresses from left to right. Plot the spectrogram of the audio using the following command and report this spectrogram figure in the write up.

```
      specgram(x, 1024, Fs);
```

Note: you will need to have the signal processing toolbox installed in order to use this function. If you haven't selected this toolbox during the initial installation, on the main command interface, you can use Add-Ons -> Get Add-On, and use the Add-On Explorer to find the Signal Processing Toolbox.

Zoom in on the section from 10.6 to 11.6 seconds in time and 6000 Hz to 7000 Hz in frequency. Describe qualitatively and (as best as you can) quantitatively what you see. What caused this? What would this sound like if you could isolate only this portion and play it? Please include those discussions in your write-up.

(3) Play the audio and describe what you hear. ***This audio has been corrupted by very annoying noise so you will want to turn the volume down and use headphones to avoid disturbing others.***

```
      sound(x, Fs);
```

In audio processing, *digital filters* are used extensively to manipulate the sound of audio recordings. We will use one very basic filter called a *notch filter* here to try to eliminate the annoying sound:

```
% Filter coeffients
b = [ 0.999643720498015  -1.918666629446868 0.999643720498015 ];
a = [ 1.000000000000000  -1.918666629446868 0.999287440996029 ];
y = filter(b, a, x);
sound(y,Fs);
```

Describe how the sound has changed in the write-up. Plot the spectrogram of y in the same way as before and comment on what have changed. Notice that a small portion of the noise is still present at the beginning of the recording. How long does it persist (in seconds) before it becomes unnoticeable? Report the spectrogram figure of y and the discussions in the write-up.

Extra credit [+10 points]: There is a special coded text message embedded in this audio recording. Find it and report it. Describe your work to receive full credits. We will not answer questions about the extra credit problem in TA office hours, and you may reach out to Ann Chen for hints.

Submission Instructions:

There should be 4 files in your submission:

1. A write up (any type- .txt, .docx, .pdf are all fine) that contains your answers to all questions in problem 1-3.
2. The .m file for problem 1.
3. The .m file for problem 2.
4. The .m file for problem 3.

Make sure your last name is included in the filename.

Optional Short answers questions. The following questions will not be graded. You may use them for preparing your next week's quiz.

(1) Given that

```
A = [1  2  3  4; 5  6  7  8; 9  10  11  12]

x = [2; 1; 0; 1]
```

What is the output of the following expressions?

```
a) A(2, end)
b) A(1, 2)
c) A(4)
d) A(:, 1:2:end)
e) A(1:2, 2:4)
f) max(A)
g) min(A(:))
h) x(end)
i) x(2:2:end)
j) A*x
```

(2) Let x be a 1x2 row vector, and v is a 2x1 column vector.

```
x = [1  4]
y = [2; 1]
```

What is the result of the following expression? Please first compute the results by hand and verify them using MATLAB (**please be careful on which one use element-wise multiplication**).

```
a) x * y
b) y' .* x
c) y * x
```

(3) what is the output of the following block of MATLAB code? Please first compute the results by hand and verify them using MATLAB.

```
x = [2;-5;0;-8];
A = eye(length(x));
disp(A*x);
```

```
    B = [1:4; ones(1,4)];
    C = [x x*2];
    disp(B);
    disp(C);
    disp(B*C);
```

(4) Please generate a vector x as 201 points evenly spaced between -1 and 1 using the MATLAB function `linspace`. Please generate another vector y based on $y = \frac{1}{x^3+1}$ **without** using loops, and plot `y vs. x`.

(5) what is the output of the following block of MATLAB code?

```
    n = 4;
    D = diag(ones(n,1))+diag(-1*ones(n-1,1),1);
    D(end,:) = [];
    x = [1; 3; 6; -9];
    disp(D*x)
```

(6) Generate a 5000-by-900 matrix A and a 900-by-1 vector b (the elements can be any random numbers) and answer the following questions:

(a) Generate a matrix C that contains all columns and all even rows of A **WITHOUT using loops**.

(b) What is the dimension of `A*b`?

(c) How do you use a single index (linear indexing) to access the element in the second row and the second column of A?

(d) Find the largest number in A **WITHOUT using loops**.

(e) Generate a vector d that contains all positive elements in b **WITHOUT using loops**.

(f) Remove the last 10 columns in A and store this new matrix in M **WITHOUT using loops**.

(7) Given a 5000-by-900 matrix A, a 900-by-1 vector b, and a 1-by-900 vector of c, what is the dimension of the following matrices/vectors?

```
B = [A; c];

C = [b c’];
```

```
D =[b’; c];
```

*****Extra optional practices*****

Short problems to practice MATLAB syntax. Create one MATLAB script, which will have several parts. Note: If you separate parts of a script with “%%”, it will create a “section”, which you can run separately with Ctrl-enter or clicking “Run section”. **We will be giving most of the answers here simply as a way for you to practice using these new commands.**

```
%% MATLAB Indexing practice
```

Let's practice selecting parts of a matrix using indexing notation of MATLAB.

First, make a multiplication table from 1 to 5 along the rows, and 1 to 7 along the columns:

```
    a = 1:5;
    b = 1:7;
    disp('a (5x1) vector times a (1x7) vector')
    disp('The result is a (5x7) matrix')
    A = a' * b;
    disp(A)
```

Now to start indexing, remember that its "A(rows I want, columns I want)".

So the 4th row, 5th column is

```
    A(4, 5)
```

Print that element out.

To get multiple elements at once, we'll use colons.
Take the top 3x3 box:

```
    A(1:3, 1:3)
```

Notice that the indexing looks the same as when we just create a vector from 1 to 3.
MATLAB any vector of integers and gives you back the elements at those indexes.

Note that the indexes don't have to be in order!
Let’s look at the 2nd row, and get the 1st, then the 5th, then the 4th column:

```
    A(2, [1 5 4])
```

% Use the "end" keyword when you want to go all the way to the end
% Take the very last row, last column:

```
    A(end, end)
```

When you see only a colon for an index, it's really just a shorter way to write "1:end"
It means "select all the elements for this index"
To take the 3rd column, we want to use indexing that says
"Take all rows, and only column 3"

```
    disp('3rd column:')
    A(:, 3)
```

This is the same as A(1:end, 3)

```
      A(1:end, 3)
```

You can use that if you'd like to make it clearer, but you'll see just a colon much more often

Print the bottom row:

```
      disp('Bottom row')
      A(end, :)
```

Now let’s find matrix elements that meets a condition.

Use the relational less than operator, <, to determine which elements of A are less than 9. Store the result in B:

```
        B = A < 9;
        disp(B)
```

The result is a logical matrix. Each value in B represents a logical 1 (true) or logical 0 (false) state to indicate whether the corresponding element of A fulfills the condition A < 9. For example, A(2,1) is 2, so B(2,1) is logical 1 (true). However, A(5, 7) is 35, so B(5,7) is logical 0 (false).

You can use the find function to locate all of the elements in A less than 9.

```
        index = find(A < 9);
```

The result is a column vector of linear indices. Each index describes the location of an element in A that is less than 9. As we discussed in class, we can access elements of an array using a single index, regardless of the size or dimension of the array. This method is known as *linear indexing*. While MATLAB displays arrays according to their defined sizes and shapes, they are actually stored in memory as a single column of elements.

To display all the elements in A that are less than 9 as a column vector, you may use

```
      disp(A(B))
```

or

```
      disp(A(index))
```

So in practice A(index) returns the same result as A(B). The difference is that A(B) uses logical indexing, whereas A(index) uses linear indexing.

NEW SECTION!

%% Practice with matrix multiplication

Clear the value saved for A

```
      clear A
```

In your linear algebra classes, you will get plenty of doing matrix multiplication by hand. This problem is supposed to get you a better feel for which you can legally/sensibly multiply, and which don't make sense.