Programming Assignment

Ellie McGregor
60722584

Problem 1: The Power of Matrices (20 points)

1.1 First things first, let's implement the recursive implementation of the Fibonacci sequence. We have set up the recursive function for you, and your job is simply to fill in a couple of lines in here that will perform the recursion itself. For those of you who need a reminder, the idea of recursion is based on the idea that functions can call themselves until you reach a base case. Please paste the code for your solution here. [5]

```
def fib_recurse(i):
    # Base cases
    if (i==1):
        fib_num = 1;
    elif (i==2):
        fib_num = 1;
    else:
        # Recursion
        fib_num = fib_recurse(i - 1) + fib_recurse(i - 2) # TODO - FILL THIS IN
    return fib_num
```

1.2 Now, let's consider how we might turn this into a matrix implementation. To help you out, we'll walk you through the thought process to turn the recursive equation into something you can do with matrices. Please paste the code for your solution here. [10]

$$\begin{bmatrix} fib(n) \\ fib(n-1) \end{bmatrix} = \begin{bmatrix} fib(n-1) + fib(n-2) \\ fib(n-1) \end{bmatrix}$$

n = 2, m_times = 1

n = 3, m_times = 2

n = 4, m_times = 3

$$\begin{bmatrix} fib(n) \\ fib(n-1) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} * \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} fib(n) \\ fib(n-1) \end{bmatrix} = \begin{bmatrix} 1+1 \\ 1+0 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} fib(n) \\ fib(n-1) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} * \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2+1 \\ 2+0 \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \end{bmatrix}$$

$$\begin{bmatrix} fib(n) \\ fib(n-1) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} * \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 3+2 \\ 3+0 \end{bmatrix} = \begin{bmatrix} 5 \\ 3 \end{bmatrix}$$

? = $\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{(n-1)}$

actually ^(n-2) to account for initial values

```
    n = 40 # Which fibonacci number do you want
    mat_fib = np.linalg.matrix_power(np.array([[1,1],[1,0]]), n-2) # TODO
    fib_base = np.array([[1],[1]]); # First and second fibonacci numbers as a column vector
    fib_n = np.matmul(mat_fib, fib_base) # Solve for the nth and n-1th fibonacci numbers
```

1.3 Now that you have these two implementations, let's see which is going to be faster and ensure they still arrive at the same answer. Try using both implementations to solve for the 40th Fibonacci number. You don't have to time exactly how long this takes, but give us a rough estimate (within 5 seconds) of how long it took for your implementations to solve for this. Note, I don't recommend you try Fibonacci numbers higher than this for the recursive implementation. Then answer a few questions [5]:

Matrix - < 1 second
Recursion - 34 seconds

a) Which one was faster?
Matrix approach

b) How much faster?
34 seconds

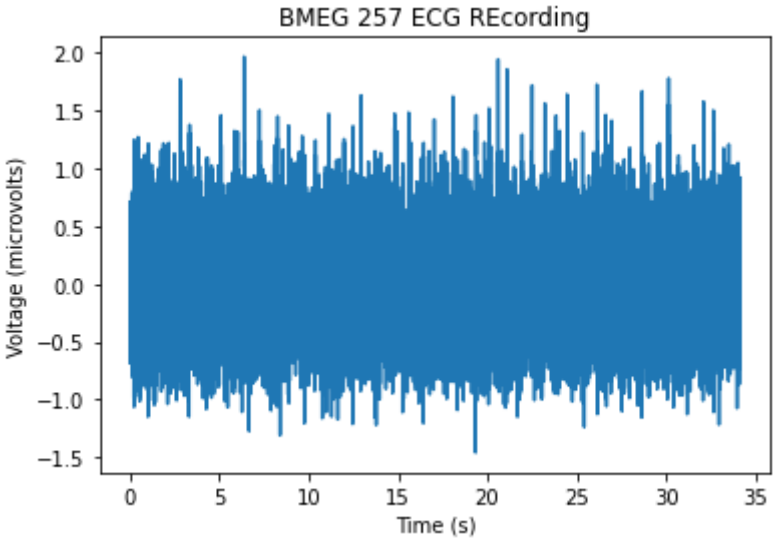
c) Why do you think this is the case?
Because instead of looping through the function and reusing a lot of values, there is only one step to calculate the fibonacci number.

d) Which one was easier to implement as code (e.g. which one was easier for you to write code for)?
The recursive approach in 1.1.

e) Which one was easier to conceptualize (e.g. which method was easier for you to come up with the solution)?
Recursive approach because I learned about it in CPEN 221.

Problem 2: Identifying Heart Beats (50 points)

2.1 First things first, let's have a quick look at the data. The template for loading the data and plotting the data are already provided in the main script "HeartBeatDetectionScript" (.m or .py for matlab or python). This is the main script you will run throughout this program to track your progress, however much of the code will be placed in separate functions. You'll notice that the plotting functions are commented out here (e.g. they will not be run). Uncomment the plotting lines and run the script to plot the ECG data. Copy the plot and show it here. What does the signal look like to you? Do you think you could figure out how many heartbeats are represented here? You can use this code for future reference on the syntax to plot things. [2]



2.2 Now that you've plotted the data, let's take a look at the data itself. You'll notice that the data is housed as a matrix in a variable called "ecg_data". What are the dimensions of this matrix? What does each dimension represent? (hint: refer to the matrix programming lecture). [3]

shape of matrix: (34141, 2)
This means that the matrix stored in ecg_data is a 34141 x 2 matrix or that it has 34141 rows and 2 columns.

2.3 The original file where this ECG data was stored is in binary form and has a size of 267kB (kiloBytes, in the Data folder as "ECG_Signal.bin"). Now we will have you save the data as a .csv file to demonstrate how the file size changes depending on how you represent the data. Before we do this, we will have you estimate how big the .csv file will be. To do this, assume each value in your "ecg_data" matrix will be stored using 6 characters in the final .csv file. Each character is represented using a single byte (1 byte = 8 bits). About how many bytes will your .csv file have (show your work)? Is this larger than the original binary file? (hint: refer to the data storage lecture) [5]

34141 rows x 2 columns x 6 characters x 1 byte/char = 409692 bytes

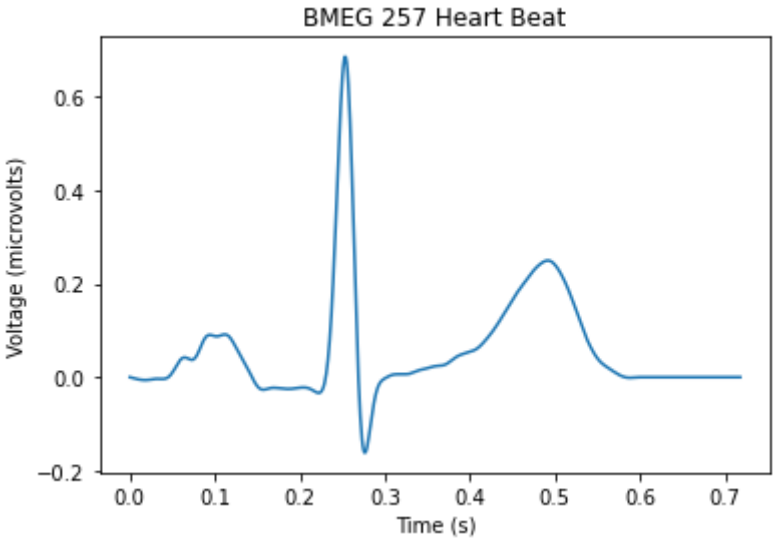
original binary file = 273000 bytes

The .csv file is about 2x as big as the original binary file.

2.4 Now, use Matlab or python functions to save your data as a .csv file. How big is this file? Is it close to your estimate? Note, your estimate may be off because the code you use chose to represent data with more or fewer characters, check the file to see how many characters are being used to represent each data roughly. [5]
hint: you can see documentation here on how to save data as csv files

The new file is 1.7MB or 1700000 bytes which is about 4x the estimate. This is because the average character length of the data points is about 18 characters.

2.5 Now let's start to work on the cross correlation. For this problem, we are providing you with the heartbeat shape that is recorded in the "ecg_data" signal. The data for the step is loaded into the script in as "heartbeat". You'll notice the that these data matrices have a similar organisation as the "ecg_data" signal. First, let's plot this to see what a step looks like. Paste the plot here. [5]



2.6 Now we will actually write an implementation of cross correlation. Cross correlation is a mathematical process that is useful for checking the similarity between two signals. In our case, we are trying to identify when the ECG data shows a pattern matching a heart beat. We can use cross correlation here to compare a what a heart beat looks like with a piece of the ECG signal, and when these match, we can claim that the heart likely beat at that point (see Figure 2). To compute the cross correlation, we need to take a section of the ECG recording that is the same length as the heart beat template. From there, we can perform the cross correlation, which is computed by taking each element in the heart beat template and multiplying it with the matched element in the ECG data segment, and then summing these products together. A perfect match would result in a large cross correlation, whereas a weak match would have a low cross correlation (Figure 2).

Your job is to complete the TODO block in the “ForLoopCrossCorr” file that will compute the cross correlation values using a for loop. You’ll notice that the code already has an outer for loop that is will help you cycle through segments of the full “ecg_data” signal. The reason we set this up for you already is because the cross correlation output vector is smaller than the neural signal and as a result you’ll notice this for loop does not go the full length of the “ecg_data” signal (Figure 2). We have set this up for you here but keep this in mind for Problem 2. (Hint: You may use the index variable “i” for your own code, which will point to the start of the portion of the “heartbeat” signal that will be cross correlated.) [12]

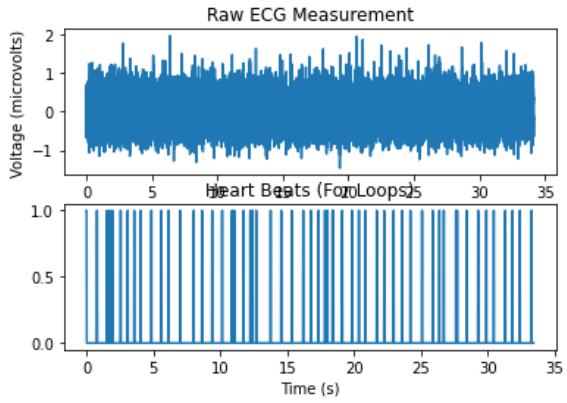
```
cross_correlation = 0;

    for j in range(heartbeat_samples):
        cross_correlation += ecg_data[i+j] * heartbeat[j]
```

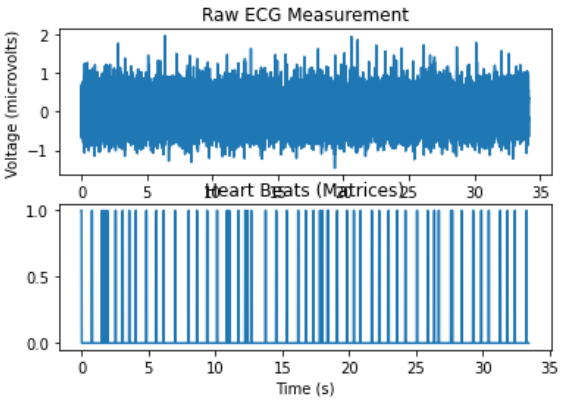
2.7 Now we will implement this cross correlation using matrix math. To perform this, you will complete the TODO block in the “MatrixCrossCorr” file without using any additional for loops (as with the previous problem, the outer for loop is already provided for you and you may use the index variable “i” for your purposes). Consider what does the cross-correlation process look like to you? How would you represent it using matrix arithmetic (hint: what does a 1xn matrix multiplied with a nx1 matrix look like?). [12]

```
cross_correlation = 0;
    cross_correlation = np.dot(ecg_data[i:i+heartbeat_samples], heartbeat)
```

2.8 Both the “ForLoopCrossCorr” and “MatrixCrossCorr” files generate a vector of cross correlation values (the green vector in Figure 2) representing comparisons of the heart beat template against segments of the ECG signal. But alone, those vectors do not really tell you where a heart beat is. Here, we will have you complete the TODO block in “ThresholdBeats” to identify when the cross correlation values exceed a threshold and can be considered a heart beat. There are several ways to implement this and this is where you can get a bit creative (simplest: you can say a heart beat occurs whenever the cross correlation is above a threshold, currently set at 7 because I know this works, but you are welcome to play around with this to see how it changes the results). What is important is that whenever you detect a heart beat, you set the appropriate index in the “beats” variable to one (by default, everything is set to 0 so no heart beats anywhere). When you complete this code and run the main “HeartBeatDetectionScript” file, it will automatically generate figures depicting these heart beats using both the for loop method (figure 10) and the matrix method (figure 11). Did both of your methods produce the same results according to the figures (they should!)? Based on the output figures, are the predicted heart beats the same? The ECG signal we used here is a simulated arrhythmia which is usually identifiable by the fact that the heart rate is not consistent (e.g. the timing between heart beats is variable). Do you think the predicted heart beat patterns confirm this? Please copy both output figures here (figure 10 and figure 11) for full credit. [6]

*With threshold of 6

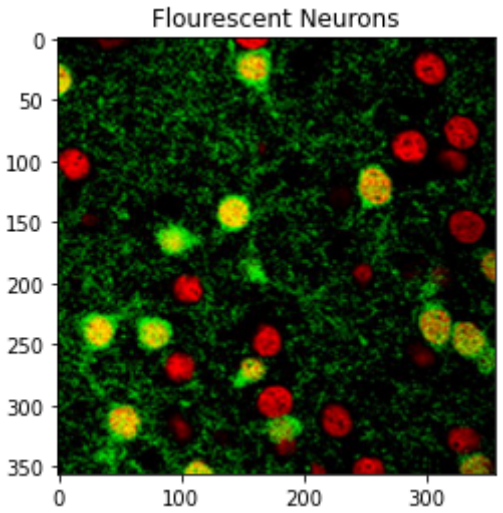




Both detect 1158 heartbeats with a threshold of 6, and they have identical heart beat graphs. The heart beats are clearly not evenly spaced, showing indication of an arrhythmia.

Problem 3: Finding Neurons in an Image (30 points)

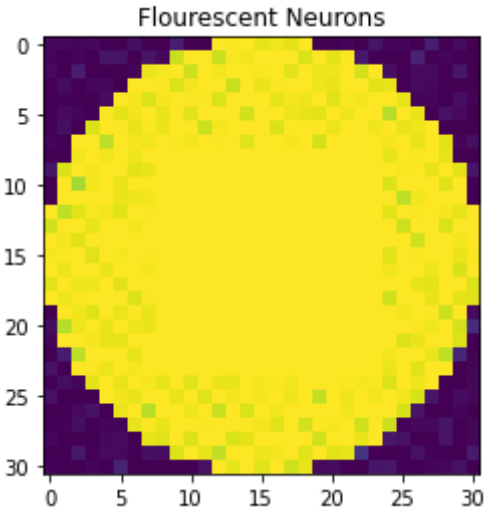
3.1 For this problem, the main script we will be using is called “NeuronIdentificationScript”. As with problem 2, you will primarily run this script to test your code, but you will implement code in some of the other functions as well. As with the ecg data in problem 1, we first start by loading in the fluorescing neuron image in the variable “image_neuron” and plotting it for you. Make sure you can see a picture of what appears to be red and green neurons on a mostly black image. Your first task here is to explore the data that is stored in “image_neuron”. What are the dimensions of the matrix? What do you think each dimension represents? (hint: refer back to the matrix programming lecture). This will be important as we want to separately identify the red and green neurons. How might you go about looking for neurons of just one colour? [10]



(356, 356, 3)
The shape of image_neuron is 356 x 356 x 3. This means that the images are 356 (height) x 356 (width) pixels x 3 colour channels (RGB).
To find neurons of one colour, such as red I would only analyse the colour channel that corresponds to that colour for the neurons since all other pixels would have a brightness of 0 in that colour. Code would look like: matrix[:,:,# for colour channel]

3.2 Before we start the cross correlation process, let's have a look at the template that you will be using to look for neurons in the image. The template is loaded in for you, so your job here is simply to plot the template (hint: use the same code that we used to plot the neuron image). What does this template look like to you? Post the template image here and let us know what it looks like to you (note, depending on what you are using, the image may be black and white or purple and yellow or some other combination of two colours). [5]

```
template_neuron = img.imread('Data/template.jpg');

fig1 = plt.figure(2);
fig1.clf()
ax = fig1.add_subplot(1,1,1)
plt.imshow(template_neuron);
ax.set_title( 'Flourescent Neurons' );
```



Here is the yellow and purple circular template that will be used for identifying neurons within the image.

3.3 Now let's get into the cross correlation process. Again, the cross correlation process is conceptually the same as with the 1-D ecg signal case, but now in 2-D. In the 2-D case, we are taking small sections of the larger image that match the size of the template and cross correlating the two. In this case, the cross correlation is computed by taking the product of each pixel in the template with the matching pixel in the image section, and then summing all of those products. As with the 1-D case, the larger the cross correlation value, the better the match. For this, we are allowing you to implement the cross correlation in whatever way you wish within the TODO block of the "ImageCorrelation" file. In problem 2, you will recall that we provided you with an outer loop that cycled through the larger neural signal and picked out sections to cross correlate. Here, we have left that process for you to figure out how to cycle through the image to pick out image sections to cross correlate (note, you will have to iterate left and right across the image as well as up and down over the image). I recommend referring to the code provided in problem 2 for inspiration. For this, also keep in mind that you will need to use this function to find both red and green neurons. Go back to the main script and fill in the inputs to your function that allow you to process just the red neurons and just the green neurons. [12]

ImageCorrelation.py

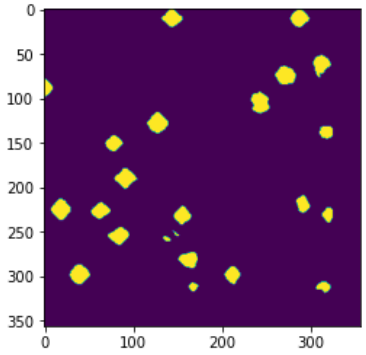
```
img_width = image.shape[1]
    img_height = image.shape[0]
    template_width = template.shape[1]
    template_height = template.shape[0]

    test_section = np.zeros( [template.shape[0], template.shape[1]] )
    for i in range( img_height - template_height):
        for j in range ( img_width - template_width):
            test_section = image[i:i+template_height,j:j+template_width]
            correlation = np.sum(test_section * template)
            corr_matrix[i,j]=correlation

    return corr_matrix
```

NeuronIdentificationScript.py

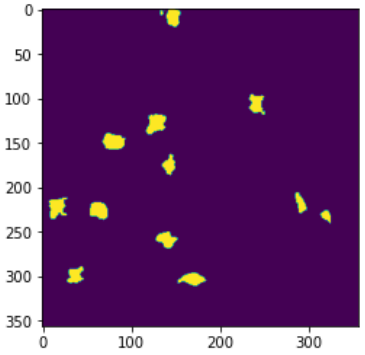
```
    red_correlation = ImageCorrelation( template_neuron, image_neuron[:,:,0] );
    green_correlation = ImageCorrelation( template_neuron, image_neuron[:,:,1] );
```

3.4 Now that you've completed the cross correlation, we have written the thresholding code here for you as it is a bit more complicated than in the 1-D neural data case. You are welcome to take a look at the thresholding function to see what's going on under the hood (it is commented to explain what is happening), but the idea is very much similar to the 1-D neural data case. What's important is that at the end of the script, a black and white image (or some other two colours) is produced that shows where the correlations were above threshold (white) and potentially represent neurons. Paste the plot here for both the red and green channels. Compare where the computer found red and green neurons with the original image. Do you see any similarities? Any differences? [3]

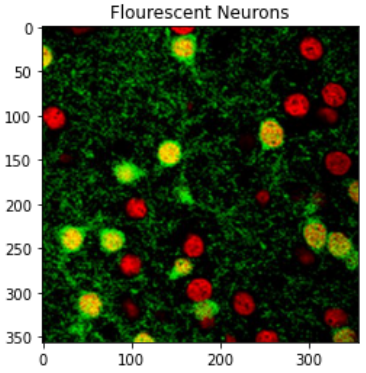
Red Plot



Green Plot



All Colours



For both plots it didn't pick up neurons in the area greater than ~325 pixel mark along the y-axis. In the red plot it identified some extra neurons because some of the yellow and green neurons have red spots on them. It picked up most of the neurons except for some of the darker red ones.