ENGR131 – Homework 4: Array Operations

Requirements:

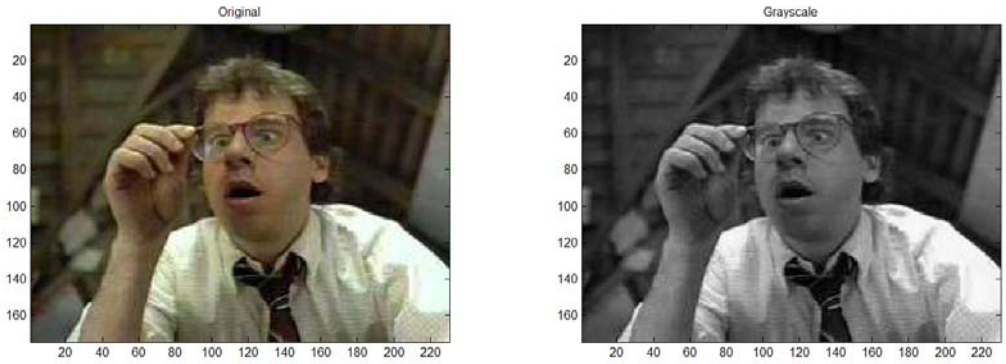
- Submit a single .zip file containing all .m files by the posted due date.
- Every .m file should include a comment at the beginning with your name and network ID.
- All work should be your own, as explained in the Academic Integrity policy from the syllabus.
- Problems will be graded for completeness (reasonable attempt for each rubric item). They will not be checked for correctness.

Using demo programs: Demo programs are provided as .p files. There are three steps for running a .p file: (1) copy it to a folder that's *different* from where you are writing your own program, (2) select that folder in Matlab as the current folder, and (3) type the script name in the Command Window. *WARNING:* Having a .p file in the same folder as your .m file may cause an error message.

Problem #1: Digital Images (loop approach)

Write a function named "grayscale1.m" that receives a filename for an image file and returns a 3-D array with a grayscale version of the image. It should work for images of any size. Your solution should use nested loops to set the value of every pixel.

Sample output: Below is an example of a color image and the grayscale image returned by the function. Use any image file you wish or the file shown here that is available on the website.



Tip (loading and displaying images): As explained in lecture, images can be loaded using the "imread" function and displayed using the "image" function. A sample .jpg file is provided and can be displayed as shown below.

```
>> pic = imread('RickMoranis.jpg');
>> image(pic)
```

Tip (image sizes): To work with images of any size, you will need to get the number of rows and columns using the "size" function like this: `[rows cols colors] = size(pic)`. As explained in lecture, the sizes of the first two dimensions are the number of rows and columns, respectively, and the third dimension is number of colors.

Tip (making grayscale): Grayscale images have the same value for the red, green, and blue brightness amplitudes for each pixel. Create the grayscale value by calculating the mean (average) of all three values in the color image. Consider an image with the following color values for the pixel at (100, 200):

```
>> pic = imread('RickMoranis.jpg');
>> pic(100, 200, :)
ans(:,:,1) =
   32
ans(:,:,2) =
   26
ans(:,:,3) =
   14
```

The mean can be found as follows:

```
>> mean(pic(100, 200, :))
ans =
    24
```

Tip (converting to grayscale): To convert the pixel at (100, 200) to grayscale, you could set all three values to 24 like this:

```
>> pic(100, 200, :) = 24;
```

Tip (convert image to 3D array): Note that your function should receive the filename and return a 3-D array that can be used with the "image" function. You should be able to display the new image like this:

```
>> image(grayscale1('RickMoranis.jpg'))
```

Rubric

Item	**Points**
Function header	10
Has loops	5
Computing the grayscale value	5
Total	20

Problem #2: Digital Images (vectorized approach)

Write a function named "grayscale2.m" that performs the same task in Problem #1 but uses matrix mathematics instead of loops.

Tip (calculating the mean): As explained in lecture, arrays of the same size can be added together using the "+" operator. Notice that the 3-D array contains three 2-D arrays. You can create a new 2-D array that contains the grayscale (mean) values by summing the three 2-D arrays and dividing by three. However, there is a complication because the image array has a data type of "uint8" which has a maximum value of 255 (to reduce memory size). If you try to add two or more values of type "uint8", you cannot get a result that is greater than 255. For example, try this in the Command Window using the "uint8" function to create a variable with that data type:

```
>> a = uint8(255)
a =
  255
>> b = uint8(255)
b =
  255
>> mySum = a + b
mySum =
  255
```

This is not helpful! In order to compute using the "+" operator, you will need to create a copy of the 3-D array with a data type of "double" as discussed earlier in the semester and in Ch 1 of the textbook. Below is an example of using the "double" function to add two "uint8" values. This converts both values to type "double" and the correct value is stored in a variable named "myDoubleSum".

```
>> myDoubleSum = double(a) + double(b)
myDoubleSum =
   510
```

Tip (creating the grayscale 3-D array): The previous tip explained how to create a single 2-D array of grayscale values. Once this is done, you can simply copy that 2-D array into the three different 2-D arrays in the original 3-D array using the assignment operator "=". Fortunately, this will automatically convert the "double" values into "uint8" values for use with the "image" function. This is important because the "image" function won't work with arrays of type "double".

ADVANCED (for fun, not for credit): You can learn more about the "uint8" data type by typing "doc uint8" in the Command Window and exploring the tutorials at the bottom of the help page. Also, the "mean" function can be used to avoid handling the mathematics directly. Type "doc mean" and see if you can figure out how to use the "mean" function instead of adding the 2-D arrays.

Rubric

Item	**Points**
Function header	10
Computing the grayscale value	5
Assigning new values to array	5
Total	20

Problem #3: Heart Rate Analysis (loop approach)

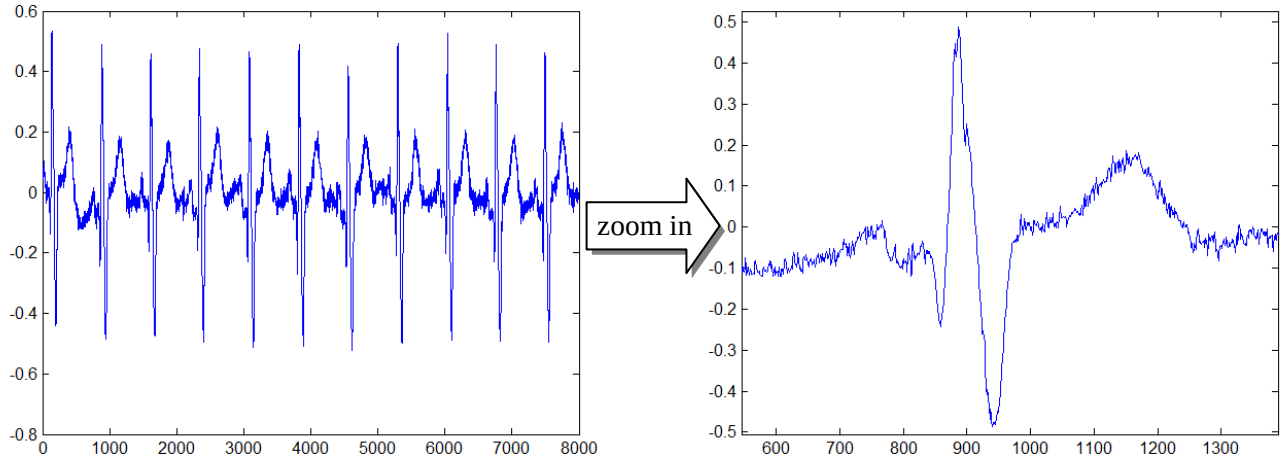
Write a function named "getRate1.m" that receives the name of a file with electrocardiogram (ECG) data and returns the mean (average) heart rate. Two sample ECG files are provided on the website. Calculate the heart rate by counting the number of heartbeats using a loop.

Sample output: Below are examples of using the function with the two data files. The heart rates are 81.6327 and 66.1765 beats per minute, respectively.

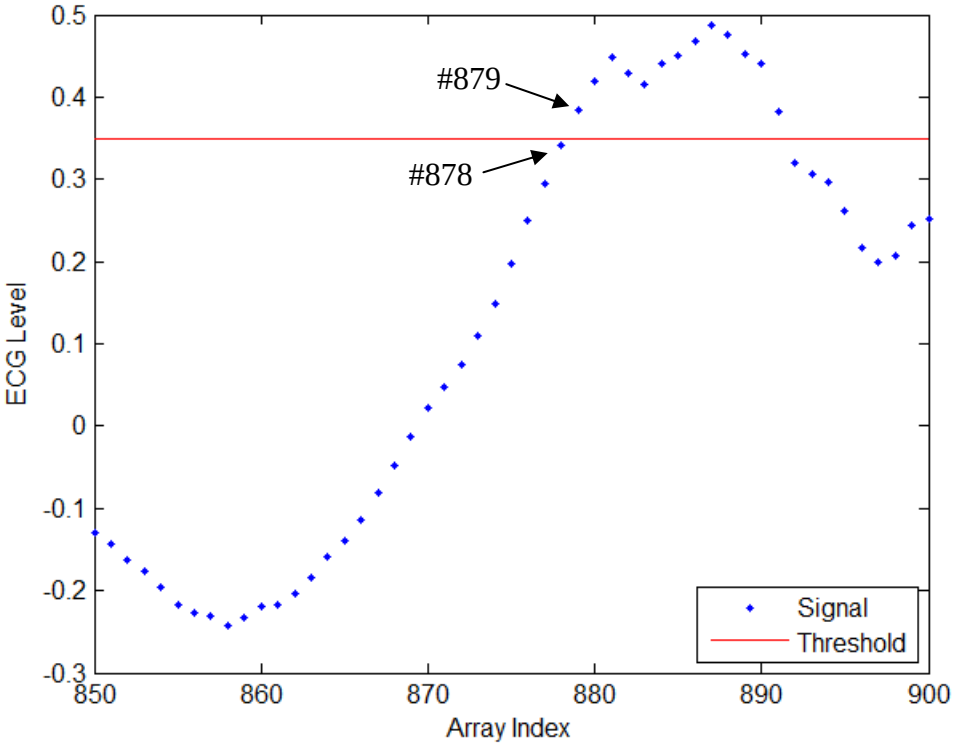
```
>> getRate1('ecg1.txt')
ans =
   81.6327
>> getRate1('ecg2.txt')
ans =
   66.1765
```

Tip (understanding the data files): An ECG is a measurement of cardiac electrical activity over time. (Curious students can visit http://en.wikipedia.org/wiki/Electrocardiography.) The provided files contain ECG recordings with one measurement every 1 ms. The file "ecg1.txt" contains 7,999 measurements (called "samples") over an 8 sec period, and the file "ecg2.txt" contains 8,999 samples over 9 sec. Though not required, you can visualize the data as shown below. You can use the menu or toolbar to zoom in on a single heartbeat as shown below.

```
>> ecg = load('ecg1.txt');
>> plot(ecg)
```



Tip (counting heart beats): One way to detect a heartbeat is to look for a place where the signal crosses a threshold only once for each heartbeat. This can be done inside a loop using an "if" statement to check whether one sample is below or equal to the threshold value and the next sample is above the threshold. Below is a plot that is zoomed in even further than the previous plot and shows the individual array values. The red line represents a chosen threshold of 0.35, and the arrows point to samples #878 and #879.



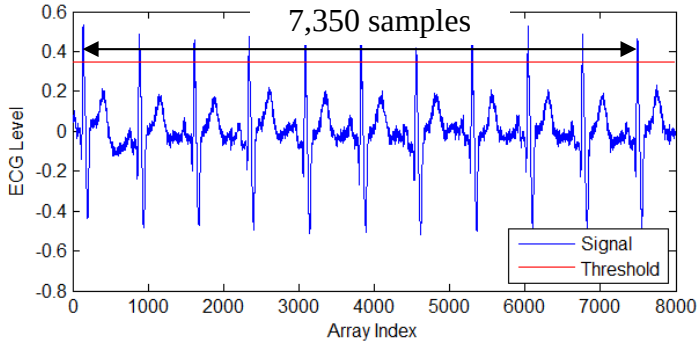
For example, if a variable "i" is used for the index of an array named "data", we could check for a threshold crossing using the following "if" statement:

```
if data(i) <= 0.35 && data(i + 1) > 0.35
```

Notice that this uses "data(i + 1)", so the maximum value of "i" should be one less than the length of the array.

Tip (mean heart rate): The mean heart rate is the number of complete heart beats within a period of time, such as beats per minute (bpm). A complete heartbeat lasts from one threshold crossing to the next threshold crossing. One way to estimate the rate is to divide the number of complete heartbeats by the total time from the first heartbeat to the last heartbeat. We can calculate the total time by using the indexes of the samples for the

first and last threshold crossings. The total time between them is 0.001 seconds multiplied by the number of samples between the first and last. For example, the plot below has 10 complete heartbeats in 7,350 samples which is 81.6 bpm.

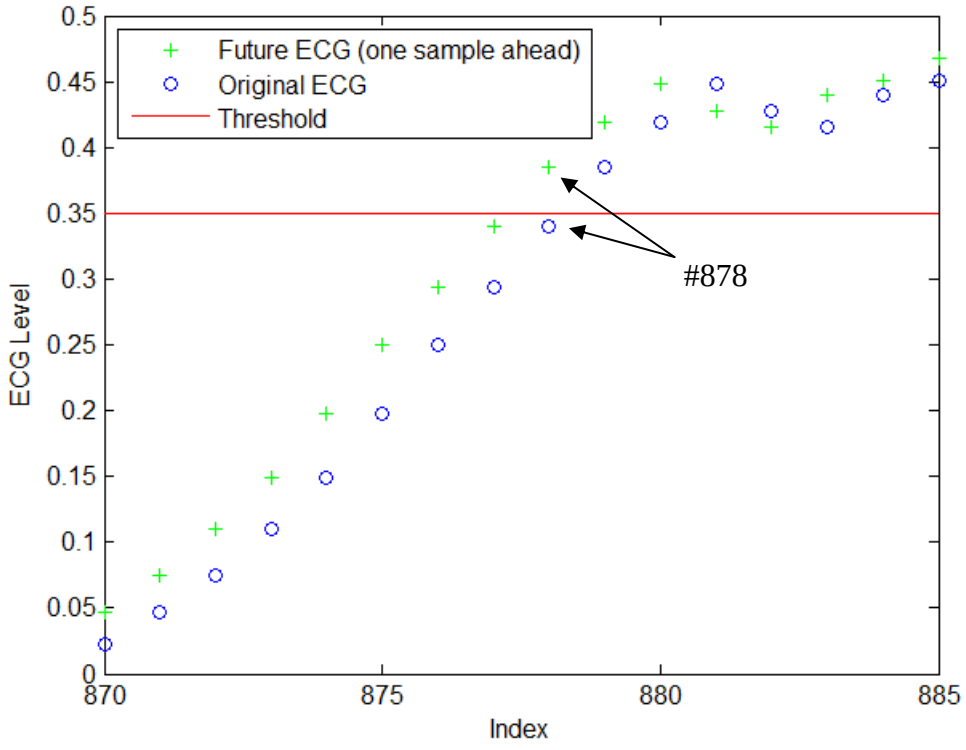


Rubric

Item	**Points**
Function header	10
Has loop	5
Checking for threshold crossing	5
Total	20

Problem #4: Heart Rate Analysis (vectorized approach)

Write a function named "getRate2.m" that performs the same task in Problem #3 without using a loop. We can try to use the "find" function to detect the heartbeats. Unfortunately, we cannot specify an index value of "i + 1" when using the "find" function. We can overcome this limitation by using another array that looks into the future by exactly one sample. In other words, the array of future values is a copy of the original array that is shifted to the left by one sample. The plot below shows a zoomed-in view of both arrays. Notice that sample #878 in the original array (blue circles) is below the threshold and sample #878 in the array of future values (green crosses) is above the threshold.



This allows us to use a logical vector with the “find” function as shown below with arrays named “original” and “future”:

```
find(original <= 0.35 & future > 0.35)
```

Notice that we have used the “&” operator instead of the “&&” operator. The “&” operator performs an element-wise operation on the two logical vectors that are created, as described in p 63 of the textbook.

Tip (creating an array of future values): Remember from Problem #3 that using “data(i + 1)” required “i” to be one less than the length of the array “data”. In this vectorized version, having an array of future values means the array will lose the first value that was in the original array. More importantly, it will also be shorter. However, the arrays named “original” and “future” in the above example must have the same length. You will need to decide how you wish to make the arrays have the same length.

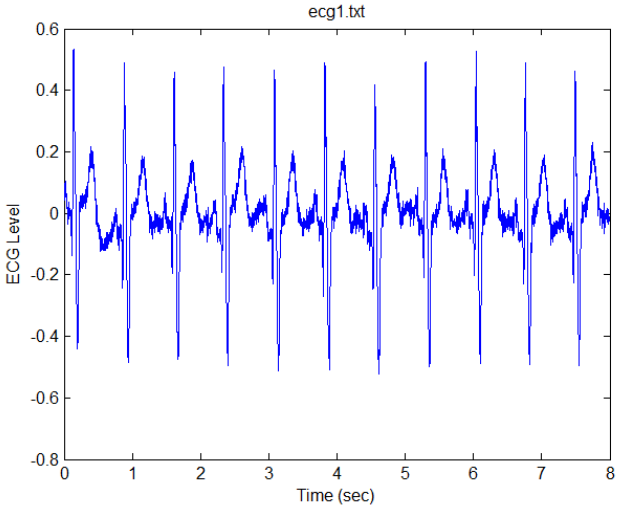
Rubric

Item	**Points**
Function header	10
Uses “find” function	5
Computes rate	5
Total	20

Problem #5: Heart Rate Integrated Application

Software that integrates several related features is often called an “application”. (Curious students can read more at http://en.wikipedia.org/wiki/Application_software.) Create a program that allows the user to repeatedly choose to either plot an ECG file or print the mean heart rate for that file. There should also be an option to quit. Plots should use units of time on the x-axis and be completely labeled with the filename in the title. The heart rate should be computed using either one of the functions above and then displayed in a formatted message. *Sample output:* Below is an example of using the program (named “ECG.m”) and the results that should be displayed.

```
>> ECG
0. Quit
1. Plot ECG
2. Get heart rate
Enter your choice: 1
Enter a filename: ecg1.txt
```



```
0. Quit
1. Plot ECG
2. Get heart rate
Enter your choice: 2
Enter a filename: ecg1.txt
The heart rate is 81.6 beats per minute.

0. Quit
1. Plot ECG
2. Get heart rate
Enter your choice: 0
```

Rubric

Item	**Points**
Has user-input loop	10
Checks user input using “if” statement(s)	5
Calls “getRate” function	5
Total	20