MATLAB Assignment 3

Spring 2020, Section A

This problem set's goal is to familiarize you with two useful concepts in MATLAB, functions and logical indexing, as well as begin getting you used to plotting.

Please submit this homework as *.m* files (note the plural), with suppressed output (obviously, the plots will still be displayed). Remember that all lectures and homeworks may be found at *github.com/guybaryosef/ECE210-materials*. This homework is due by 4:00 PM on February 12th to guybymatlab@gmail.com. Remember to also bring a hardcopy in to class!

1. Lunar Eclipse This question guides you through some basic image processing techniques in MATLAB. You will create interesting images with relational and logical indexing, as well as use the ***imshow*** function to visualize what you have created.

a. Create a 100×100 matrix A whose contents are all ones.

b. Create a 100×100 matrix B whose contents are all zeros.

c. In matrix A, set the values of entry $a_{i,j}$ equal to 0 if $\sqrt{(i-50)^2 + (j-50)^2} < 20$. **Hint** : ***meshgrid*** would be useful in creating the indices.

d. In matrix B, set the values of entry $a_{i,j}$ equal to 1 if $\sqrt{(i-40)^2 + (j-40)^2} < 20$.

e. Visualize the following results with ***figure*** and ***imshow***. Describe each of the results with one sentence each, either as a comment or by printing to stdout.

- A
- B
- The intersection between A and B
- The union between A and B
- Complement of the intersection between A and B
- Complement of the union between A and B

2. Sincing Ship Here we will look at a function near and dear to the signal processing community's heart - the sinc function. You are going to implement your own functions to find the local extrema and roots. As we haven't yet gone over plotting, don't stress over stylizing your plots beyond what is asked.

a. Sample a sinc with 10001 linearly spaced points on $[-2\pi, 2\pi]$ using the ***sinc*** function. Plot your results, and give the plot a title.

b. Create a function (either anonymously or in another file) which locates the indices at which the input vector transitions from one sign to another. **Note:** This can be done in one line of code but it is $trecherous^{FF}$. For one scenario the vector has a positive value followed by a negative value, i.e. $v(n) > 0$ and $v(n+1) < 0$. In this scenario the root occurs somewhere in between and you can pick either n or $n+1$ as the root. We could loop through the vector and check this condition at every point - but don't do that. Instead think of a way to use logical indexing: You will want to write conditions on the vector and some kind of shifted version of itself. Beware however, when you do this you will have non-overlapping points. It is up to you to figure out what to do with them.

c. Apply your function from part (b) to the sinc you created in part (a). Find the roots (x and y coordinates) and plot them as black circles on top of the sinc using ***plot(xRoots,yRoots,'ko')***.

d. Now we are interested in finding the extrema (local minimums and maximums). Firstly, approximate the derivative by taking the difference between all adjacent elements and dividing by their time spacing. Then apply your roots function to the approximate derivative of your sinc to obtain the extrama. Finally, plot them as red stars on top of the sinc using ***plot(xMinMax,yMinMax,'r*')***.

3. Fun with find Write a function to return the value and index of a number in a vector / matrix that is closest to a desired value. The function should be called as $[val, ind] = findClosest(x, desiredValue)$. This function can be accomplished in less than five lines. Show that the function works by finding the value (and index) closest to 3/2 in ***sin(linspace(0,5,100)) + 1***. You may find the functions ***abs***, ***min*** and/or ***find*** useful. **Hint**: You may have some trouble using ***min*** when x is a matrix- remember that you could convert any matrix to a column vector using $y = X(:)$.

Bonus (worth 1 extra point): Implement MATLAB's *sub2ind* function for the 2-D case. Because you are only doing this for the 2-D case, the function prototype will look like: $linearInd = sub2ind(matrixSize, rowSub, colSub)$. You do not need to worry about infinite error checking, but try to make it at least a plausible implementation.