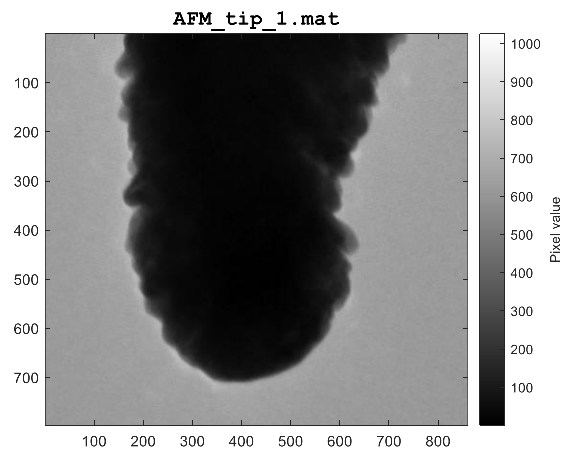
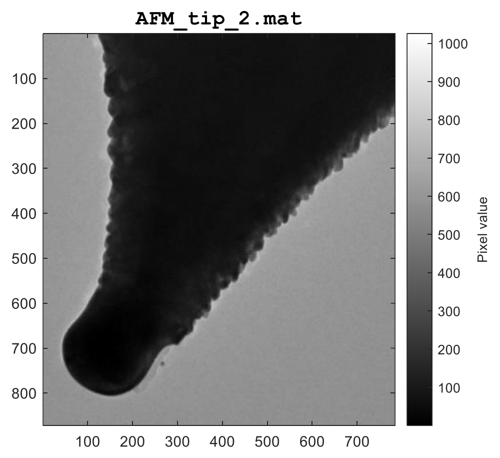
ENGR 105 – Introduction to Scientific Computing

Assignment 7

Due by 11:59 pm on Weds. 3/2/2022 on Gradescope

Problem 1 (40 pts): Below are transmission electron microscopy (TEM)[1] images of nanoscale atomic force microscopy (AFM)[2] probe tips – needle-like devices used to image and poke at surfaces with nanoscale resolution. The dark regions are the probe tips and the light regions are background noise from the image capture process. The color value of each pixel in the image comes from a double precision value stored in an m x n matrix.





Create a function that identifies the edges of the AFM tip. A reasonably effective edge detection routine for images like those above involves checking the nearest neighbors - i.e., all row/column indices that border the index of interest including the corner indices - for each matrix index. If the index of interest has a value greater than or equal to some pixel threshold value AND at least one nearest neighbor has a value less than that threshold, then the pixel of interest is an edge pixel.

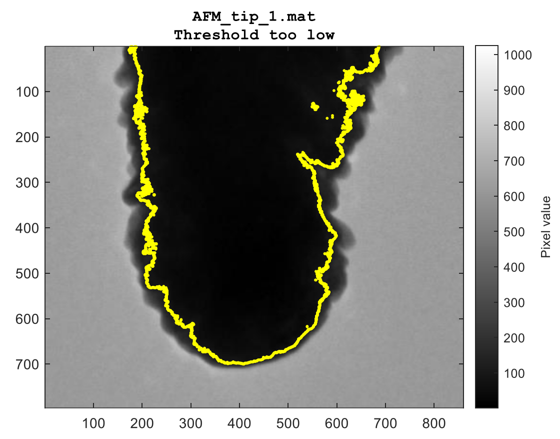
Your function should take as input an m x n double precision matrix that represents the image information and a scalar variable representing the threshold. It should output the row and column indices of the edge pixels as two separate vectors. Your function may not leverage built-in MATLAB functions related to edge detection.

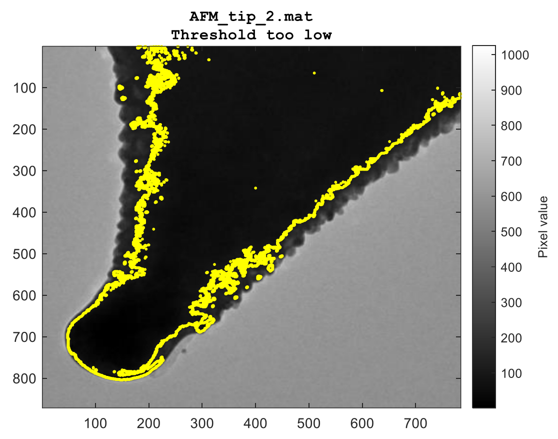
In addition, create a script that loads the example AFM probe tip images you have been provided – `AFM_tip_1.mat` and `AFM_tip_2.mat`, calls upon your edge detection function to determine the edges of each tip, and plots original edge detection image with edge pixels overlaid. The script you upload should include the thresholds you found that yield the most accurate edge detection.

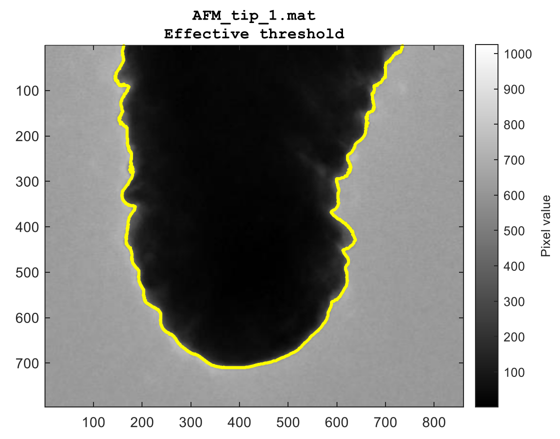
[1] See https://en.wikipedia.org/wiki/Transmission_electron_microscopy for more information on the TEM electron imaging technique.

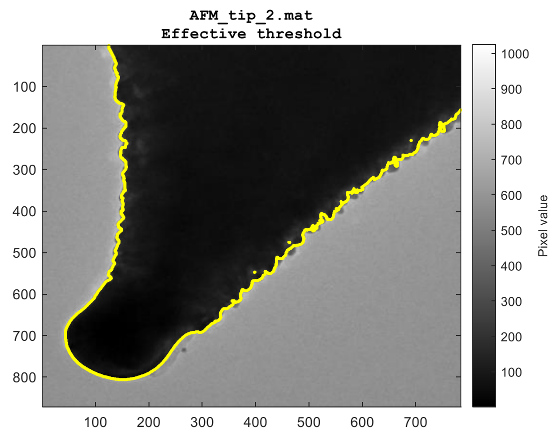
[2] See https://en.wikipedia.org/wiki/Atomic_force_microscopy for more information on this AFM probing technique.

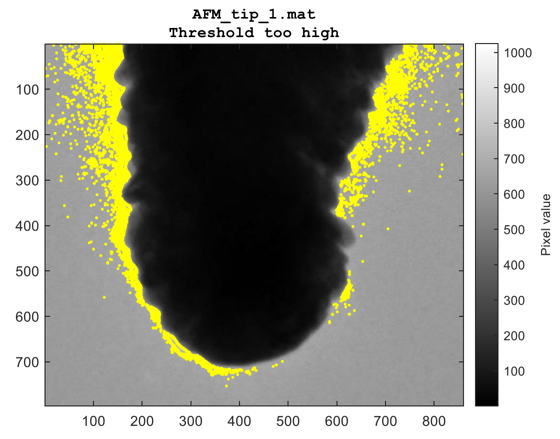
As an example, the following images were produced with a threshold that was too low, appropriate for effective edge detection, and too high.

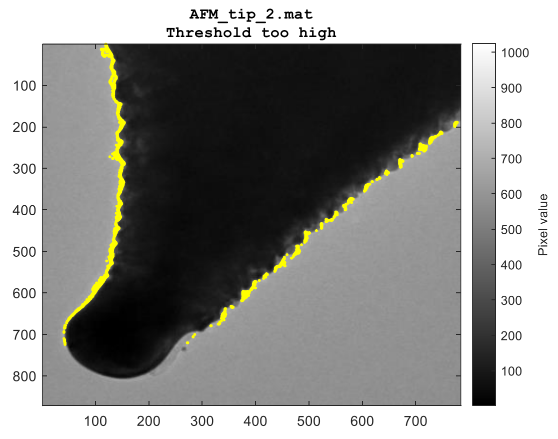












The following code was used to overlay the row and column indices (contained in vectors `row` and `col`) on top of a grayscale plot of AFM probe tip image data `AFM_tip_1.mat`.

```
figure                  % create a new figure window

image(AFM_tip_1)        % plot AFM probe tip image
colormap(gray(1024)) % specify gray colormap with 1024 discriminations
cb = colorbar;          % add a color bar with label
ylabel(cb,'Pixel value')

hold on
plot(col,row,'y.')      % overlay edge indices in yellow
hold off
```

Upload your script, function(s), and .jpg images of your best edge detection results for `AFM_tip_1.mat` and `AFM_tip_2.mat`. Identify the names of all files associated with this problem in your README.txt.

Problem 2 (20 pts): You are in charge of wind speed measurements at your local weather station. Part of your job is understanding how quickly the wind speed changes as a function of time. The following wind speed measurements (units: m/s) were taken as a function of time (units: s):

```
speed = [0.2, 0.3, 0.7, 1.3, 1.9, 255.0, 1.6, 8.5, 1.4, 255.0, ...
     0.9, 0.7, 255.0, 20.7, 0.6];
time = [0, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, ...
     1100, 1200, 1300, 1400];
```

Use logical vectors and logical indexing to answer the following via the development of a script.

A) How many measurements were less than 0.5 m/s?

B) What times correspond to measurements that were less than 0.5 m/s?

C) Which wind speed measurements were recorded between 350 and 1050 seconds?

D) Every wind speed measurement of 255 is a sensor error. Replace all the erroneous measurements with `NaN` and produce a pleasing plot of the result that employs data markers and lines joining the data markers. Note that the `NaN` values will be excluded from the plot (i.e. there will be breaks in the plot series line).

Upload the script and a .jpg copy of the resulting plot. Identify the names of all files associated with this problem and responses to questions a, b, and c in your README.txt.

Problem 3 (40 pts): You are working with the English Department to analyze the frequency of words used in classic texts and in service of understanding the evolving nature of language. You have been charged with developing a function with the following function declaration.

```
function [wordList,wordCount] = count_words(filename)
```

This function should take as input the filename of a plaintext (.txt) file that contains historic literature and is located in the same working folder as the function. The function should output a 1xN cell array `wordList` that contains all N unique words in the text and a corresponding 1xN vector `wordCount` that contains the corresponding number of incidences of each word with both outputs sorted in descending order of frequency. The algorithm should not distinguish capitalization. For example, the text "Of" and "of" would constitute two appearances of the word "of".

The function might be invoked as follows.

```
[uniqueWords,count] = count_words('CanterburyTales.txt')
```

Your central algorithm must be homebrew; it should not leverage high-level built-in functions such `unique` or `regexp`. However, the high-level functions `readlines`, `fgetl`, and/or `sort` may be used to load the text and sort the outputs. Low-level functions such as logical functions/tests (e.g. `isempty`, `isspace`, and `strcmp`) and character modification functions (e.g. `char` and `lower`) may be employed.

You are encouraged to use `readlines` or the `fgetl` method to import the text file. The following shows an example using `readlines`, though you may choose a different variable name in place of `rawText`.

```
rawText = readlines(filename);
```

The following represents one general approach to the algorithm.

1) Convert the incoming string array to one long character string of text being careful that all words are separated by spaces.

2) Remove punctuation and convert all characters to lowercase.

3) Generate a cell array (1 x M) of all words that appear in the long string of text. The beginning and end of each word could be distinguished by the spaces that precede and follow the word.

4) Generate a cell array (1 x N) that contains each unique word in the text and a vector (1 x N) that contains the number of instances of the words. This will likely be the most demanding part of the algorithm.

5) Sort the results (list of unique words and instances of each word) generated by (4) similarly and in descending order.

You have been provided with the plaintext file `CanterburyTales.txt` in order to develop and test your code. To help you check your results based on the example text file, the outputs should have 176 indices with the content of the first 10 indices shown below.

```
uniqueWords =
{'and'}{'to'}{'the'}{'that'}{'in'}{'of'}{'i'}{'with'}{'they'}{'whan'}
```

```
count = [17, 13, 12, 11, 11, 8, 7, 5, 5, 4]
```

Upload the function. Identify the name of the function in your README.txt.