**Introduction to Scientific Computing, Homework #3**

**Due by 12am, Tuesday, May 21, 2024**

**Problem 1 [10]:** The equation of an ellipse in polar coordinates is given by

where a is the semi-major axis and e is the eccentricity, if one focus is at the origin and

the semi-major axis lies on the x-axis.

Halley’s Comet, which visited us in 1985/1986, moves in an elliptical orbit about the Sun

(at one focus) with a semi-major axis of 17.9 A.U. (astronomical unit, the mean distance

of the Earth from the Sun, which is 149.6 million km). The eccentricity of the orbit is 0.967276. Write a program that draws the orbit of Halley’s Comet and the Earth (assume

the Earth is circular).

**Problem 2 [10]:** There are two dates, find the number of days between them, if the two dates are consecutive, set the number of days between them as two day. The input format is two integers representing two dates in the form YYYYMMDD. Note that leap years and peace years need to be taken into account. Leap years happen in two ways. The first case is when the year is a multiple of 4 and not a multiple of 100. For example, 2004 and 2020 are leap years. The second case is when the year is a full hundred and must be a multiple of 400. For example, 1900 is not a leap year, and 2000 is a leap year.

Your job is to write a function called processScores with the following signature

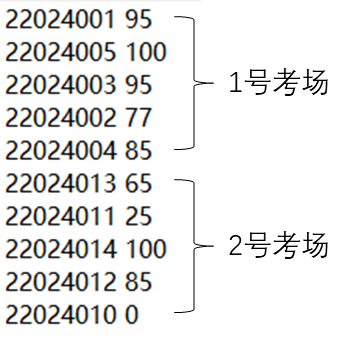
function difference=The\_date\_difference(time1,time2)

Some examples are as follows

If time1 is ‘20210101’, time2 is ‘20210105’, the difference is ‘5’.

If time1 is ‘20210105’, time2 is ‘20210101’, the difference is ‘5’.

**Problem 3 [20]:** You need to write a function to rank the grades of Zhejiang University students. The initial information includes each student's student number and grade. There are n examination rooms, each with 5 students. The initial information is placed in a text file and arranged in the order of the examination room.



You are to write a function called Scores\_ranking with the following signature

function rank=Scores\_ranking(n)

Your output should be an array of 5 columns, representing student number, score, examination room number, local rank, and final rank. Where, local rank represents the ranking within the scope of the examination room, and final rank represents the ranking among all students. The output must be sorted in nondecreasing order of the final ranks. The students with the same score must have the same rank. If the scores are the same, the output must be sorted in nondecreasing order of their student numbers.

22024005 100 1 1 1

22024014 100 2 1 1

22024001 95 1 2 3

22024003 95 1 2 3

22024004 85 1 4 5

22024012 85 2 2 5

22024002 77 1 5 7

22024013 65 2 3 8

22024011 25 2 4 9

22024010 0 2 5 10

**Problem 4 [20]: Lost in Translation**

Here is a procedure for scrambling words in a piece of text in such a way that the text remains readable. The trick is to keep the first and last letters of each word the same and simply scramble the middle letters. Your job is to write a function called scramble() which takes as input the name of the text file to be scrambled and writes the scrambled text to another file whose filename is the second argument. For example scramble(‘in.txt’,‘out.txt’) would read the contents of the text file in.txt and write the scrambled output to the file out.txt.

Approach:

You should begin by writing a function that scrambles a single word – ie write a function with the following signature that takes a string as input and outputs another string of the same length where the first and last characters are preserved but everything else is scrambled.

function out = scrambleWord(in)

So scrambleWord (‘forest’) may return ‘fresot’. You may find Matlab’s randperm function useful for this purpose. If the input string has punctuation characters or numbers you should leave those in the same place in the string. You may find the isletter function useful for finding the letter characters in the string.

Once you have written and tested the scrambleWord function you can use it to implement your final solution. Here’s one approach.

* Open the input file for reading and the output file for writing using the fopen command egs fid = fopen (‘foo.txt’, ‘w’) opens the file foo.txt for writing creating a new file if necessary while fid = fopen (‘foo.txt’, ‘r’) opens the file for reading.
* Repeat:
  + Read a line from the input file using the fgets function. Egs line = fgets(fid) – note that when you reach the end of the file this function will return a -1 – that’s how you know when you are done.
  + Break the line down into words using the strread function egs a = strread(line, ‘%s’) returns a cell array of words.
  + For each word that you found scramble it using the scrambleWord function and write it out to the output file. You can use the fprintf function to write to the output file egs fprintf (fid, ‘%s ’, str) will write the string in str to the file followed by a space and fprintf (fid, ‘\n’) will write a newline character to the file signaling the end of a line. (end of line is an ASCII character just like space or tab)
* Close the input and output files using the fclose function.

We will provide a few text files as examples that you can use to test your program. You can also create your own text files with an editor like notepad or emacs that allows you to save raw ASCII text files. Note that .doc files of .rtf are not native text files and won’t work for this purpose.

For fun try sending scrambled emails to friends and relatives and see if they notice any difference.

**Problem 5 (20 pts): The Game of Life**

The game of life, developed by British mathematician John Conway, is the classic example of a cellular automata. You can find a wonderful description of this game at the following Wikipedia site <http://en.wikipedia.org/wiki/Conway%27s_Game_of_Life>. Your job will be to implement this game in Matlab. The game state will be represented by a 2 dimensional logical arrays where 1s correspond to ‘live’ cells and 0s to ‘dead’ cells. In order to do this you will need to implement the following MATLAB functions.

* Function out = getCell (in, row, col)
  + Takes as input a 2 dimensional logical array and a row and col index and returns the value of the cell at those coordinates. If the coordinates are illegal, say the row or col indices are less than 1 or greater than the dimension of the array, this function should return 0.
* Function out = countNeighbors (in)
  + Takes as input a 2 dimensional logical array, in, and produces an output array the same size as the input which indicates how many live neighbors each cell in the input has. (Hint: use the getCell function)
* Function out = updateCells (in)

Takes as input a 2 dimensional logical array, find its living neighbors by using the countNeighbors(), and computes what the next generation would look like based on the rules of the game:

**For a space that is 'populated':**

Each cell with one or no neighbors dies, as if by loneliness.

Each cell with four or more neighbors dies, as if by overpopulation.

Each cell with two or three neighbors survives.

**For a space that is 'empty' or 'unpopulated':**

Each cell with three neighbors becomes populated.

You should test your life program using some of patterns listed in the wikipedia entry. You may find it convenient to store the patterns in small array (egs glider = [1 1 1; 1 0 0; 0 1 0] and then copy these subarrays into various locations in the input array). A 100 by 100 array should prove suitable for testing, a sample script LifeScript.m is provided for your testing.

**Problem 6 [20 pts]:** You have obtained a summer research position with Apple. The company is currently investigating reliability issues of the electrical connectors on the lightning cables. In particular, a new, lower cost manufacturing process has led to high resistances between the lightning plugs and the electrical connections with which they mate.

You are scanning these metal surfaces using atomic force microscopy (AFM) (<http://en.wikipedia.org/wiki/Atomic_force_microscopy>) with the hope of identifying variations in the surface of the electrical connecter that may contribute to these reliability issues. The measurement consists of rastering the connector surface with a nanoscale probe tip while measuring the topography of the surface, the friction force between the probe tip and surface, and current through the probe tip-surface junction. The resulting data sets are output as two dimensional matrices of data representing the magnitude of a response at regularly-spaced locations across the sample surface.

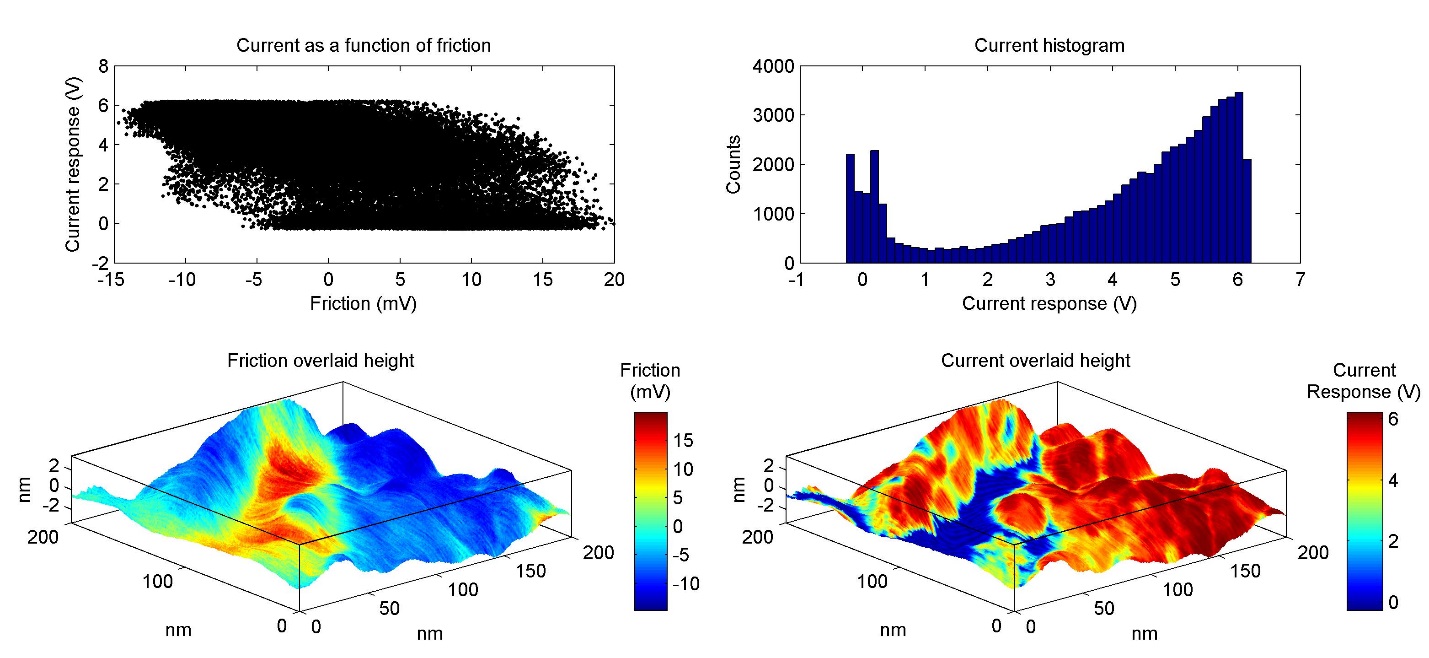
You have been performing a lot of these AFM scans and would like to visually summarize your findings for each data file. You decide to use MATLAB to quickly produce standardized plots of each data set and save the plots to your working folder. To do this, you concoct a plotting function with the following function declaration.

function plotAFMdata(H,F,I,ss,saveName)

where H is a two dimensional matrix of height data, F is a two dimensional matrix of friction data, I is a two dimensional matrix of current response data, ss is a scalar describing the scan size of the data set (taken to be 200 nm for your data), and saveName is the data file to which you will save the figure you produced. You can assume that H, F, and I will always be square and have the same dimensions.

The data is provided to you from the AFM (after some additional processing) as a .mat file with fields corresponding to ‘height’, ‘friction’, and ‘current’ (current response) in units of nanometers, millivolts, and volts, respectively. You have been provided with the data set AFMdata0001.mat to test your function.

plotAFMdata should produce a figure closely resembling the following figure.



plotAFMdata should produce a single figure with the following axes and specifications.

Axes 1 and 2) Produce a three dimensional plot of height with color provided by the friction data and a three dimensional plot of height with color provided by the current response data. The plots should include the following modifications.

* Each axis should extend no further than the range of the data.
* The azimuth and elevation should be specified.
* Do not include a grid.
* Include a bounding box.
* Include a color bar. You may have to adjust the position of the color bar so that it does not overlap the plot.
* Include a title on the color bar.

Axes 3) Plot current response as a function of friction.

* You can let MATLAB automatically select the domain and range.

Axes 4) Produce a histogram of the current response for all current data points.

* Specify a domain of -1 to 7 but do not specify the range.

All plot modifications should be invoked using code in plotAFMdata. All plots should include the following elements, specifications, or modifications.

* Include labels on all axes.
* Include titles on all plots.
* Specify the font size and font name for all text elements.
* Specify a white figure background color.

You want to ensure that your code will display the plot correctly on any modern computer. Therefore, your figure should be positioned in the center of the computer screen and have dimensions of 1000 x 460 pixels (width x height).

plotAFMdata will save the figure to the working folder under the name specified by saveName as a 24-bit jpeg with a resolution of 300 pixels/inch. Note that the command used to save MATLAB figures often resizes the figure during the file writing process. You may have to specify the ‘PaperPositionMode’ as ‘auto’ when first establishing the figure. plotAFMdata will then close the figure window after saving the jpeg.

You will create an associated script to test your function that loads the test data and launches the plotAFMdata function.

Include a copy of your plot in your .docx submission.