CS1133, Spr 2018, HW 4 (Lect 8, Pts=40+30)

Due February 20, 2018 (before 11:00 pm)

Problem 1 [40 pts]

Here we repeat problem 2 of the last assignment where the digits at one of the odd positions may not have been read correctly. Whatever that odd position happened to be, the digits at that same position in all the UPC codes were affected.

In this problem, we still assume that each of the UPC code has a missing or undetermined digit at a single odd position. However, we no longer assume that the locations of the missing digit be the same for all the UPC codes.

In computing, `NaN`, standing for not a number, is a numeric data type value representing an undefined or unrepresentable value, especially in floating-point calculations. Systematic use of `NaN`s was introduced by the IEEE (Institute of Electrical and Electronics Engineers, pronounced I-triple-E) 754 floating-point standard in 1985. Well-defined rules are in place representing and dealing with `NaN`s in numerical computations. So we will use a `NaN` every time there is a missing value in the UPC code. In Matlab, `NaN` is one of the few quantities whose name contains uppercase letters (to conform to the IEEE standard). It can also be written all in lowercase letters.

Here is an example of a matrix of UPC codes:

```
  7    2  NaN    8    3    5    7    9    0    1    4    6
  2    8    3    5    7    9    0    1    4    6  NaN    8
  0    1    4    6  NaN    8    5    9    3    3    3    2
NaN    8    5    9    3    3    3    2    9    0    1    2
  9    2    8    3    5    7    0    1    4    6  NaN    1
```

You can try working with this example, but as usual, your program must be able to work properly with other similar matrices.

Matlab has a function `isnan` that takes an input logical array and returns another logical array the same size as the input array. The resulting array has a value of `true` if the corresponding element of the input array is a `NaN`, otherwise its value is `false`.

Use the same method as in problem 2 of your previous assignment to compute a vector of missing values. You should use logical indexing to convert the missing values to zero so the missing values contribute nothing to the weighted sums of all the remaining digits.

Next, we want to put those values back into the original array. This turns out to require a little more work. We cannot simply use logical indexing or vector indexing with a vector of linear indices obtained using the function `find`, because doing so will assign the first missing value not to the first UPC code, but at the position of the first `NaN` accessed column by column. The other ones will also be misplaced in general.

Instead we have to use the function `find` on the above logical array to produce two vectors, one for the row and the other one for the column indices. We then have to use the `sort` function with two output arguments to sort the vector of row indices. The second output argument is then used to sort the vector of column indices. Next, we use the function `sub2ind` to produce a vector of linear indices. The first input argument of the `sub2ind` function must be the size of the array of UPC codes. This vector can then be used to place the computed missing values back to the array of UPC codes.

Display both the original array of UPC codes and the new one.

Finally check to make sure that the resulting UPC codes are correct by computing Eq. (1) found in the previous assignment.

Problem 2 [30 pts]

A 2D array named `COORD` is given to you in a file named `RingSquare.mat` that can be loaded when a Matlab program is run. In order for Matlab to be able to find the file, it has to be located in the Matlab path (containing a list of all the directors where a Matlab program may be stored). If the file is on the path, then the statement

```
load RingSquare
```

will load into Matlab's memory any variables that have been stored in that file. In our case here, the array `COORD` will be created with the values stored.

That array has the coordinates of a large number of points in 2 dimensions. The coordinates of each of the point are stored on each row of the array. The x-coordinates of these points have values that vary from -4 to 2 in a random fashion. Similarly the y-coordinates of the points have random values varying from -2 to 4.

Your program must be written so that it will work independent of the total number of points in the array. First have the user select during runtime one of the points in the array as the target point. You must display a prompt like

```
Enter a whole number from 1 to 1234:
```

if the total number of points in the array is actually given by 1234. The user must then enter a number between 1 and 1234 for the program to continue to run.

Next, we want to find all the points in `COORD` that obey a certain criterion. The figure here illustrates one of the possible cases. The target point is shown here by a red circle. Points in the ring portion of the figure lie within a distance of R_1 and R_2 (with $0 < R_1 < R_2$) from the target point. Here you should let $R_1 = 1$ and $R_2 = 2$. The figure also has a rectangular portion. Those points all located in the northwesterly direction of the target point.

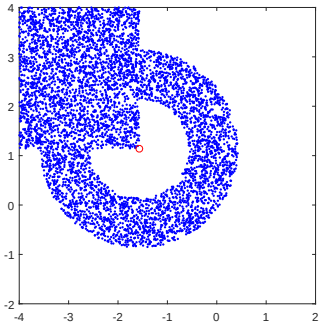
You need to identify all the points that belong to the figure and stored their coordinates in an array named `COORD2`. Display the total number of those points.

Since we are dealing with two-dimensional points, so we can plot those points in `COORD2`. Mark the points with a blue dot (marker `b.`). Use the command `hold on` so that you can place more items on the same graph. Finally plot the target point and mark it with a red circle (marker `ro`), and then use the command `hold off` to release the plot so that subsequent plots will no longer accumulate together. You can generate a graph to visualize the data using the following statements:

```
% Plot the selected points with a blue dot:
plot(COORD2(:,1),COORD2(:,2),'b.');
axis square;
hold on
% Plot the target point with a red circle:
plot(COORD(theTarget,1),COORD(theTarget,2),'ro');
hold off;
```

For the given dataset and for the case where the target point is 12789, the following graph is produced.

Figure 1: Resulting Graph for Target point 12789.



The figure will look differently depending on the target point chosen.

The output display is:

```
Enter the point to be used as the target.
Enter a whole number from 1 to 15230: 12789
Number of points in the figure is: 5828
```

Your program output should be similar to this example.