1 Pokemon Olympics

This is not really a pokemon thing - it is inspired by both pokemon and the 2016 Summer Olympics from this past August! Groups of pokemon are meeting up from around the world to battle each other and determine which pokemon species is the best!

You have been given the medal data in `pokemonMedalData.txt`. The data is given in order of ranking, such that the first row of data corresponds to the winning pokemon species with the most medals, the second row corresponds to the second place winner, etc. The first column designates the number of gold medals per species, the second column designates the number of silver medals, and the third column designates the number of bronze medals. To load the data into a MATLAB variable, you can use the following code:

```matlab
data = load('pokemonMedalData.txt');
```

Then you can use the `data` variable as a typical 2-D matrix.

1.1 Graph the Data

In a script file named `plotData.m`, draw in one figure (one set of axes) the following graphs:

- Plot the number of gold medals per species as a solid blue line (yellow is too difficult to see).
- Plot the number of silver medals per species as a solid black line.
- Plot the number of bronze medals per species as a solid red line.
- Plot the total number of medals per species using the default color that MATLAB assigns. Do not use built-in function `sum` for calculating the total as we would like you to practice with vectorized code and/or loops here.

Your figure should

- show the grid lines (i.e., use the command `grid on`),
- include an appropriate title (e.g. Number of Olympic Medals per Pokemon Species),
- include an x-axis label (i.e., Pokemon Species) and a y-axis label, and
- include a legend. Put it in the top left corner by using two additional parameters, 'location' and 'North-West', to the end of the argument list to the `legend` function. E.g., if the figure includes two curves the function call could be `legend('First curve description', 'Second curve description', 'location', 'NorthWest')`.

See Section 2.11 Introduction to Plotting in the textbook for more example code on plotting. (However, the explanation on the legend on p. 62-63 does not seem to work for MATLAB version 2016a. Use the example given above for placing the legend in a specific location.)
1.2 Joint Probability Distributions

Joint probability refers to probabilities based on two or more variables. Consider the following data below, which gives the number of students who play the piano and/or violin:

<table>
<thead>
<tr>
<th>Gender</th>
<th>Piano</th>
<th>Violin</th>
</tr>
</thead>
<tbody>
<tr>
<td>Male</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>Female</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

To calculate the probability of how many male and female students play each instrument, we must first calculate the total number of instruments played. Since there are 2 + 5 + 3 + 2 = 13 pets, the joint probabilities are as follows:

<table>
<thead>
<tr>
<th>Gender</th>
<th>Piano</th>
<th>Violin</th>
</tr>
</thead>
<tbody>
<tr>
<td>Male</td>
<td>2/12</td>
<td>5/12</td>
</tr>
<tr>
<td>Female</td>
<td>3/12</td>
<td>2/12</td>
</tr>
</tbody>
</table>

Dividing and rounding to three decimal places, this simplifies to

<table>
<thead>
<tr>
<th>Gender</th>
<th>Piano</th>
<th>Violin</th>
</tr>
</thead>
<tbody>
<tr>
<td>Male</td>
<td>0.167</td>
<td>0.417</td>
</tr>
<tr>
<td>Female</td>
<td>0.250</td>
<td>0.167</td>
</tr>
</tbody>
</table>

(We will not consider marginal probabilities here.)

Calculate the joint probabilities for the pokemon medal data. You will write two separate functions, one non-vectorized and the other vectorized, to do the calculations:

- Function `jointProb` has one input parameter, which is the pokemon medal data matrix (not the file name), and returns a matrix of joint probabilities. In order for you to practice element-by-element processing, this function must NOT use vectorized code. You must use nested loops to access each element in the matrix individually to calculate the total number of medals and use nested loops to store the probability values in a matrix.

- Function `jointProbVectorized` does the exact same calculations as `jointProb`, except that it uses vectorized code. You are allowed to use the built-in `sum` function here. You should be able to write the body of this function in no more than three lines of code.

In order to make sure that both functions do indeed perform the same calculations and therefore return the same matrix, test your functions in a script file and compare the results. You should use the given olympic medal data, as well as fabricated data of your own choosing that is a different size (i.e., not the same number of rows and columns as the pokemon medal data matrix). Name this file `jointProbTester.m`.

2 Finding Pokemon

Let us continue on our pokemon journey! In the game, you often must travel to find these pokemon, walking long distances to find these mystical creatures. Let’s see how many pokemon we can find!

2.1 Function `findPokemonInRadius`

Let’s say that there is matrix that contains the number of pokemon at each location, such as the 3x5 matrix below. Notice that there is 1 pokemon at row 3, column 3. Let’s say we start our journey at that location. If I take one step in any direction, including diagonals, the total number of pokemon I can find is 11. If I take another step out again, I can reach 34 pokemon. Let’s consider the blue squares to be at a radius of 1, and the green squares to be at a radius of 2 from our starting location. Just the start location itself is at radius 0.
As another example, below are the radii if I start at row 1, column 5, where the blue represents a radius of 1 from the starting point, the green a radius of 2, the red a radius of 3, and the orange a radius of 4. I can obtain 5 pokemon at radius 0, 8 pokemon at radius 1, 13 pokemon at radius 2, 22 pokemon at radius 3, and 34 pokemon at radius 4.

Here is one last example of the radii starting from row 5, column 3:

Notice that for each radius, you can create a smaller rectangular matrix that covers all of the locations that you need to analyze. Determine which row numbers and which column numbers you should start and end with, and then do a summation over that selected portion. In the third example, to obtain the squares at a radius of 1, I would want to sum over rows 4 and 5 and columns 2 through 4. Using this knowledge, implement the following function:

```matlab
function numPokemon = findPokemonInRadius(r, c, radius, locations)
% Returns the number of pokemon within a given radius from a given
% starting location in a matrix. If the radius encompasses an area larger
% than the matrix size, only include the portions that are reachable
% and stop at the boundaries of the matrix (see examples 2 and 3 above).
% r - the integer representing the row to start at
% c - the integer representing the column to start at
% radius - an integer representing the distance in all directions from a
% starting point
% locations - a matrix with the number of pokemon at each location
% numPokemon - the number of pokemon that can be obtained

You need to figure out how to do this summation in two different ways:

1. Use nested for loops to iterate through the 2-D array and sum up the number of pokemon.
2. Use the sum function (no loops!) to sum up the number of pokemon. Use the MATLAB documentation to help you!

You can keep both parts in the same file. You need to comment one version out when you are writing the other version, but make sure both parts are available for us to grade!

2.2 Function findNPokemon

Suppose we want to find \( n \) pokemon within an area of radius 100 starting from row \( r \), column \( c \). Is it possible? Implement the following function:
function radius = findNPokemon(r, c, n, locations)
    % Returns the smallest radius less than or equal to 100 at which at least
    % n total pokemon can be found when starting at row r column c in the
    % locations matrix. If the area within a radius of 100 contains fewer
    % than n pokemon, return -1 instead.
    % r - the integer representing the row at which to start
    % c - the integer representing the column at which to start
    % n - the number of pokemon to find
    % locations - a matrix with the number of pokemon at each location

    You may use findPokemonInRadius to help you. Note that this is the perfect place to use a while loop,
    since you do not know exactly when you will stop walking.

2.3 Script findPokemonTester.m

This is where you can test your findPokemonInRadius and findNPokemon functions. A pre-made locations
matrix has been included for you to test your functions. Create your own matrices to test other scenarios
(e.g. all ones, or all zeros). Make sure to print your results neatly using fprintf.

3 Self-check list

The following is a list of the minimum necessary criteria that your assignment must meet in order to be
considered satisfactory. Failure to satisfy any of these conditions will result in an immediate request to
resubmit your assignment. Save yourself and the graders time and effort by going over it before submitting
your assignment for the first time.

Note that, although all of these are necessary, meeting all of them might still not be sufficient to consider
your submission satisfactory. We cannot list everything that could be possibly wrong with any particular
assignment!

- Comment your code! If any of your functions is not properly commented, regarding function purpose
  and input/output arguments, you will be asked to resubmit.

- Suppress all unnecessary output by placing semicolons (;) appropriately. At the same time, make sure
  that all output that your program intentionally produces is formatted in a user-friendly way.

- Make sure your functions’ names are exactly the ones we have specified, including case.

- Check that the number and order of input and output arguments for each of the functions matches
  exactly the specifications we have given.

- Test each one of your functions independently, whenever possible, or write short scripts to test them.

- Check that your scripts do not crash (i.e., end unexpectedly with an error message) or run into infinite
  loops. Check this by running each script several times in a row. Before each test run, you should type
  the commands clear all; close all; to delete all variables in the workspace and close all figure
  windows.

4 Submission instructions

1. Upload files plotData.m, jointProb.m, jointProbVectorized.m, jointProbTester.m, findPokemonInRadius.m,
   findNPokemon.m, and findPokemonTester.m to CMS in the submission area corresponding to Assignment
   1b before the deadline.

2. When the scores are released, read the grader’s feedback carefully.

3. If you need to resubmit, fix all the problems and go back to Step 1! Otherwise you are done with this
   assignment. Well done!