Notice - Testing plot outputs:

For this homework, some of your outputs will be plots. We have provided you with the function `checkPlots()` to help you test the plot outputs of your functions. Use

```
help checkPlots
```

for a full description of how the function works, including what inputs to call with it. Below is the example from the documentation explaining how to run the function.

If you have a function called `testFunc` and the following test case:

```
testFunc(30, true, {'cats', 'dogs'})
```

To check the plot produced by `testFunc` against the solution function `testFunc_soln`, for this test case you would run:

```
[match, desc] = checkPlots('testFunc', 30, true, {'cats', 'dogs'})
```

Happy Coding!
~Homework Team

Function Name: `badRobot`

Inputs:

1. (*char*) The name of an Excel file

Outputs:

1. (*cell*) The completed data set

Topics: (*Interpolation*), (*High Level I/O*)

Background:

After taking MATLAB, you fall in love with coding and decide to add a minor in Computer Science. In CS 3630, you are trying to program your robot to move to the correct location after detecting the correct image. However, your robot has a faulty LIDAR sensor, and some of the collected data is corrupted. Oh no! The project is due tomorrow, so you need to find a way to predict what the missing values should have been so that the robot can complete its mission. MATLAB got you into this mess, but it can also help you get out of it.

Function Description:

You have an Excel sheet with headers and two columns of data. The first column represents x data and the second columns represents y data. There are missing values in the y data represented by the string 'Error', find them and use linear interpolation to predict the value. Output the updated data as a cell array.

Example:

`data.xlsx` →

time	distance
0	6
1	Error
3	10
5	11
6	Error
7	12

```
updated = badRobot('data.xlsx')
```

`updated` →

time	distance
0	6
1	7.33
3	10
5	11
6	11.5
7	12

Notes:

- There won't be two consecutive missing values. The first and last value won't be 'Error'.
- Round interpolated values to 2 decimal places.

Function Name: `surveyProcessing`

Inputs:
1. (*double*) 1 by M vector of x values
2. (*double*) 1 by M vector of y values
3. (*double*) polynomial order to fit data to

Outputs:
1. (*double*) 1 by M vector of predicted y values using model with outliers
2. (*double*) 1 by M vector of predicted y values using model without outliers

Banned Functions:

```
isOutlier()
```

Topics: (*Data manipulation*), (*Curve fitting*)

Background:

Outliers are data points that differ significantly from other points. When fitting data to a model, outliers can cause the model to vary widely. In this homework, we'll explore building a model to fit the data before and after removing outliers.

Function Description:

Given a set of data, find the predicted y-values using a model with outliers, and a model with outliers removed using the following procedure:
1. Find Q1, the median of the lower half of the y-values
2. Find Q3, the median of the upper half of the y-values
3. Find the interquartile range (IQR), the difference between Q1 and Q3
4. Classify data points as outliers if they're below Q1 - 1.5*IQR, or above Q3 + 1.5*IQR
5. Fit the data to an N-order polynomial (where N is the third input)
6. Evaluate the x-values using this polynomial and output the predicted y-values
7. Remove the outliers, and fit the remaining data to an N-order polynomial
8. Evaluate the x-values using this polynomial and output the predicted y-values

Notes:
- You can use the median() function to calculate medians.
- Output both sets of y values in the same order that you were originally given.
- You may see a warning about poorly conditioned polynomials, you can ignore it.
- For fun, try comparing the SSE or R^2 of both models!

Function Name: `publixMarathon`

Inputs:

1. (*double*) a 1xN vector of times (in hours)
2. (*cell*) a 4x(N+1) matrix of runner names and corresponding velocities at different times (in km/hr)

Outputs:

1. (*double*) a scalar time value of when Alex completed the race

Plot Outputs:

1. A 2x2 subplot figure with the following attributes:
    a. The first subplot contains distance with respect to time for all runners and a special marker where Alex completed the race
    b. The second subplot contains velocity with respect to time for all runners
    c. The third subplot contains acceleration with respect to time for all runners
    d. The fourth subplot contains jerk with respect to time for all runners

Topics: (*plotting*), (*interpolation*), (*integration*), (*differentiation*)

Background:

It's the last turn of the race and you're waiting anxiously for your favorite MATLAB TA Alex Lehman to complete the Publix Marathon. It's anyone's game as nobody has yet finished the race, yet you know in your heart that Alex will pull through as the victor from all his training. Suddenly, it happens: Alex pulls around the corner with his fully sponsored GT Adidas gear, sprinting at full speed with MATLAB Mobile open on his iPhone—no wonder he's so fast! Within a moment's time he crosses the finish line and the crowd goes wild! With him raised up on the 1st place pedestal, his metal of pure gold shining against the morning sun, and a dozen girls fighting for his undivided attention, he appears to be a legend among Atlanta. Captivated by his perfection, you decide to calculate out his race statistics in order to understand what it would take for you to live up to his magnificence, a perpetual euphoric state you refuse to be further deprived of.

Function Description:

You are creating a figure with 4 subplots (2x2). You are given a vector of different times and a cell array of 4 runners' names and corresponding velocities to the given times. There will always be exactly 4 runners and the first column will always be the names of each runner. The next columns after the first column correspond to that runner's velocity at each time from the first input. Finally, output the time (rounded to two decimal points) Alex finishes the marathon.

(continued...)

In subplots, do the following:

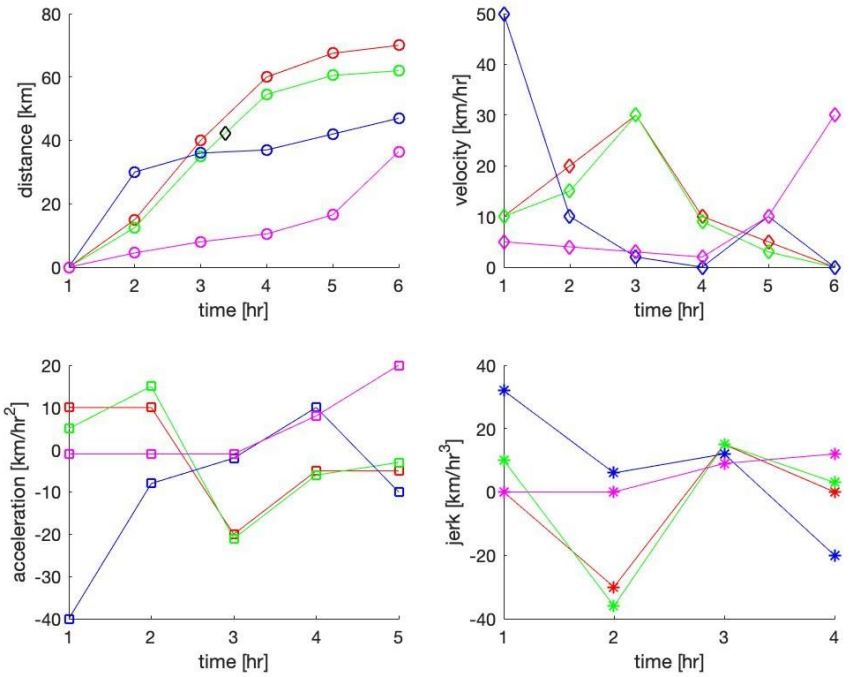
1. Subplot 1
    a. Integrate the velocities to get distances
    b. Plot the each runner distance as it corresponds to the given time. The first runner distance should be in red, the second in green, the third in blue, and the last in magenta. Use continuous lines ('-') and denote points with a circle ('o').
    c. Mark the time when Alex finishes the marathon (this time should be rounded to 2 decimal points) on your subplot as a black ('k') diamond ('d').
    d. Set Alex's rounded time when he crossed the finish line to be your only variable output for your function.
    e. Label the x-axis 'time [hr]' and the y-axis 'distance [km]'
2. Subplot 2
    a. Plot the each runner's velocity as it corresponds to the given time. The first runner distance should be in red, the second in green, the third in blue, and the last in magenta. Use continuous lines ('-') and denote points with a diamond ('d').
    b. Label the x-axis 'time [hr]' and the y-axis 'velocity [km/hr]'
3. Subplot 3
    a. Take the derivative of the velocity to get the second time derivative of position: acceleration.
    b. Plot the each runner's acceleration as it corresponds to the given time. Since the differential function leads you have a vector of length N-1, do not include the last time in the time vector when plotting. The first runner distance should be in red, the second in green, the third in blue, and the last in magenta. Use continuous lines ('-') and denote points with a square ('s').
    c. Label the x-axis 'time [hr]' and the y-axis 'acceleration [km/hr^2]'
4. Subplot 4
    a. Take the derivative of the acceleration to get the third time derivative of position: jerk.
    b. Plot each runner's jerk as it corresponds to the given time. Since the differential function leads you have a vector of length N-2, do not include the last two times in the time vector when plotting. The first runner distance should be in red, the second in green, the third in blue, and the last in magenta. Use continuous lines ('-') and denote points with an asterix ('*').
    c. Label the x-axis 'time [hr]' and the y-axis 'jerk [km/hr^3]'

Example:

```
t = [1 2 3 4 5 6];
ca= {'Michael' 10 20 30 10  5  0;
     'Alex'   10 15 30  9  3  0;
     'Satvik' 50 10  2  0 10  0;
     'Blake'   5  4  3  2 10 30};
xTime = publixMarathon(t,ca)
xTime → 3.37
```

(continued...)

Plot Output:



Notes:

- 'Alex' is guaranteed to be one of the runners given
- For reference, the distance of a marathon is 42.195 kilometers.
- In the cell array input (input 2), the second column given corresponds to the first index of the time vector input (input 1), the third cell array column corresponds to the second index of the time vector, and so on.
- The first runner should always be plotted in red, the second in green, the third in blue, and the fourth in magenta.
- The times given will always be in ascending order
- Continue plotting if the runner keeps running after the finish line, sometimes they need to get away from the adoring fans

Hints:

- The subplot(), xlabel(), and ylabel() plot detailing functions may come in handy
- You may have to use the different distances as your first input and the times as your second input in your spline function to get the time at which Alex crosses the finish line
- Linear interpolation should be used to find the exact time Alex finishes the marathon, do not use spline or any other interpolation method

Function Name: `noisyData`

Inputs:

1. (*double*) A 2xN array of timestamped data
2. (*double*) The filtering percentage (as a number 0-1)

Plot Outputs:

1. A plot showing both the original data and your filtered data

Topics: (*numerical methods*), (*plotting*)

Background:

As an engineer, you are often going to be collecting data from sensors and will need to process it. Often, sensors can give back faulty data or something in the environment has caused your sensor data to be less than accurate, or noisy. Therefore, you often need to filter your data to make it smoother and easier to use or display, and MATLAB can help with that!

Function Description:

You are given a 2xN array of timestamped sensor data where the second row contains the sensor data and the first row contains the time it was taken and a filtering percentage *r*, which is how much you want to rely on your sensor data vs your filtered data (0 means rely entirely on your sensor data, 1 means filter out your of your sensor data). You should filter your sensor data such that $y_n = r * y_{n-1} + (1-r) * sensor_n$. Basically, this means your filtered data at timestamp *n* (y_n) is a weighted sum of the filtered data from the previous timestamp (y_{n-1}) and the sensor data at your current timestamp ($sensor_n$). *r* is the filtering percentage, so if $r = .7$, 70% of y_n comes from y_{n-1} and 30% comes from your actual sensor data. Assume the first data point of the filtered data is the first sensor value

After filtering the sensor data, first plot your original data in a red dashed-dotted line, then plot your filtered data in a blue solid line on the same graph. Format your graph with `axis tight`.

Example:

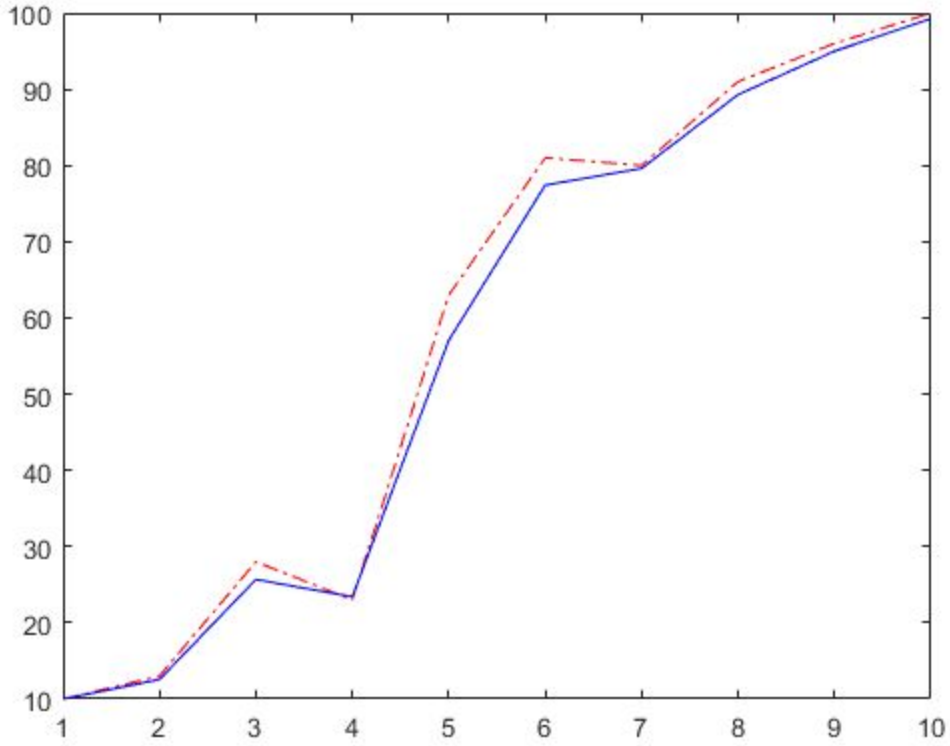
```
data = [ 1  2  3  4  5  6  7  8  9  10;
         10 12 28 23 63 81 80 91 96 100]
r = .15
noisyData(data,r)

filtered_data =>
[10 11.7 25.555 23.3833 57.0575 77.4086 79.6113 89.2917 94.9938 99.2491]
```

(note: you SHOULD NOT output or display this vector, it is merely to help you understand how the problem works)

(continued...)

```
Plot Output =>
```



Notes:

- Your data will always be in ascending time order, meaning each data point is timestamped after any data points that precede it in the array.
- All plotting should be timestamp (x-axis) vs sensor value (y-axis).

Extra Credit

Function Name: `histogram`

Inputs:

1. (*char*) Name of input file that contains one line of positive integers separated by commas

Plot Outputs:

1. A colorful histogram that represents the frequency of each number

Topics: (*plotting*), (*file I/O*)

Background:

You have been given a set of mysterious data that consists of integers given in a text file. In order to analyze the data, you'll have to create a histogram.

Function Description:

Create a histogram using the following rules:

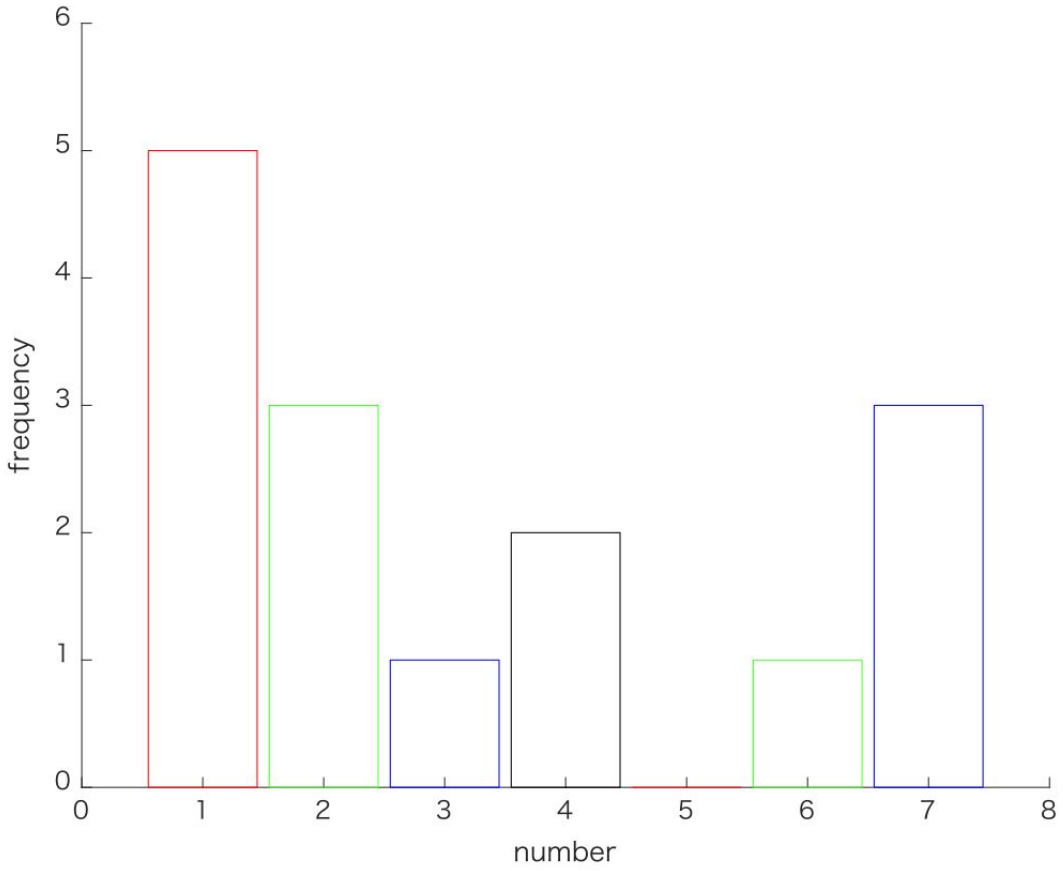
- Each bar is a hollow rectangle that is horizontally centered around the number it represents
- Each bar should have a width of 0.9 units
- Each bar should vertically start at 0 and have a height that is equal to the frequency of that number
- The color of the outline of each bar should alternate between pure red, pure green, pure blue, and pure black, starting with red
- Set the x-axis from the smallest number - 1 to the largest number + 1
- Set the y-axis from 0 to highest frequency + 1
- Title the x-axis `'number'` and the y-axis `'frequency'`

Example:

`data.txt` →

```
3,1,6,1,1,4,7,1,2,2,7,4,2,7,1
```

```
>> histogram('data.txt');
```



Notes:

- If a number has a frequency of 0, it should be plotted as a rectangle of height 0.
- The text file will always be only 1 line of comma separated integers.