Notice

Several problems on this homework require you to create plot outputs, which cannot be checked for visual equality using any built-in tool in MATLAB. Hence, we have provided the function `checkPlots()` with this week's homework, which compares the plot output of your function and the solution function for equivalence. 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'})
```

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

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

If you have any questions regarding `checkPlots` or how this week's homework will be graded, please direct them to the "HW Grading Issues and Questions" pinned Piazza post:
https://piazza.com/class/ixkvuiueg6l6kn?cid=1089

Happy Coding!
~Homework Team

Function Name: `spaceship`

Inputs:

1. (*double*) A 3xN array of time, temperature, and speed data points
2. (*double*) The distance to the surface of the Earth from the spaceship's start position (m)

Outputs:

1. (*char*) A sentence describing results of the calculations based on the spaceship data

Function Description:

Oh no, there's a spaceship plummeting towards the Earth! When the spaceship re-entered the atmosphere, NASA was able to track it and begin recording data. Unfortunately, the connection was lost a short time after the ship's warning signal began to sound and before it landed. At a time like this, NASA needs someone with top-notch MATLAB skills to make some calculations and determine the fate of the spaceship. The thermal shielding of this spaceship can withstand temperatures up to 2000 ºC, before the spaceship is doomed. The warning alarm will begin to sound at a temperature of 1260ºC, as the shielding begins to deteriorate. The input array of data will always have the following format: the first row in the input array represents N time data points (s), the second row represents N temperature data points (ºC), and the third row represents N speed data points (m/s). You will determine the following:

1. Did the spaceship burn up before reaching the ground? (exceed 2000ºC)
2. How far was the spaceship from the Earth when it reached the warning temperature? (1260ºC)

Based on the results of these calculations, output the following sentence:

```
'The spaceship's warning alarm sounded <distance> meters from the Earth.
                    Luckily, the ship landed safely!'
                                  OR
'The spaceship's warning alarm sounded <distance> meters from the Earth.
                 Unfortunately, the ship did not make it.'
```

Notes:

- Round ONLY your final calculations to two decimal places.
- Use `%0.2f` to display two decimal places in the output string.
- The input of the distance from the spaceship's start position to the Earth's surface is the distance when the very first measurement was taken.
- The temperature data is **linear** with respect to position, not time or speed; therefore, you will need to find the distance traveled towards the earth at each time.

Hints:

- `cumtrapz()` will be useful.

Function Name: `scoreBoard`

Inputs:

1. *(cell)* A 1x2 cell array with the names of the 2 teams
2. *(double)* A 3xN vector of the team's points scored through the game

Outputs:

1. *(char)* Game report about the overall leads and winners

Plot Outputs:

1. Both teams scores compared over the total time of the game

Function Description:

It's basketball time! What better to do than show off your data visualization techniques in MATLAB! You decide to make some great plots of the game score over time with your own commentary included. From the second input, you gather the first and second team's scoring patterns (rows 1 and 2, respectively) over time (row 3, minutes into the game). For example, the input of points scored may look like this:

```
  2     0     3     0     1     0     0     ...     1
  0     1     0     3     0     1     3     ...     0
0.5   1.9   2.4   2.7   3.3   3.6   3.9      ...   39.8
```

The points will not be cumulatively added for you as the input only represents which team scored how many points at between each timestamp.

Here is some of your design criteria for your "baller"-er plots:

- The first team's points will be plotted in red and the second team's points will be plotted in blue.
- The bounds of your graph should be `0` to `40` minutes along this the x-axis (because this is college ball), and `0` to the maximum score plus `5` on the y-axis
- You also separate the two halves (at `20` min) by a black vertical line that reaches the top of your plot (again, `5` points above the maximum score)
- The Title of your plot will be `'<Team 1 Name> vs. <Team 2 Name>'`
- The y-axis title is `'points'`
- The x-axis title is `'minutes'`

After creating your plot, you will synthesize the final data for those less sports-savvy than yourself. Using the score information, figure out who had the lead for the majority of the game-time. If the one team had the lead for more than 75% of the game time (30 minutes, non-inclusive) *and* won, your game report will read:

```
'<teamName> totally dominated this game and took home the win from
                        <otherTeamName>!'
```

However, if one team led for more than 75% of the game time (30 minutes, non-inclusive) but lost, your report will read:

```
'<teamName> totally dominated this game but <otherTeamName> stole the win
                                tonight!'
```

Finally, if the scores were close (no team led for more than 30 minutes of the game), your report will read:

```
'Although a fairly evenly matched game, <teamName> came out on top over
                        <otherTeamName> tonight!'
```

Notes:

- There will never be a tie.
- The time steps will not necessarily be evenly spaced.
- For more great graphics about March Madness, see here: http://www.gilliganondata.com/index.php/2009/03/28/data-visualization-march-madness-style/

Hints:

- To plot a vertical line, only 2 x- and y-points are needed. One of those values will not change and the other will span the entire length of its bounds.
- Although the topics does not directly relate to this problem, there is a very helpful function you learned this week to cumulatively add up the points.

Function Name: `greenshield`

Inputs:

1. *(char)* The name of an excel file containing traffic data

Plot Outputs:

1. Subplot with Speed vs. Density, Speed vs. Flow Rate, and Flow Rate vs. Density plots

Function Description:

You live in Atlanta and the traffic sucks. You decide to take transit one day and with *all the free time you now have*, you break out your laptop and start to conduct some traffic analysis. Greenshield's model is your prefered as it uses a linear relationship between traffic density and traffic speed. Classic choice.

The flow rate (q), speed (u), and density (k) will be stored in the provided excel file. The first row of the excel file will be a header row. Each parameter will be stored in its own column, titled `'flow rate (q)'`, `'speed (u)'`, and `'density (k)'` respectively. The excel file is guaranteed to have these columns, although they will not necessarily be in that order and there may be extraneous columns.

After extracting the data, you will make three plots: Speed vs. Density, Speed vs. Flow Rate, and Flow Rate vs. Density. Use subplots to place your Speed vs. Density plot in the top left, the Speed vs. Flow Rate plot in the top right, and the Flow Rate vs. Density plot in the bottom left. Each of these subplots should contain all relevant data points plotted as **black asterisks**, one regression line, and one "critical" line. The colors and formulas for these lines are specified below. The axis bounds should be specified from 0 to the max value of that axis data set. While you don't need to title the plots, be sure to include axis titles like those given here:

```
Density, k [veh/mi/ln]
Speed,  u [mi/hr]
Flow Rate, q [veh/hr/ln]
```

First, find the linear relationship between Speed and Density by data-fitting. Use those coefficients to calculate the 2nd-degree polynomial curves for modeling Speed vs. Flow Rate and Flow Rate vs. Density. Those equations are provided here:

In which i *is the intercept of the linear fit,* r *is the slope, and R is the absolute value of the slope*

For the Flow Rate vs. Density plot: $q = i*k + r*(k^2)$

For the Speed vs. Flow Rate plot: $q = (i*u - u^2) / R$

These equations (with actual values) should also be printed using the `text()` function, and plotted on the subplots in various colored lines as specified in the table below:

Plot	Text Position	Line Color
Speed vs. Density	3, 5	green
Speed vs. Flow Rate	500, 10	cyan
Flow Rate vs. Density	10, 250	red

Finally, consistent with Greenshield's Model, the critical (or optimal in specific cases) Speed is one half of the maximum Speed in the data set. From this value (or the closest in the data set), find the corresponding critical Density and critical Flow Rate values. Plot these lines in yellow across all subplots such that the Flow Rate vs. Density plot has a vertical line at critical Density, the Speed vs. Flow Rate plot has a horizontal line passing through critical Speed, and the Speed vs. Density plot has a 2 lines: vertical from 0 to critical Speed, and horizontal from critical Density to max Density (or the end of the plot).

Notes:

- When plotting lines, it will be helpful to sort the values in order of the dependent variable so that the line is continuous and doesn't jump across the points
- Round to 3 decimal places when displaying on the plots. Do not round during calculations.
- To learn more about Greenshield's Model, look here: http://faculty.unlv.edu/pushkin/TrafficFlowTheory.pdf

Function Name: `findLocalMin`

Inputs:

1. *(double)* A string of a polynomial function (Ex. '`f(x) = x^2`')
2. *(double)* A lower x-axis bound
3. *(double)* An upper x-axis bound
4. (*double)* `x0`, a starting x axis value for the local minimum search

Outputs:

1. *(double)* The coordinates of the local minimum

Plot Outputs:

1. A plot of the given function with tangent line segments used for finding the local minimum

Function Description:

Have you ever looked at a polynomial curve and thought, "Wow, I really wish I knew what that local minimum was."? Well, you're in luck! Today we're going to figure out the local minimums to all those pesky polynomials that you have long since given up on. For any normal programming language, this task may be impossible… good thing that MATLAB makes the impossible possible!

You will be provided with a string representing some polynomial function. The left of the equal sign will always be formated as `'f(<variable>)'`. The right of the equal sign will be a polynomial in terms of that variable. Below are some valid example string inputs:

```
'f(x) = 3x^2'
'f(z) = 12z^3 + 2.2z + 5'
'f(q) = q^5 + -2q^2 + 5q'
```

A valid polynomial term looks like `'<coefficient><variable>^<power>'`, where `<coefficient>` could be negative (preceded with a `'-'`) or positive. Terms to the 1 power and 0 power will **not** have a `'^1'` or a `'x^0'`. Each polynomial term will be separated by a `'<space>+<space>'` and there will always be a space before and after the equal sign. `<variable>` will always be a single character (not a string). Powers will always be positive integers.

First, plot the function with 100 **evenly** spaced points between the lower x-axis bound and upper x-axis bound (inclusive) as a solid blue line. Label the y-axis `'f(<variable>)'` and the x-axis `'<variable>'`, where `<variable>` is given in the input function string. To find the local minimum, we are going to go through a certain range of x-axis values until the slope of the function undergoes a sign change.

Starting at `x0`, calculate the slope of the function at the current x-axis value. Use this slope to plot a red line segment between `(x - 5)` and `(x + 5)`, which represents the tangent to the curve.

If the slope is positive, decrease your current `x` value by `1`. If the slope is negative, increase your current `x` value by `1`. Repeat this process until the slope changes sign. When it does change sign, divide the current `x` increment change by half (after a sign change, `x` will increase or decrease by `.5`, then `.25`, etc.). Continue this process until your slope is within the range `0 ± 0.1` (inclusive). Plot the final tangent for when the slope is within the range. Your final plot's axis bounds should be the same as they were when you plotted the original function (blue line) **without any** tangents (red line segments). Output the coordinates of the local minimum found as a vector: `[xmin ymin]`

Notes:

- The given function is guaranteed to have a local minimum in the domain of the lower/upper bounds.
- Your lower and upper x bounds are guaranteed to be integers.
- `x0` may not be an integer.
- The first term in the polynomial function string will always be the highest degree polynomial term.
- Round your slope to the 3rd decimal place each time you calculate it.
- Round the bounds (x and y values) of each tangent (red line segments) to the 3rd decimal place.
- Round your final local minimum coordinates to the 3rd decimal place.
- A `-1` coefficient term will always be written as `'-1x^3'` for example, NOT `'-x^3'`.

Hints:

- Consider what the output of the `axis` function is.
- Think about what the highest order term tells you about the function.

Extra Credit

Function Name: `rollerCoaster`

Inputs:

1. *(char)* Name of an Excel file containing the design of the roller coaster

Plot Outputs:

1. Plot of the roller coaster

Background

Congratulations! You have been hired by Six Flags to design their newest roller coaster at Six Flags Over Georgia. While you've got an idea of the loops and turns you want to use in your new coaster, you want to have a nice way to visualize it so that you can show it to the executives. For such an important assignment, you naturally turn to MATLAB to help you plot your roller coaster!

Function Description:

Write a function that takes in an Excel file containing your plan for the roller coaster. The first row in the Excel file will contain the headers `'Name'`, `'StartCoordinate'`, `'NumSides'`, `'Radius'`, and `'AdditionalPoints'` in that order. Each row will represent one shape to graph. The possible shapes and the necessary information to graph them are listed below. Starting at `(0,0)`, graph the shapes from top to bottom.

Name	Coordinate	NumSides	Radius	NumPoints
`ellipse`	`'(x,y)'` coordinates of the point on the ellipse with the lowest y-value	N/A	`'(Semimajor axis, Semiminor axis)'`	Number of points to use when graphing the ellipse
`helix(x|y)`	`'(x,y)'` coordinates of the beginning of the helix	Number of spirals	radius of the Helix	Number of points to use when graphing the helix
`loop`	`'(x,y)'` coordinates at the beginning of the loop	N/A	`'(Radius of larger circle, radius of smaller circle)'`	Number of points to use when graphing each segment of the loop
`arc`	`N/A`	`1|2`	Radius of the circle	N/A

The ellipse starts and ends at the point with the smallest y-value, and is guaranteed to be longer horizontally than vertically.

The helix should be generated with the specified number of spirals and an additional half-spiral. Since the helix is a three-dimensional figure, a helper function called `projection` has been provided to you. It takes in the x, y, and z coordinates of the helix and returns the x and y coordinates of the 2D projection that will be used. The helix should be "right handed": it should follow the right hand rule. Graph the helix using radians.

The loop consists of two 90 degree arcs of the larger circles in the picture linked above and a 180 degree arc of the smaller circle. It starts from the left side and goes to the right. Additionally, add straight lines on each end of the loop with length equal to the larger radius.

The arc should travel between the two adjacent shapes. To help you, a helper function called `arcGenerator()` has been provided. `arcGenerator()` takes in two points, each formatted as a 1x2 vector of `[x,y]`, and an angle and returns two arcs between those points with a subtending angle equal to the third input. Input the endpoint of the shape before the arc and the beginning of the shape after the arc, and use geometry to determine the angle for the given points and radius. The 1 or 2 in the Excel sheet determines if you should use the first or second output of `arcGenerator()`.

Notes:

- The helix name will either be formatted as `'helixx'` or `'helixy'`. `'helixx'` should travel right, along the x-axis, while `'helixy'` should travel down from the starting position, along the y-axis.
- An arc is guaranteed to either come between two other shapes or be at the very beginning, in which case it should start at `(0,0)`.
- The `projection()` and `arcGenerator()` functions will map the beginning of the helix and arc to `(0,0)`.
- `arcGenerator()` uses radians as its input.
- `arcGenerator()` may display a line between the two endpoints: this is expected.
- The first row of each of the outputs of `arcGenerator` outputs is the x-values of the arc, while the second row is the y-values.
- Set `axis square` and `grid off`.

Hints:

- It will be helpful to generate the points for each shape parametrically before graphing them.
- Look at the solution and the excel files for an idea of what your output should look like.
- Try adding additional shapes and designing your own coasters!