Drill Problem #1

Function Name: topPun

Inputs:

1. (*char*) A string of the name of a recruit enrolled in the academy
2. (*double*) A vector containing the scores earned each day

Outputs:

(*none*)

Plots:

1. A plot of the total score earned by the recruit

Function Description:

Welcome to your first day on the job as an analyst at Top Pun, an intense 9-week long program that trains the best of the best in the art of the pun. The recruits you'll be working with are masters of the Pundamentals, and may even think that they can Cruise through this course. However, they'll soon learn that they're way in over their heads. By the end of this program, graduates will possess a degree in Advanced Homophonics, allowing them to Kil-more words than a teenager's Twitter account.

Each day of the program, recruits receive scores based on the number of creative and worthy puns that they make. As our new technical analyst, your job is to keep track of each recruit's progress over the course of the program. You will be given a recruit's name and the scores that he or she earned during each day of training. You are not guaranteed to have a fixed number of scores, but you know that the first score will be the points earned during Day 1, and each subsequent score will correspond to the following day.

Given a string of the recruit's name and a vector containing these scores, write a MATLAB function that plots the recruit's total cumulative score over days. You should add the point `(0,0)` to the beginning of you data, as this indicates progress from Day 0 to Day 1. For example, if your vector of scores is `[2,3,1]`, then you should plot the points `(0,0)`; `(1,2)`; `(2,5)`; and `(3,6)`. Your plot should be formatted as follows:

- Your line style should be solid red with a circle marker at each point.
- The title should read `'<name>''s Total Score'` where `<name>` is replaced by the recruit's name.
- Label your x-axis `'Day'`.
- Label your y-axis `'Score'`.

Hints:

- The function `cumsum()` can do all the heavy lifting.

Drill Problem #2

Function Name: fieldGoalDenied

Inputs:

1. (*double*) A 2xN data array of a football's velocity over time after a kick (m/s)
2. (*double*) A 1x2 vector of Patrick Gamble's vertical leap at time t

Outputs:

1. (*logical*) Whether or not the kick is blocked

Plots:

1. Three subplots containing the velocity, the first derivative describing acceleration, and the integral describing the position over time, along with Patrick's jump.

Function Description:

The Miracle on Techwood Drive has captured the attention of thousands of Yellow Jackets. But as an engineer, you don't like miracles and surprises – you want to know ahead of time what will happen. You decide to write a MATLAB function that takes in a 2xN vector of data points describing the football's velocity over time, the first row being velocity and the second row being the time at the respective velocity, and will create three subplots (1x3) that plot the original data over time, the acceleration of the football over time, and the position of the football over time. Calculate the football's position over time by numerical integration of the data via trapezoidal integration. Further, calculate the acceleration over time via numerical differentiation; when plotting the acceleration, do not plot the first time value.

In addition to the football's velocity, you also have a record of Patrick Gamble's vertical jump height at a specific time t during a field goal. The first index of the second input is the height of his jump, and the second index is the time at which he reaches this height. If Patrick's position at that time t is higher than (or at the same height as) the football's height at time t, the field goal is denied (and the output will be true). Plot Gamble's position at time t, and plot it on the football's position subplot. To calculate the football's position at time t, fit the data to a seventh order polynomial. The data given is guaranteed to be greater than seven data points.

Notes:

- The data should be plotted as points with a black line for velocity, blue line for acceleration, and magenta for the ball position. Patrick's jump height should be a plotted with a green plus sign, and the position of the ball at that time plotted with a red circle.
- Label the x-axis as `'Time'` and the y-axis `'Velocity'`, `'Acceleration'`, and `'Position'`, respectively, for the first, second, and third subplot.

Hints:

- The `diff()`, `cumtrapz()`, and `polyfit()` functions are for numerical methods.
- The `subplot()`, `xlabel()` and `ylabel()` functions will be useful for plotting.

Drill Problem #3

Function Name: metaData

Inputs:

1. (*char*) The string of an Excel (`'.xlsx'`) file name

Outputs:

(*none*)

Plots:

1. A plot containing a `2xN` array of subplots according to the following description.

Function Description:

Imagine you are a student in a MATLAB class at Georgia Tech, and you must complete a homework problem about graphing data in order to keep your homework grade up. The problem requires you to write a function called `metaData` that reads in an Excel file and plots the data it contains in a series of subplots. The Excel file will contain multiple columns of numerical data each with a header. The data will be plotted in multiple subplots. The first column of data are the x values for all of the subplots. Each of the other columns of data will be the y values for a corresponding subplot. Subplot 1 should contain the first column of y data, subplot 2 should contain the second column of y data, and so on. Every axis should be labeled with the column header from the data plotted along that axis. The dimensions of the subplot array should be `2xN`, and the first row should be plotted before the second row.

```
N = ceil(number of subplots / 2)
```

Notes:

- The excel file will be formatted with the headers in the first row and the data in columns.
- The data should be plotted as black points with black lines along the graph.
- There will be no non-numerical data or extraneous columns of data.

Hints:

- The `subplot()`, `xlabel()` and `ylabel()` functions may be useful.

Drill Problem #4

Function Name: plotShapes

Inputs:

1. (*char*) A string of the shape to be plotted
2. (*double*) The edge/diameter length of the shape
3. (*double*) The number of degrees to rotate the shape
4. (*double*) A 1x2 vector representing a translation

Outputs:

(*none*)

Plot Outputs:

1. A plot of the specified shape

Function Description:

Write a function that can plot a square or a circle of a given size; this function should first rotate and translate the shape by the given input specifications, and then plot it. The first input to the function will be a string, either `'square'` or `'circle'`. If the first input is `'square'`, then the second input describes the edge length; if the first input is `'circle'`, the second input describes the diameter.

The shape should then be rotated clockwise by the number of degrees specified in the third input and translated by the amount designated in the fourth input. The translation input will be given in the form of

`[<x-coordinate shift>, <y-coordinate shift>]`

And should be applied to the entire shape. The object that you plot should be centered at this location. An input of `[0, 0]` as the fourth input should be centered at the origin.

To rotate the shape, use the 2D clockwise rotation matrix from geometry:

$$\begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix}$$

- Any circle you plot should be composed of 100 evenly spaced points.
- The square should have a point at each corner (no points in between)
- The plot should be made with a **black** line connecting the shape's points.
- Both the x and y axis should go from (- size) to (+ size) where 'size' is the second input.
- The axis should be set to `square`.

Notes:

- You do not need to worry about the translation putting the object out of the axis range.
- You will only be given a square or circle to plot.

Hints:

- To rotate pairs of x-y coordinates, separately multiply each pair by the rotation matrix.
- `cosd()` and `sind()` take in degrees instead of radians.
- Rotating a circle doesn't change anything.

Drill Problem #5

Function Name: intervention

Inputs:

1. (*struct*) A 1x1 structure with data to plot

Outputs:

1. (*double*) A 1x3 vector representing the R^2 values for the given data

Output Plots:

1. Three subplots containing linear, spline, and polynomial values plotted with original data

Function Description:

You've just been hired by Goliath National Bank and discovered that your sweet new corporate job comes complete with a multimedia lab to create all sorts of charts and graphs. While no one understands what you do ('ha, PLEASE."), you create some graphs for a presentation and are instantly hooked. You start making bar graphs of your favorite pies, pie charts of your favorite milkshakes, and linear regression plots regarding anticipation levels of the new movie, Regression. Your charts have become a regular companion on outings with your friends. But the most recent time you went to meet with you friends, you walked into an intervention where they said the graphs had to go. Because of this, you decide to write a function in MATLAB to determine the most accurate method to plot and interpolate your friends' opinions of your charts.

You are given a single structure with 6 fields: `'X'`, `'Y'`, `'minx'`, `'maxX'`, `'numPoints'`, and `'Power'`, as described in the table below.

Field Name	Value
X	The original x data points
Y	The original y data points
minX	The minimum x value to use in the interpolated data
maxX	The maximum x value to use in the interpolated data
numPoints	The number of points to use in the interpolated data
Power	The degree of polynomial to use for the polynomial fit

Create a 1x3 plot with three subplots, where each subplot plots the original data as well as an interpolated fit of the data. The first subplot should use a linear interpolation method, the second subplot should use a spline interpolation method, and the third subplot should use a polynomial fit for the data. Plot the data within the specified minimum and maximum x data points with the given number of points to use in the interpolated data. For the polynomial fit, use the power given in the input structure.

The function should then calculate an estimated R^2 value for the three different fits to determine their accuracy in conforming to the data.

This can be calculated with the following:

$$R^2 = \frac{SS_{reg}/n_{reg}}{SS_{tot}/n_{tot}}$$

where

$$SS_{reg} = \sum_i (f_i - \bar{y})^2$$

$$SS_{tot} = \sum_i (y_i - \bar{y})^2$$

and f_i are the interpolated data points, $\bar{y}$ is the mean of the original data points, and y_i are the original y values.

The original data should be plotted in blue stars on each subplot, and the interpolation method data should be plotted in green plus signs. The title of each subplot should follow the format: `'<interpolation method>: R^2 = <R^2 value>'`, where the linear method is referred to as `'Interp1'`, the spline method is referred to as `'Spline'`, and the polynomial method is referred to as `'Polyfit/Polyval'`. The R^2 value should be given to four decimal places, the x axis should be labeled `'x-axis'` and the y axis should be labeled `'y-axis'`. Finally, you should output the three R^2 values as a 1x3 vector, with each number rounded to the fourth decimal, in the same order as the subplots.

Notes:

- The equation given for the R^2 value is an estimate that only holds for certain assumptions. Do not try to use built in MATLAB regression functions for this problem.
- The output should be a double vector; do not try to format it as a string.

Hints:

- You may find the `linspace()` function useful.
- If you write the carrot, `'^'`, in the `title()` function call, MATLAB will write the character following it as a superscript.
- You can use the format `'%0.4f'` inside of `sprint()` to print a number to 4 decimal places.

Drill Problem #6

Function Name: winterWonderland

Inputs:

1. (*double*) The length of each "branch" of the snowflake
2. (*double*) A positive integer of the number of branches that the snowflake should have
3. (*char*) A 1x2 string describing the type of snowflake that should be made

Outputs:

(*none*)

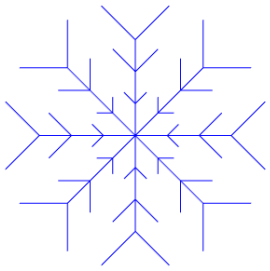
Plots:

1. A plot displaying the completed snowflake

Function Description:

Ahh… Fall has taken over the beautiful Georgia Tech campus, and the approaching winter makes you giddy. Just picturing Tech Green blanketed in a soft layer of fresh snow fills you with glee. But behind all this excitement lies an ominous truth: this is Atlanta, and there will likely be nothing more than cold rain this winter. Heartbroken, you decide to apply your MATLAB skills to engineer your own solution. You'll make your own snow!

Since no two snowflakes are identical, your code must account for three different variables: the size of each branch, the number of branches, and the type of snowflake. For example, if you were to run the code `winterWonderland(30,8,'pe')`, it would create this snowflake:



Notice that this snowflake is made up of branches and sub-branches, which sprout off in pairs along the branches. In total, there are 8 main branches and 3 sets of sub-branches off of each main branch. The first input of `30` indicates the length of the snowflake's branches, not including its sub-branches. The second input gives the total number branches, which start at the origin and radiate out evenly.

The third input is a string guaranteed to contain two letters. Each letter corresponds to a certain quality of the snowflake, and together they make up the type of snowflake that your code should output. The first character in the string describes how the sub-branches should be formed off the branches, and can be one of four letters:

Drill Problem #6

First Letter	Stands for	Angle	Length
‘n’	no sub-branches	0	0
‘p’	plain	360 / (number of branches)	(distance from origin) / 2
‘s’	star	360 / (number of branches)	(distance from origin)
‘r’	rose	720 / (number of branches)	(distance from origin) / 2

For this example snowflake, the first letter of the third input is `'p'`. Given this, the table indicates that the angle that each sub-branch makes with its branch is `360 / 8 = 45` degrees. Further, it is given that each sub-branch has a length dependent on its distance from the origin, measured from the point where it’s connected to its branch. Given the length of the branch as a whole is `30`, the sub-branches in the outermost ring are a distance of `30` from the origin. Therefore, their lengths must be `30 / 2 = 15`.

The second letter indicates the distance that each subsequent set of sub-branches should be from the origin. The next ring’s distance is dependent on the distance of the ring directly outside of it. This relationship can be one of three possibilities:

Second Letter	Stands for	New Distance
‘h’	heavy on the outside	(distance) – 200 / (distance)
‘e’	even throughout	(distance) – 10
‘t’	tight on the inside	(distance) ^ 0.85

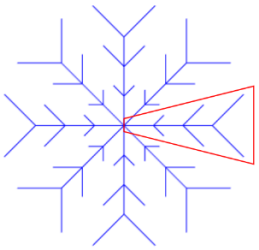
Looking again at the example snowflake, the second letter of its third input is `'e'`, meaning that the sub-branches should be evenly spaced from one another along their branches. Thus, because the outermost ring of sub-branches is a distance of `30`, the distance of the ring immediately inside of it is `30 - 10 = 20`, and the distance of the third ring, inside the second, is `20 - 10 = 10`.

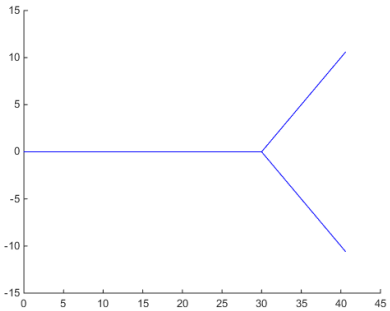
The smallest ring of sub-branches must be at least `5` away from the origin.

Walkthrough:

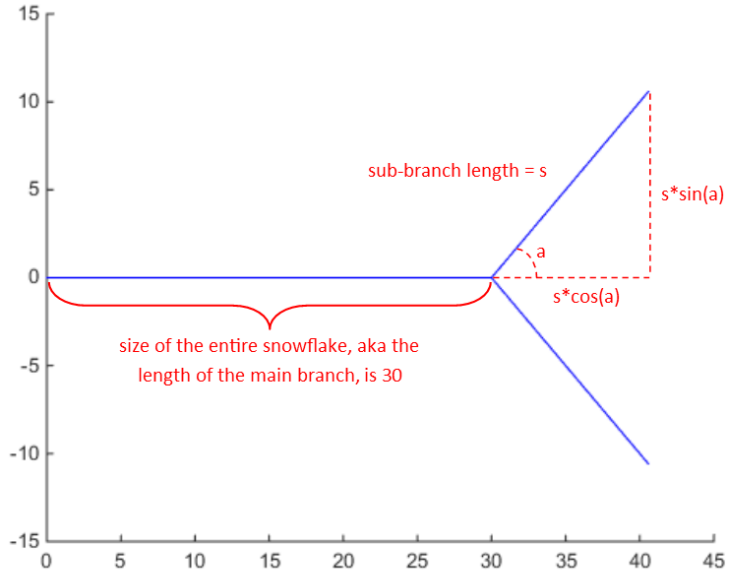
When writing your code, your first task should be figuring out how to create the sub-branches of a single branch.

Looking at only a single branch and its outermost sub-branches:



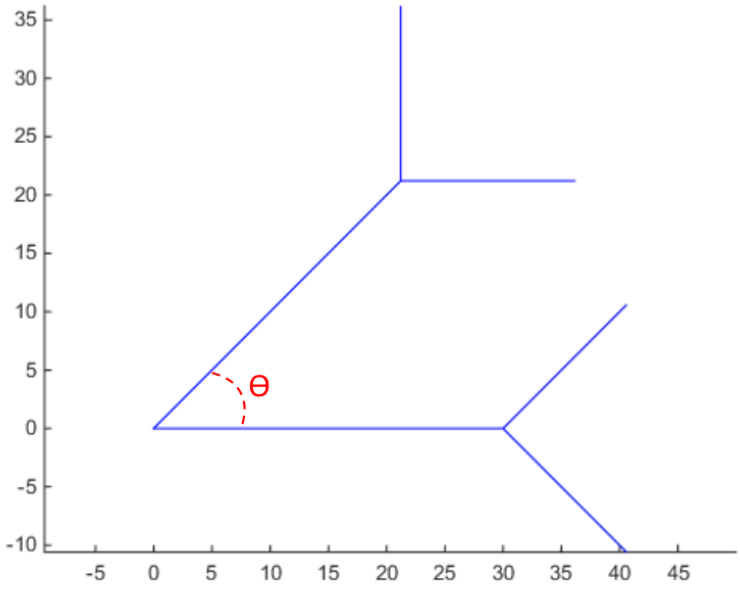


Drill Problem #6



Once you create this single branch, you want to create all the other branches around the snowflake. To rotate a set of points counter-clockwise around the origin, multiply the coordinates by the rotation matrix as follows:

$$\begin{bmatrix} x_{rotated} \\ y_{rotated} \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \times \begin{bmatrix} x_{original} \\ y_{original} \end{bmatrix}$$



Once you have made all of your main branches, you must recursively create all of the sub-branches.

This explanation only details one way to solve this problem, but there are other methods. While there are other ways to solve this problem, they may result in your plot being slightly off from the

solution's plot. The autograder has built-in plot tolerance, but we still cannot guarantee you will get full credit it your plot ends up being off from the solution plot due to method variation. If you choose to solve this problem a different way, the function does have to be recursive!

Notes:

- You **must** use recursion to solve this problem!
- Your snowflake **must** be blue.
- The final result must be square (even-sized axes), and the axes must **not** be visible.
- You are guaranteed that the snowflake's size will be at least 5.
- You are guaranteed that the number of branches will be at least 1.
- The functions `sind()` and `cosd()` are the sin and cos functions with inputs in degrees.

Hints:

- Try first creating the biggest outer branches, and then using recursion to make the smaller branches within. Another way of thinking about this is making the outermost ring of sub-branches, and then working your way in.
- Think about this problem as first making a simple snowflake with only its outermost ring of sub-branches. You can then recreate this simple snowflake in smaller and smaller proportions, each overlapping the original snowflake.