are met. A real-world comparison can be made with shopping. When you go shopping you only will purchase an item if it below a certain price. In other words, IF the item is below a price, you will buy it. Otherwise, you will not buy it and proceed to get something else. This example demonstrates the general format of an If statement.

```
If condition is true

    first result occurs

Else second result occurs

End
```

When applied to the shopping example:

```
    If orange juice is below a specified price

    Purchase orange juice

Else purchase milk.

End
```

In this example, if the initial condition is true, then the first result will occur. When the initial condition is false, then the code will skip over instructions nested under the “if” statement and will execute instructions nested under the “else” statement. The “end” denotes the end of the if statement. Every “if” statement you write must have an “end” command.

When more than two options are required for a code the “if” and “else” structure can be modified to also include “elseif” options. “elseif” functions as an additional option that can be executed after the “if” and before the “else”, which is always the last option. The user can have as many “elseif” elements within an if statement as required. An example expanding on the grocery example using multiple “elseif” elements is shown below.

```
If orange juice costs less than 1 dollar

    Purchase 3 cartons of orange juice
```

```
Elseif Orange juice costs between 1 dollar and 1.50 dollars

    Purchase 2 carton of orange juice

Elseif orange juice costs between 1.50 and 2 dollars

    Purchase one carton of orange juice

Else purchase milk.

End
```

Switch Statements

for Loops

When coding in MATLAB, you will encounter situations where you want to execute a command when specific conditions are true. For example, if a code is to repeat an operation a, it would use the conditional statement known as a for loop. The structure of a for loop is provided below. This example runs the code included under the for line repeatedly for x = 1 through x = n. After the code is ran for n iterations the code goes after the end line and runs the remainder of the code.

```
    for x = 1:n

    Task here. This could be a mathematical operation, for example.

end
```

The following is an example of a for loop that is being used to sum all prime numbers between 1 and 1000. This code includes the use of the “isprime” function, which checks if the input (in this case k is prime). The for loop cycles every number between 1 and 999 through the isprime function, which will proceed to add together only the numbers that are prime. The isprime function will not be included in the scope of ME 160, but I am including it as a supplementary piece of information which may benefit you as you write codes.

Notice how the code starts with a variable called “total” which initially is assigned the value of zero. As the for loop runs over and over the value for total is over written by the value of total in the previous iteration summed with any new prime number that is found in the current iteration.

Another example of a for loop is included below. This code uses a series of integers that are stored in an array named m. The dimension of the matrix is determined by using the “size” command. This creates another array that lists the number of rows and columns, respectively in a matrix or array. The second value in this size array is pulled out, informing us of the number of columns (and quantity of numbers) present. The objective of the code is to sort through the array and decrease all odd numbers by one so that every value ends up being positive. Two images are included, one with just the MATLAB script and another with comments added walking through the code.

Include section on nesting for loops and show pseudo code and real code examples of this. Show 3d plot using this.

The next example shows two for loops that are nested within each other. This means that for every time the outer for loop runs once, the inner loop runs its entirety of iterations. This example is used to populate values within a 2-dimensional matrix.

While Loops

As you learn more commands in the MATLAB language, you will find that loops are a powerful tool to ensure your code runs under the correct parameters. In addition to for loops, MATLAB contains the while loop, which operates similarly to the for loop. When coding with the while loop, the user defines a parameter that must be true for code nested inside the while loop to run. Within ME 160, while loops will be used often to add a way to rerun or end your code. An example of a rerun option using a while loop is listed below.

While loops are frequently used in conjunction with a for loop as a way to control how long the for loop runs. For example, in optimization problems a for loop may be used to attempt to find a minimum value in a quadratic function using a line search method where values are checked until the slope of the function equals zero. A for loop can search every value within a desired range of the function. A while loop placed outside the for loop can be used so once the

minimum point with zero slope is found, a variable is changed which stops the while loop, and the for loop within, from running.

Examples

Create a code which can prompt the user to input the year they graduated from high school, calculate the year that they would receive a bachelor's degree (assuming a four-year program), and display that year for the user. Use input and fprintf functions to complete the script.

A company wants to know how much it costs to pay an employee each year. If the employee receives some raise each year they work, create a calculator which will determine the pay the employee receives for each of 5 years and in total. Allow the user to input the worker's initial pay and how many hours they initially work. Assume the worker works 40 hours per week for 52 weeks a year.

Examples in previous chapters have created codes which can calculate the hypotenuse of a right triangle using the Pythagorean Theorem. These codes had to be written to accommodate each triangle and was not practical. Create an improved Pythagorean Theorem calculator which:Informs the user what the code does using a display command.

Prompts the user to input the length of both short legs of a triangle using an input function.

Displays the length of the hypotenuse using a fprintf command.

The following equation converts between degrees Fahrenheit (°F) and degrees Celsius (°C):

$$℃=(℉-32)×5/9$$

Create a MATLAB code which converts temperatures in Fahrenheit to Celsius. Use a display function, input function, and fprintf function to create the code. Test the code with any temperature in degrees Fahrenheit.

Problems

When solving various engineering problems, it is useful to easily convert from various units.

Create a code which can convert seconds to hours and days. For example, 3600 seconds should equal 1 hour and 1/24th of a day. Use an input function and at least one fprintf function to complete this code.

Modify the script written in problem 1 such that the user is asked if they would like to run the code again. Either repeat the code or end the code with a loop depending on the user's selection.

Create a code which enables the user to determine the density of a specific volume of a material. Have at minimum the following components:

The ability to select between aluminum or 1020 steel, with respective densities of 2.7g/cm^3 and 7.87 g/cm^3.

Have the user input the volume of the object. Ensure they are informed of the units they are expected to use.

Objects fall at different rates on the Earth and the moon as a result of their gravitational attraction having different magnitudes. Create a code that calculates how much longer it takes an object to fall from an input height on the moon than the Earth. The gravitational acceleration of Earth is 9.8 m/s^2 and 1.62 m/s^2 on the moon. The following kinematic equation will prove useful to write the request.

time=√((2(height))/g)

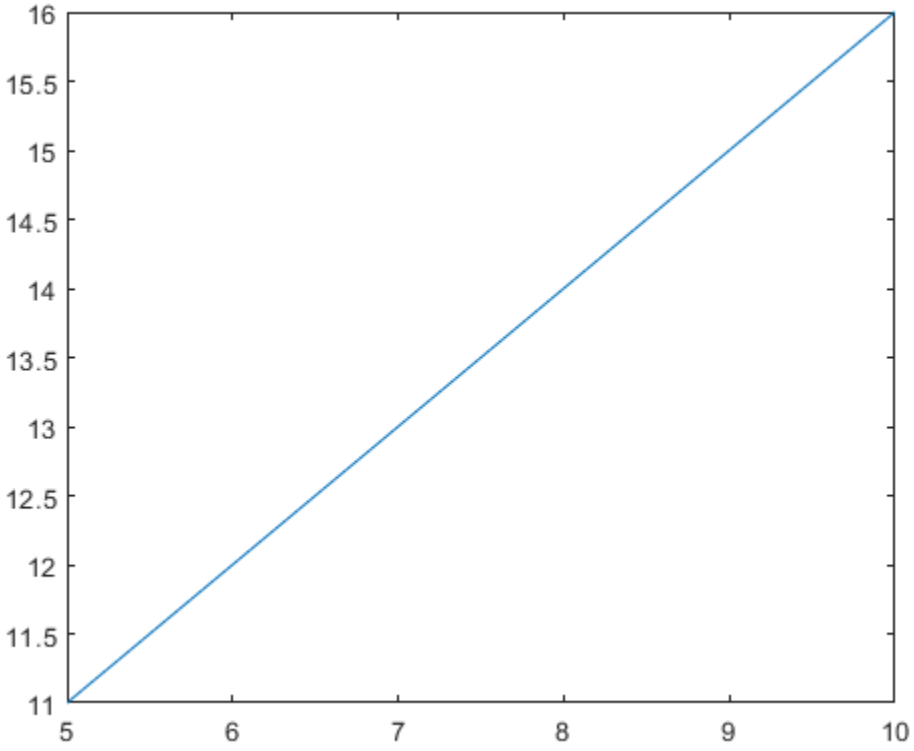
CHAPTER 6: GRAPHING IN MATLAB

Creating Graphs

MATLAB has tools that enable the user to display data within visual forms such as tables, 2D, or 3D graphs to increase readability for the user. General graphs can be created by the user with the plot command, which can be modified to incorporate colors, symbols, labels, and other aspects of the graph to ensure that the data is able to be read and interpreted by the user.

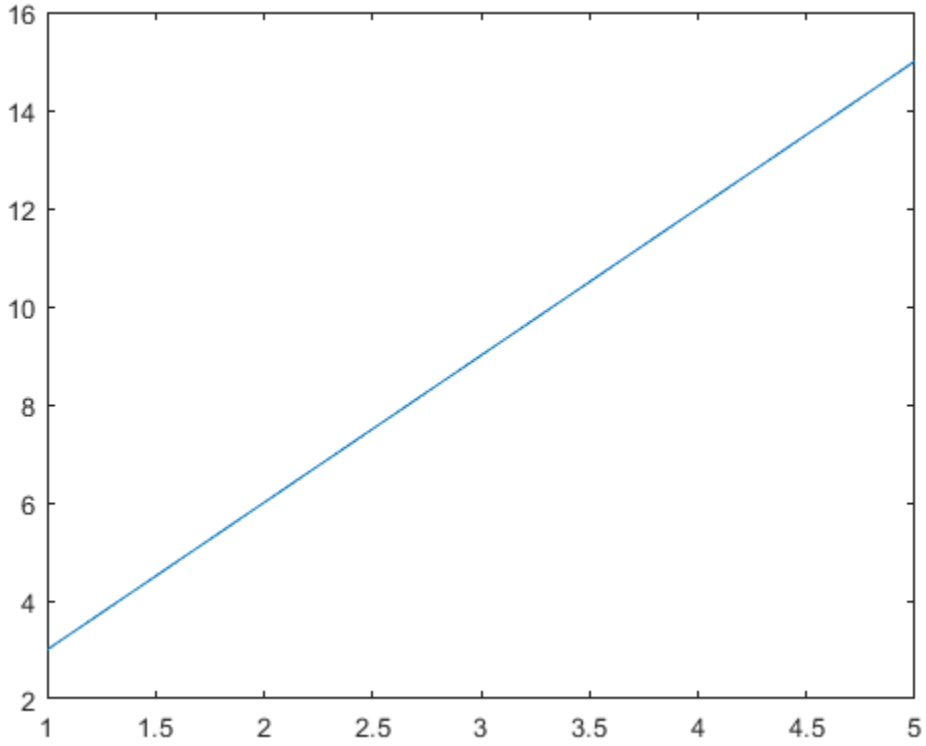
The plot function operates by plotting data assigned to a variable onto a graph. A simple way to graph the first-order line onto a plane is by listing the range of values for both the x and y coordinate which need to be graphed. The following example shows how the user could assign the range for the x- and y-axis, respectively, using vectors. This notation will generate a graph with a line running from the point (5,11) to (10,16). This is an easy way to generate a linear graph but most applications within scripts will be more involved than this.

```
>> x = 5:10;
>> y = 11:16;
>> plot (x,y)
```



The above technique can be modified to make one variable dependent on the other. For example, if you have a set range of values (let's say x = 1,2,3,4,5 for this example) and another variable that is a multiple of x (say y = 3*x for this example) the following code could be used to plot the above example.

```
>> x = 1:5;

>> y = 3*x;

>> plot(x,y)
```



Most applications of the plot function within MATLAB will incorporate an equation for one of the variables, thus creating the need for a plot to visualize the data. As a result, equations such as the y-value in the above example are much more prevalent than predetermined vector operations.

Another way to generate an evenly spaced vector to use when plotting a function is the `linspace` command. The `linspace` command selects the initial value, the final value, and how many values should be generated in between this range. An example is shown below, which represents a vector containing 100 values ranging between 0 and 10. The `linspace` command is used below in the example plotting two functions on the same graph.

```
>> linspace(0,10,100)
```

Several functions can be graphed within a single MATLAB, enabling the user to display several data sets more efficiently. This can be done by listing each set of variables in a series. For example, for the following variables, x, y, and x, the user would graph an x vs y plot and a x vs

z plot in the same axis by using the following notation. This notation is one way the user can plot as many sets of variables as needed, though it is limited to 2D.

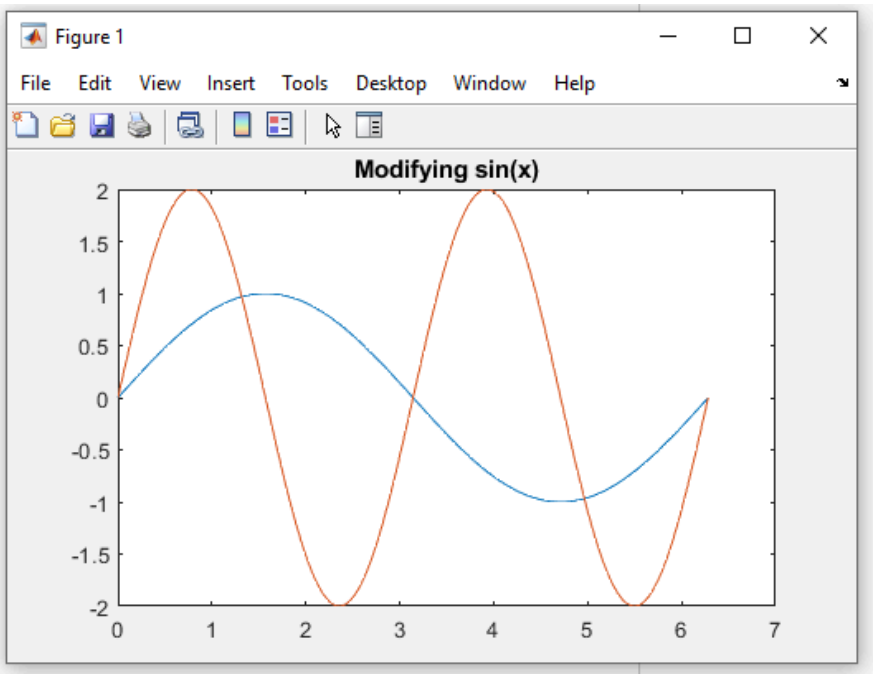
```
>> x = 1:6;

>> y = x.^2;

>> z = x.^3;

>> plot(x,y,x,z)
```

In order to place labels and titles in MATLAB plots, the following commands can be used to generate labels. Listing each of these commands after a plot command will add the labels to the graph. The following example is a general example of how to label the graph. Note that the font size can be modified by adding 'FontSize' followed by the desired font size after the title in the command.

```
>> xlabel('Name of Label')

>> ylabel('Name of Label')

>> title('Graph Title','FontSize', 12)

>> legend('Name of First Plot','Name of Second Plot')
```

The following image contains concepts addressed above within an actual MATLAB code to create graphs like the one represented in the above examples of commands.

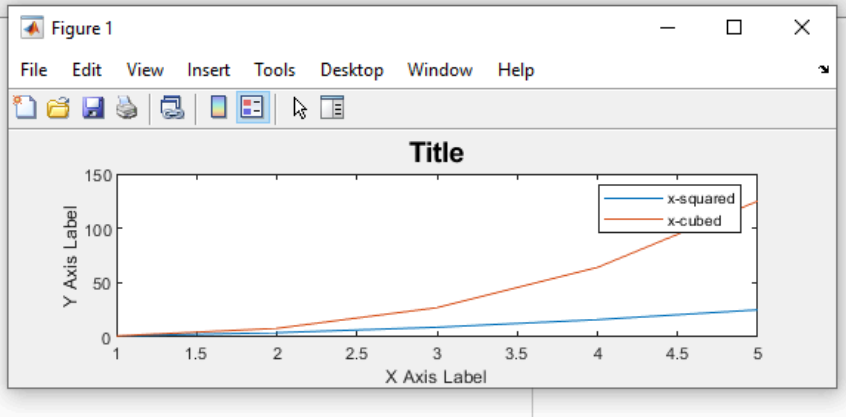
```
clear all
close all
clc

x = linspace(0,2*pi,200);
a = sin(x);
b = 2*sin(2*x);

plot(x,a)
title('Modifying sin(x)')
hold on
plot(x,b)
```



Another easy method to plot several lines on the same graph is easy within MATLAB. When writing code, use the hold-on command in between commands plotting each line. This will result in several lines being graphed on the same plot, which can be valuable when comparing data or wanting to save space when displaying information. An example is included below which also uses the `linspace` function to generate vectors for the x-axis of the plot.

SampleCode.m

```
close all
clc

x = 1:5;
y = x.^2;
z = x.^3;
plot(x,y,x,z)

legend('x-squared','x-cubed')

xlabel('X Axis Label')
ylabel('Y Axis Label')
title('Title','FontSize',14)
```



There are many methods available within MATLAB that can assist in producing graphs including scatter plots, line plots, or other non-linear display methods. To create a function that plots a scatter graph instead of a linear graph, use the function scatter in the place of the

plot, which will only place points. This type of function is when the vector used to create the x-axis is important, as the number of points in the vector will dictate the number of points in the scatter plot. Additional replacements that can be used in place of the plot function include the bar function for bar graphs, the stairstep function for stairstep graphs, and the stem plot for stem graphs. An example of the scatter function is included in example 1 below.

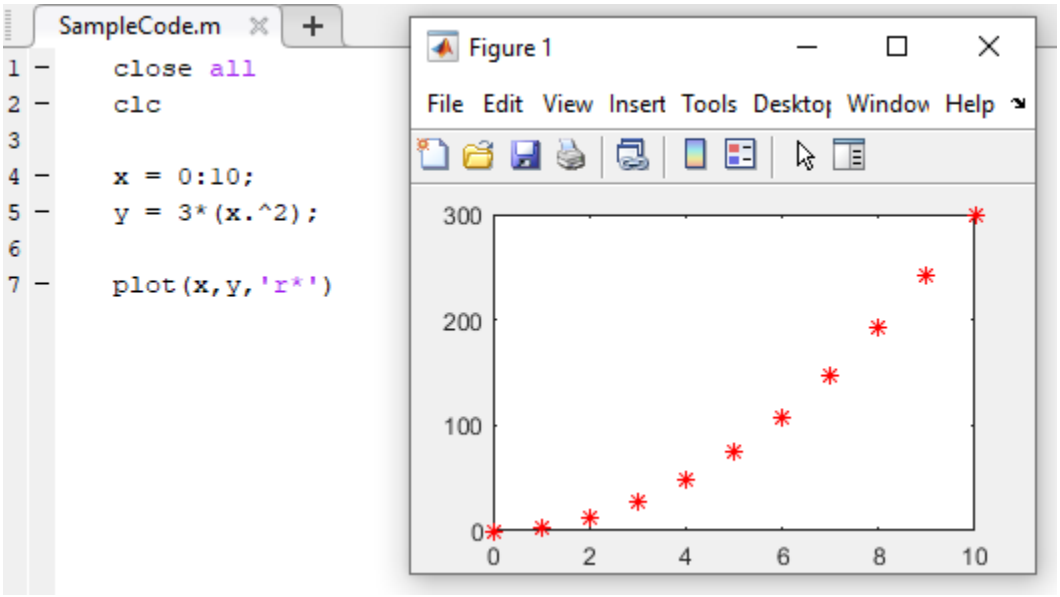
Specifier	**Line Style**
'-'	Solid Line, which is the default
'_'	Dashed Line
':'	Dotted Line
'-.'	Dash-Dot Line
Specifier	**Marker Type**
'+'	Plus
'o'	Circle
'*'	Asterisk
'.'	Point
'square' or 's'	Square
'diamond' or 'd'	Diamond
'^'	Upward-pointing triangle
'v'	Downward-pointing Triangle
'>'	Right-Pointing Triangle
'<'	Left-pointing Triangle
'pentagram' or 'p'	Five-pointed Star (Pentagram)
'hexagram' or 'h'	Six-pointed Star (Hexagram)

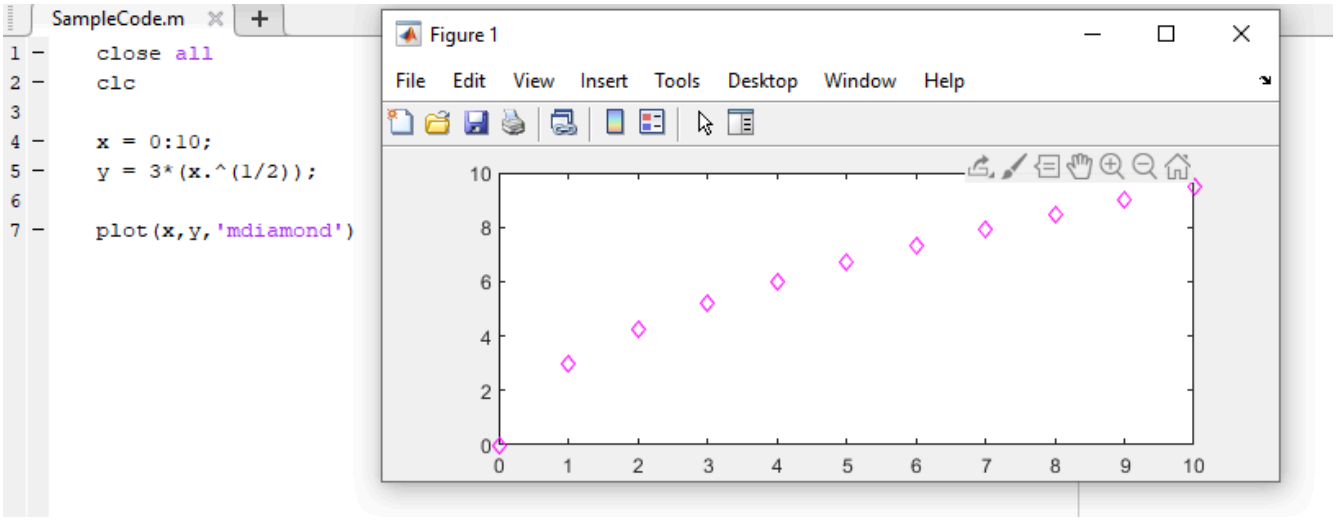
MATLAB plots enable the user to modify the physical appearance of the code to incorporate colors and shapes which will help depict data more effectively than a simple line. Being able to differentiate the physical appearance of graphs will be especially important when plotting several data sets in the same plan, which will be addressed shortly. Color or shape variations are listed at the end of a plot function within quotations. The following chart depicts a list

of symbols and letters that can be used to create corresponding symbols or colors within a MATLAB plot.

The following examples utilize colors and specifiers to create unique and more effective graphs. Notice that when using a vector, the specific values are visible when using a point notation as opposed to a continuous line. An additional detail that is necessary to make this code run is the period placed after the variable x in the equation. In order to multiply a matrix by an exponent, the period must be used to tell MATLAB to take each value in the matrix by the power. In this situation, each number from zero through 10 is squared and plotted in the graph.

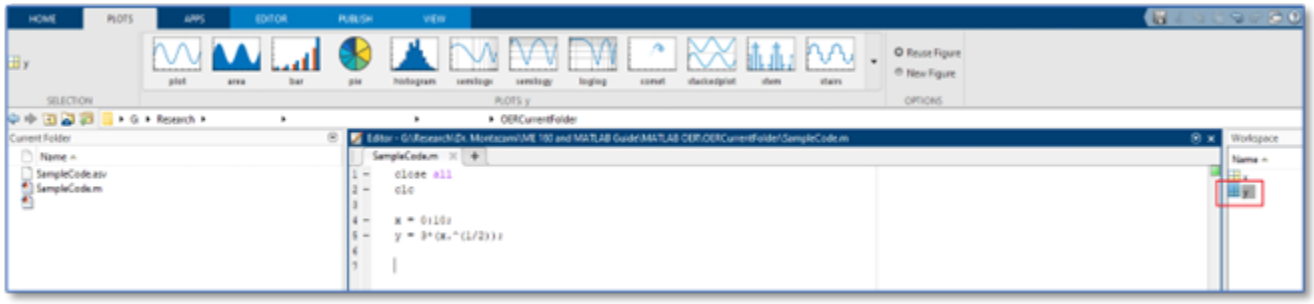
Symbol	**Color**
r	Red
g	Green
b	Blue
c	Cyan
m	Magenta
y	Yellow
k	Black
w	White





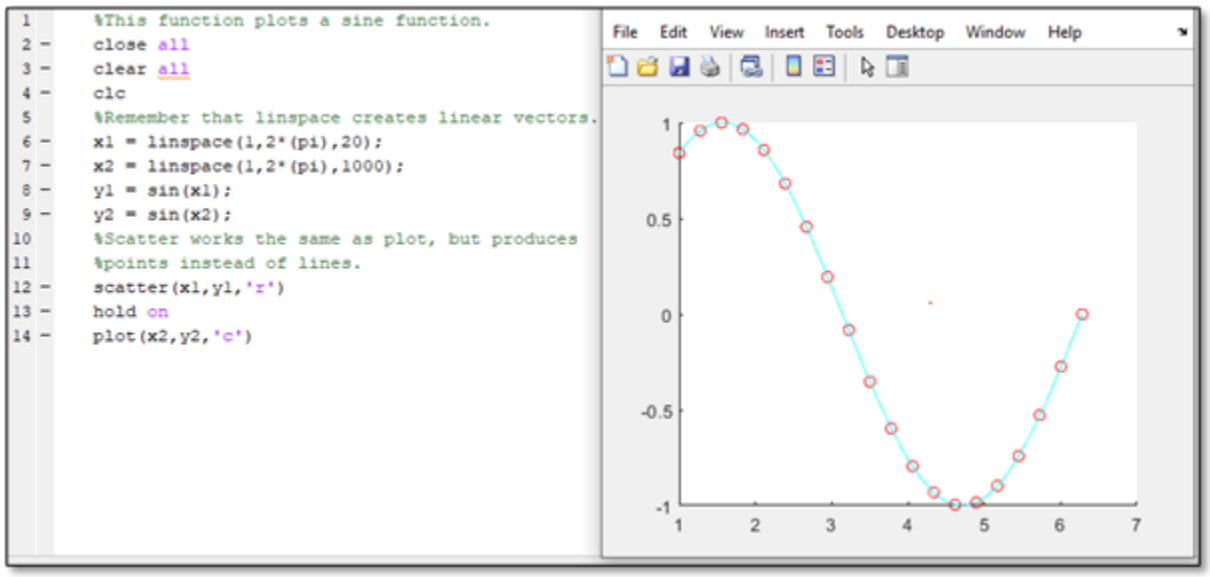
The Plot Tab

The plot tab in the MATLAB user interface is a handy tool to create involved plots for variables within a MATLAB code. By using the plot tab user can select a variable in the workspace and generate a graph that best represents the data assigned to the variable. In the following example, the variable y is selected, which enables the user to select one of the included graphs from MATLAB's library. This feature can save time and enable the user to generate a quality image that effectively displays the user's data.



Examples

1) A professor wants a graph that depicts 20 points of a sine wave undergoing one period of motion. Create the graph using a cyan sine wave and 20 red stars plotted over the cyan wave. Label the x-axis “X”, the y-axis “Y”, and title the graph “Sine Wave”.

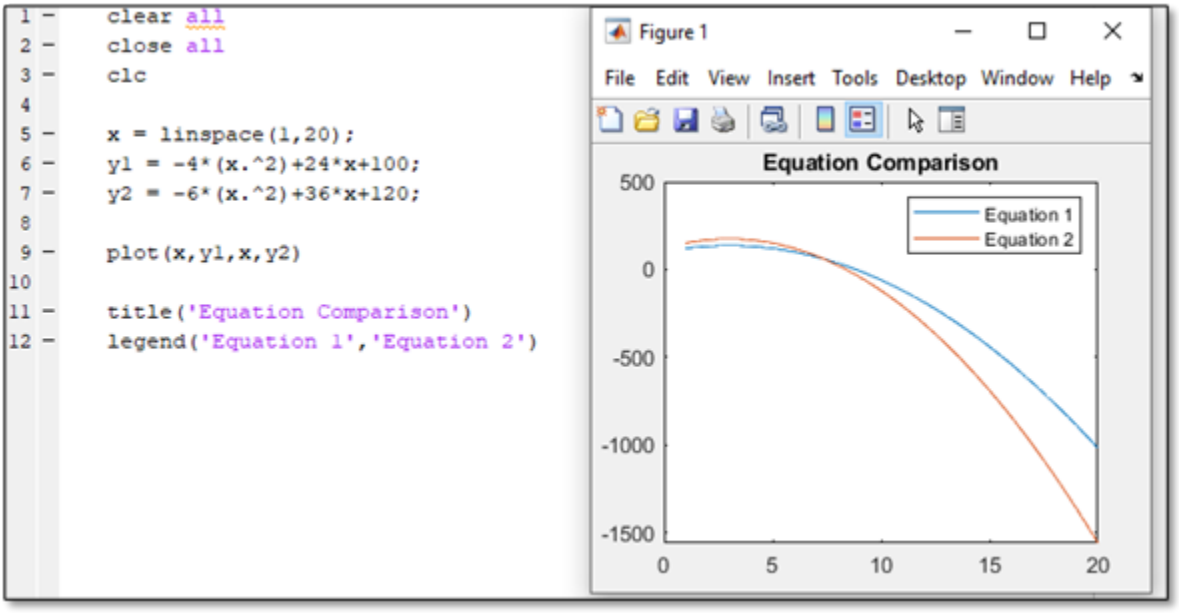


```
%This function plots a sine function.
close all
clear all
clc
%Remember that linspace creates linear vectors.
x1 = linspace(1,2*(pi),20);
x2 = linspace(1,2*(pi),1000);
y1 = sin(x1);
y2 = sin(x2);
%Scatter works the same as plot, but produces
%points instead of lines.
scatter(x1,y1,'r')
hold on
plot(x2,y2,'c')
```

2) The following functions represent the motion of two projectiles. Plot the functions on the same graph. Use proper labels and ensure the graph is sufficiently legible for users. Use x-axis values from 0 to 20. Create a key and some title for the plot.

$$\text{Equation 1: } f(x) = -4x^2 + 24x + 100$$

$$\text{Equation 2: } f(x) = -6x^2 + 36x + 120$$



Problems

1) The following data points were collected in an experiment. Write a MATLAB script that plots the data points.

Time (s)	1	2	3	4	5
Location (y)	1	4	7	12	17

2) The location of a particle is represented by the function. Write a MATLAB script that enables the user to plot the data set over a range of seconds inputted by the user.

3) The following equation can be used to model the velocity of water as it flows out of a hole located in the bottom of a tank. Given that (velocity coefficient) is .97 for water, gravitational acceleration, is 9.81 on Earth, and is the height of the water, determine the velocity of the water like a tank initially filled with 5m of water drains until empty. Create labels for the graph's axes and a title.

$$v = \ c_v(2gh)^{\frac{1}{2}}$$

CHAPTER 7: GRAPHICAL USER INTERFACE

Introduction

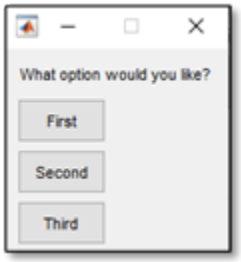
Graphical User Interfaces, or GUIs, are tools that improve how the user can interact with a code by modifying the appearances of inputs, messages, or other notices. As a result, users can type inputs or interact with codes through pop-up windows instead of using the command window. The following section will discuss various GUI commands, such as menu, input dialog, and message box.

GUI is a powerful way to improve the usability of a code. By using a GUI, it is easier for users to input and read data and removes the need for users to interact directly with the command window. This capability makes it easier for users to input data and much easier to write codes which allow users to select from several options.

The list dialog Command (listdlg)

The `listdlg` command enables the user to select from several predetermined options from the user to select from several predetermined choices for inputs. This command replaces the command "`menu`", which is no longer recommended for creating lists to select from in MATLAB. The menu function may still be present in codes more than a few years old and is useful to understand. If using `menu`, the following format creates a prompt accompanied by buttons with each listed option within the command. An example of this with an accompanying output is shown below.

```
>> a = menu('What option would you like?','First','Second','Third');
```



The `listdlg` syntax is similar to the older `menu` command. An example of the `listdlg` is shown below that outputs the same menu as the above example. First, a variable is created which is assigned an array of each list entry that will be in the menu. After the list is defined, the format of the list and the name of the list variable are entered into the `listdlg` function.

```
list = {'First','Second','Third'};
a = listdlg('ListString',list);
```

Adding options to the basic "ListString" selection enables further customization of the command. The option "PromptString" allows an instruction to be printed on the output as well. Additionally, the "SelectionMode" function allows for customization of the number of options that can be selected in the list. These features are shown added to the ongoing example below.

```
list = {'First','Second','Third'};

a = listdlg('PromptString',{'Select an Option:'},
'SelectionMode','single','ListString',list);
```

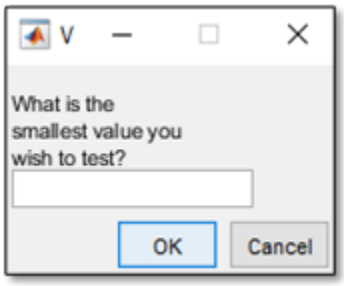
This code would generate the following textbox within MATLAB. Once the user selects one of the options, the variable a will be renamed to correspond with the selected option. Note that the variable will be named based on the order of the selected option, and not based on the text placed in each box. For this example, if the user selects the first box labeled “First”, the variable a will be assigned the value a=1. This enables the author of the code to place any text within the box and not have it affect what the variable is named.

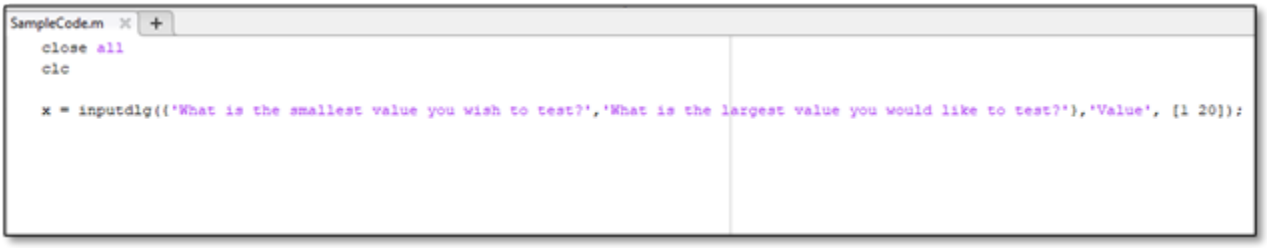
The Input Dialog (`inputdlg`) Command

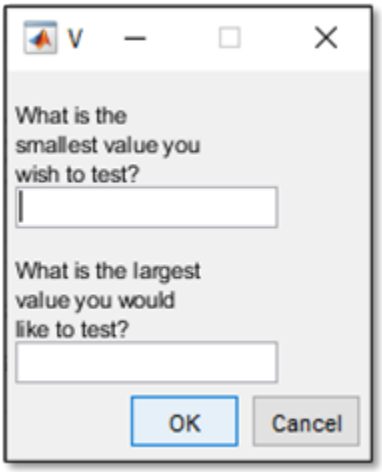
The input dialog function is like the menu command in the fact that it enables the user to interact with their codes and can interact with their codes through a pop-up window. Instead of selecting from a button as in the menu command, the input dialog command enables the user to input data or values into their code in a textbox. The following example shows how to format the input dialog command. The format for this command is slightly more complex than

the menu command due to its increased features. The first portion of the code contained in brackets and quotations and highlighted in yellow is the message which will be displayed in the textbox. The second portion highlighted in blue shows the message which can be displayed at the top of the textbox. The green portion shows how many characters as tall and long the textbox will be. In most applications, a relatively small box with dimensions around 1×20 characters is enough. The second example of the `inputdlg` function depicts how to input two values into the function. Note that each message is separated by a comma.

```
x = inputdlg ({'What is the smallest value you wish to test?'},'value',
[1 20]);
```



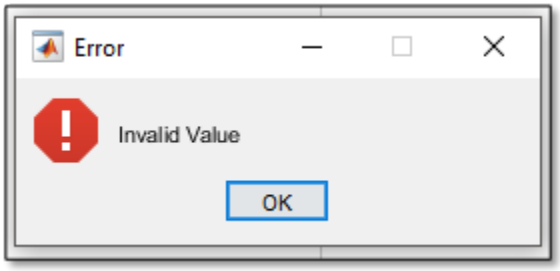




The Message Box (`msgbox`) Command

The `msgbox` function creates a message box that can display a message in a pop-up or can modify the `fprintf` function to display a message in a pop-up. An example of a `msgbox` function that displays a message and an image is shown below. Visit the Mathworks `msgbox` webpage to view a more complete listing of images available for the `msgbox` function. Within the `msgbox` function, the text highlighted in yellow is the main message in the box. The text highlighted in green in the second set of parentheses is the title which appears next to the MATLAB logo in the top header of the pop-up. The third piece of data highlighted in blue is the code for the image which appeared in the `msgbox`.

```
f = msgbox('Invalid Value', 'Error','error');
```



Examples

The following code is a unit converter developed by an Iowa State faculty member. This code can be modified to include a `msgbox` function to increase the code's usability, as shown in the second version. This example also demonstrates various elements of MATLAB addressed thus far, such as while loops, if statements, and nested loops.

```
clc; clear
d=1;
while d==1
    c=inputdlg ({'Please enter the length to be
converted'}, 'Length', [1 50]);
    b = str2num (c{1});

    a=menu('are you converting from meters or
feet?','meters','feet');
    if a==1
        x=b*3.28;
        fprintf('%f meters is equal to %f
feet\n',b,x)
    elseif a==2
        x=b/3.28;
        fprintf('%f feet is equal to %f
meters\n',b,x)
    end
    e=1;
    while e==1
        c=menu('do you want to try again?
','yes','no');
        if c==1
            d=1;e=0;
        elseif c==2
            d=0;e=0;
        end
    end
end
```

```
clc; clear; close all;
d=1;
while d==1
    c=inputdlg ({'Please enter the length to be
converted'}, 'Length', [1 50]);
    b = str2num (c{1});

    a= menu('are you converting from meters or
feet?','meters','feet');

    if a==1
        x=b*3.28;
        msgbox (sprintf ('%.2f meters is equal to
%.2f feet\n',b, x), 'Answer','help')
    elseif a==2
        x=b/3.28;
      msgbox (sprintf ('%.2f feet is equal to %.2f
meters \n',b, x), 'Answer','help')
    end
    e=1;
    while e==1
        c=menu('do you want to try again?
','yes','no');
        if c==1
            d=1;e=0;
        elseif c==2
            d=0;e=0;
        end
    end
end
```

Resources

- Mathworks listdlg command documentation
- Mathworks msgbox command documentation

CHAPTER 8: FUNCTIONS AND FUNCTION HANDLES

What is a Function?

At this point, students have learned many commands within ME 160 and have the tools to complete operations through more and more complex codes. As codes become longer and more complicated it is necessary to find ways to reuse common sections which are used in multiple scripts efficiently. MATLAB has a file type called functions which enables the user to create their own functions in separate script files and refer to them in another script they are writing. This is handy when using the same equations several times in different codes and would like to save time and not repeat the same code over and over.

How to Create a Function in MATLAB

A function can be created in a similar method to normal scripts. By having the first line of a function contain the function command, the script will be saved as a function file and not as a .m MATLAB script. The following shows a very simple function that can compute the factorial of some number n by summing every integer between one and n. By having "function" being the first thing in the script, MATLAB can automatically determine that this is not a full script, but instead just a function that may be called to a different script.

```
function f = fact(n)

    f = prod(1:n);

end
```

In this case, the function is called "`fact`". When saving the newly created file, it must be called "`fact`" in order for the function to work properly. After saving this function as its own .m file, the user can call the function like any other MATLAB command in a script that is being

written. Remember to save the function file with the same name as the function, so the .m file should appear as "fact.m" in this example. This is a very easy mistake to make that will prevent MATLAB from finding the correct function. Another word of warning, remember to rename the function file if you rename the function while editing the code during revisions! Calling the function in a script is shown in the following example. Within this script, a variable "x" is assigned a value. Then, MATLAB will assign the variable "`y`" a value after completing the operation within the "fact" function. Notice this operation takes the value assigned to "x" as input and enters that into the function for the variable "`n`". This demonstrates that variable names do not need to be consistent between the function and the script, as the function will accept any variable entered in the correct location. This enables a function to be used several times for several different values throughout a script, demonstrating the simplifying benefits of functions.

```
x = 5;

y = fact(x);

y =

    120
```

Function Example: Quadratic Equation

An example of an application for creating a function in a script is included below. This example contains a script that would like to use the quadratic equation. Instead of utilizing equations within the base script, a separate quadratic equation function is created and used for this application. The function, entitled "quadraticsolver" begins with a line defining that the script is a function that outputs a variable "x" when given inputs for variables "a", "b", and "c". Within the main script "QuadraticFunction", a user inputs values for variables "a", "b", and "c". These variables are plugged into the function "quadraticsolver" in line 9. This completes the arithmetic included in the "quadraticsolver" function file and outputs the results to x. Notice how in this example the variables "a", "b", "c", and "x" are used in both the function and main script. When using the function, different variables could have been used. For example "`x = quadraticsolver(i,j,k)`" will work the same as the code cares only about the order

the function is fed information, not the actual variable names. This means that the naming conventions of a script do not need to be changed to match a function's naming.

quadraticsolver.m

```
1
2   function  x = quadraticsolver(a,b,c)
3   x1 = (-b + sqrt((b^2)-(4*a*c))/2*a);
4   x2 = (-b - sqrt((b^2)-(4*a*c))/2*a);
5   x = [x1 x2];
6   end
7
```

quadraticsolver.m | QuadraticFunction.m

```
1   clear all
2   close all
3   clc
4
5   a = input('Input the A value from quadratic equation:\n');
6   b = input('Input the B value from quadratic equation:\n');
7   c = input('Input the C value from quadratic equation:\n');
8
9   x = quadraticsolver(a,b,c);
10
11  disp(x)
12
```

Command Window

New to MATLAB? See resources for Getting Started.

```
Input the A value from quadratic equation:
1
Input the B value from quadratic equation:
5
Input the C value from quadratic equation:
6
   -4.5000   -5.5000

fx >>
```

Resources on Functions

- Mathworks help center guide to functions
- Mathworks YouTube video introducing functions

CHAPTER 9: INPUTTING AND OUTPUTTING DATA

Up to this point MATLAB has been used to analyze singular user inputs and has only addressed scripts that output a finite number of outputs. To get the most of MATLAB and be able to scale its operation, the user must have the ability to input large data sets and output equally large data sets automatically. This chapter will address how to create codes using excel or comma-separated values (.csv) files for data input and/or output.

Introducing Comma Separated Values Read and Write Functions.

MATLAB has made it easy for the user to place generated sets of data into a separate file or word processing software which can be stored and processed outside of the MATLAB software. A comma-separated values file is a plain text file that contains a list of values that are easily input into text editors such as Microsoft Excel or Apple Sheets. Data that is stored as a variable in MATLAB can easily be exported to a .csv file using the `csvwrite` command. The notation for the command is included below.

```
>> csvwrite ('filename.ext', X)
```

The `csvwrite` command only requires a couple of pieces of information to generate a .csv file. The portion in green is the desired file name for the .csv file. Note that the name ends with the extension .csv, which is required to generate the correct file format. The portion of the command in blue is the name of the variable storing the data, which is presumably either a matrix or a vector. It is the information contained within variable X that will be output into the file.

While the `csvwrite` function is easy to use and is effective at generating new .csv files, the command is not helpful if the user would like to input data directly into an existing .csv file. To input data into an existing file, use the `writematrix` command in place of the `csvwrite`

command. The formatting for the `writematrix` command is below. Some references will recommend using the `dlmwrite` function in place of `writematrix`, though MATLAB is phasing out dlmwrite and it is currently recommended to use the writematrix function.

```
>> writematrix(M,'filename.ext')
```

Within examples of file names in this chapter, notice how each command lists the extension as ".ext", which is shorthand for "extension" and serves as a general placeholder for any extension. Within examples above, such as csvwrite, it is assumed that the user is outputting data into a comma separated value file. However, the writematrix command can accommodate many varieties of file formats. The following chart lists possible extensions which could be used to specify the file format desired by the user.

Per MathWorks documentation, the following file formats can be used to incorporate text or spreadsheet files.

There are a couple methods that can be used to input data from a .csv file into a MATLAB code. The function csvread operates in a similar manner to the csvwrite function, while the csvread function reads data from a preexisting .csv file and inputs the data into a MATLAB variable. The following shows the options the user has for inputting .csv data into MATLAB.

Option 1:

>> csvread ('filename.ext')

Option 2:

>> N = 'filename.ext'

>> M = csvread (N)

The first option inputs the data from the csv file directly into the MATLAB program. The second option ensures that the data input from the .csv will be stored under variable M. This can easily be modified to read only parts of a .csv file by adding the row and column on the file where MATLAB should input data. This is depicted below where MATLAB will input data starting at the third row and second column of the file.

>> N = 'filename.ext'

```
>> M = csvread (N,3,2)
```

CHAPTER 10: PROJECTS

The projects section of this text is intended to provide examples that are summative and encompass many topics covered within this text. Each exercise is currently provided in several versions with different levels of prerequisite knowledge. These exercises may be used as homework assignments, as a basis for more complex assigned projects, or as ways for independent students to verify that they are able to apply individual skills addressed previously to solve a complex and, hopefully, applicable process.

Contents in Projects Section:

- Elastic modulus calculation (in three versions)
- Unit conversion calculations (in two versions)
- Material density scenarios (one version)

Project 1: Elastic Modulus Calculation

The elastic modulus of a material measures a material's resistance to being deformed elastically. Useful in materials analysis, the elastic modulus is frequently calculated from data collected from mechanical testing of materials. The following projects have the user generate forms of elastic modulus calculators which input a variety of data types depending on the complexity of the version.

Project 1 Version 1: Elastic Modulus Calculator

Concepts: Use scripts with inputs to determine the elastic modulus, stress, and strain of material.

Problem: A team of engineers is conducting tensile tests on plastics that will be used in a car part and need to determine the materials elastic modulus, or E. Elastic modulus is a ratio of the

stress exerted on a material (σ) over the material's strain (), which can be further expressed by the following equation, where F is the force applied to the material, A is the material's cross-section, L is the materials initial length, and ΔL is the change in the length of the material after the test (or the final length – the initial length).

$$E = \sigma/\epsilon = (F/A)/(\Delta L/L_i\ )$$

Create a MATLAB script that enables engineers to input values for each variable and will calculate the elastic modulus.

Project 1 Version 2: Elastic Modulus Calculator

Concepts: Require students to input data and export data to determine the elastic modulus, stress, and strain of material.

Problem: A team of engineers is conducting tensile tests on plastics that may be used in a car part and need to determine the materials elastic modulus, or E. Elastic modulus is a function of the stress on a material (σ) over the material's strain (), which can be further expressed by the following equation, where F is the force applied to the material, A is the material's cross-section, L is the materials initial length, and ΔL is the change in the material's length after the test (or the final length – the initial length).

$$E = \sigma/\epsilon = (F/A)/(\Delta L/L_i\ )$$

Create a MATLAB script that enables engineers to input an excel sheet provided by your instructor containing data for each variable and will output the corresponding elastic modulus onto the excel sheet.

Problem 1 Version 3: Elastic Modulus Calculator

Concept: Determine the stress and strain values independently and graph the resulting stress/strain curve.

Problem: A team of engineers is conducting tensile tests on plastics that may be used in a car part and need to determine the materials elastic modulus, or E. Elastic modulus is a function of the stress on a material (σ) over the material's strain (), which can be further expressed by

the following equation, where F is the force applied to the material, A is the material's cross-section, L is the materials initial length, and ΔL is the change in the length of the material after the test (or the final length – the initial length).

$$E = \sigma/\epsilon = (F/A)/(\Delta L/L_i\ )$$

Refer to your instructor for a data set. Create a code that can create a graph plotting stress data on the Y-axis and strain data on the X-axis. Create labels and a title for the graph.

Project 2: Unit Conversion Calculator.

This set of problems has the user generate codes capable of converting between Standard International and US Customary system units. This operation can be modified to have the user generate graphical user interfaces and more conversion options, requiring a variety of skills addressed within the body of the text.

Project 2 Version 1: Unit Conversion Calculator.

Concept: Write a script to complete a mathematical computation converting units.

Problem: Engineers frequently convert between US customary distances and SI distances when working on projects. To speed up the job of a construction engineer, create a MATLAB code that can convert miles to feet and to meters. There are 5280 feet in a mile and 1609.3 meters in a mile.

Project 2 Version 2: Unit Conversion Calculator

Concept: Write a script using a graphical user interface and an if statement to complete mathematical unit conversions.

Problem: Engineers frequently convert between US customary distances and SI distances when working on projects. To speed up the job of a construction engineer, create a unit conversion script that can convert from either feet, miles, or meters and output lengths in terms of the other two units. Do so using a graphical user interface. There are 5280 feet in a mile and 1609.3 meters in a mile.

Project 3: Material Density Scenarios

Project 3 Version 1: Material Density Scenarios

Concept: Calculate the Density of Various Materials using a graphical user interface.

Problem: You are a mechanical engineer working with several polymer composite materials. Each of these materials has different densities. To save time, you would like a code that can calculate the total mass and gravitational force on a given polymer material with a given volume. Create a code that can select between the following materials, prompt the user to input a volume, and output the total mass of the material.

- Low Density Polyethylene (LDPE): 920 kg/m^3
- High Density Polyethylene (HDPE): 950 kg/m^3
- High Impact Polystyrene (HIPS): 1050 kg/m^3
- 80% Polypropylene + 20% Glass Fiber Composite (PP+GF): 1114 kg/m^3

APPENDIX A: ADDITIONAL MATLAB RESOURCES

MATLAB has become a prevalent resource within engineering schools and companies and as a result, many texts and references are available to help users successfully navigate the software. This appendix outlines various resources that are available either for free or for a cost by both MathWorks and third-party publishers. This appendix can be used as a resource to find help using MATLAB within the rigor of ME 160 and to solve more complicated problems later in coursework or job functions. The remainder of the appendix will list resources that are available along with discussions of their function, availability, and the author's opinions of each text.

MathWorks Website

MathWorks' website contains a variety of resources intended to aid MATLAB users as they navigate the software. As MathWorks profits when engineers use their software, the company provides many resources to ensure engineers can most effectively use the software and are able to find help when needed. The MathWorks website contains extensive documentation about the MATLAB program, various pdfs of textbooks, and several interactive training courses which can be used to gain fundamental skills with MATLAB or even to obtain certifications after completing courses. These resources are detailed below.

MATLAB Documentation

The MATLAB documentation page serves as a resource to look up specific functions of the MATLAB software. Within the documentation page for MATLAB, a user can select broad topics critical to MATLAB which directs the user to more specific webpages addressing introductory resources, language fundamentals, graphics resources and examples, and more. By searching "MathWorks Documentation MATLAB" into a search engine the following page will appear. This is an exceptionally convenient way to review fundamental aspects of

MATLAB that were addressed in ME 160 such as the syntax of specific commands, how to use commands within a code, and more complex graphics or graphical user interface operations.

Within the context of ME 160, the documentation page is most useful to research-specific functions given that the user already remembers the name of the function they are researching. As an example, if the user remembers that they would like to use a for loop but do not remember the syntax that is required, searching for the loop within the "Language Fundamentals" section of the documentation for information on loops and conditional statements.

If the user is unsure of the section of the documentation that contains the information they need, the search tool within the documentation will usually bring the user to the page discussing the application of specific functions in MATLAB. This feature is invaluable when researching a predetermined function, but the documentation tool is limited when the user does not know what function they need to complete a task. Without knowing the name of the desired function, the documentation program is limited.

MathWorks documentation serves as a valuable tool to learn functions like those learned within ME 160. Specific articles within the documentation page contains lists of each function that can be used to accomplish an operation of interest. For example, the documentation page regarding line plots, which is nested under the "Graphics" and "2D and 3D Plots" pages, discusses not only functions taught in ME 160 like plot but other variations such as plot3 or stairs, which enable 3D plots or stairstep graphs, respectively. For the most detailed pages discussing specific functions, selecting specific functions, which should appear as blue hyperlinks, will direct the user to pages dedicated to examples of syntax and situations for a function.

MathWorks MATLAB Guide

In addition to various web pages which can be used to review and supplement MATLAB knowledge on the MathWorks webpage, full texts are available for more comprehensive study. To gain a more thorough understanding of various MATLAB aspects, a list of PDF documents is available on the documentation page. When on the MATLAB documentation page (search "MathWorks Documentation MATLAB" in a search engine and go to the MathWorks website) the user should follow the "PDF Documentation" link in the top right of the page. This page will lead the user to a login page. The user, if they do not already have an account, can make

one easily using an Iowa State email at no cost. This will enable access to the list of all PDFs' produced by MathWorks and can also be used to access additional resources by MathWorks, which are addressed in this appendix.

MATLAB Training through MathWorks

MathWorks offers several training modules which can be accessed through the MathWorks website which provides interactive methods for users to practice writing codes in MATLAB. To find interactive courses offered by MathWorks, search "MathWorks MATLAB Training Courses" and visit the MATLAB and Simulink Training page on MathWorks' webpage. From this page selecting the "Find a Course" tab will lead users to a listing of courses that help users learn specific topics in MATLAB.

The "MATLAB Onramp" course is valuable to ME 160 students who intend to supplement the instruction they are receiving in class. This online module is an interactive MATLAB coding environment that will guide you through exercises and have your complete codes using skills learned in class in applications that extend beyond the scope of ME 160. This module will provide examples and correct code as it is written, given the user real-time feedback and more opportunities to practice writing code.

For more adept coders who are confident with the content addressed in class, various courses that offer instruction on specific topics are available. These courses cover content such as data processing that can supplement other mechanical engineering courses or can provide the user an opportunity to learn a skill that diverges from core curriculum in mechanical engineering, such as MATLAB for financial applications or machine learning applications.

A Practical Introduction to Programming and Problem Solving By: Stormy Attaway

Recommended by some professors at Iowa State to supplement their ME 160 courses, Attaway's introductory MATLAB textbook is a thorough guide to MATLAB that supplements material covered at Iowa State. Within a semester-long introductory course, ME 160 is not always able to discuss details behind why code works the way it does and how nuances of functions make them operate. This text discusses the introductory topics discussed in ME 160 in greater detail, which may assist students in understanding why the code they write works.

Many professors will not refer to the book directly within their classes but will rather

recommend that students purchase the book as supplementary material. I believe this book fulfills this role well and provides a comprehensive and up-to-date discussion of MATLAB at a moderate cost. For students who are interested in understanding nuances of the coding language, this book is excellent.

APPENDIX B: A COMMENTARY ON THIS WORK

This work was written with the goal of creating an easily accessible resource for students enrolled in ME 160 at Iowa State. To do this, the work is being published under an Open Educational Resources (OER) copywrite, which will make the work freely available for educational use. As a result, the authors encourage just that; feel free to use and share this text within academic settings without making modifications to the resources content without prior written permission.

This text and its accompanying slide deck have been written and modified, respectively, by undergraduate students attending Iowa State University. While the team has been closely supervised by Department of Mechanical Engineering Professor Dr. Reza Montazami, the text by no means is perfect or without the need for revisions. Rather you are a professor or student using this work I encourage feedback that may help improve the text or help the reader learn MATLAB. For this reason, I have included contact information which may be used to reach individuals involved with the creation of this text. I do not claim to have mastered MATLAB and am continuing to learn myself. To paraphrase Newton, If I am to see further it is by standing on the shoulders of giants. It is only because of the help of others that I can create this text, so any reader believes that they would be able to contribute to the work, I would love to receive any suggestions, feedback, or revisions. My contact information is provided below:

Austin Bray

ambray@iastate.edu

A slide deck of lecture slides originally created by Dr. Reza Montazami was updated by another former ME 160 student to reflect the content addressed within this work. Each work was intended to complement the other and provide a complete ME 160 learning environment containing a text, lecture slides, verbal lecture, and coding assignments. Feel encouraged to find the resources which are the most effective and spend more time working with those. As this text is expanded and revised, expect similar modifications and expansions to occur to

the slide deck. In a similar manner to this text, do not hesitate to reach out with questions, comments, or concerns about the slide deck, as feedback will influence changes made in the future to the text.

CONTRIBUTORS

Authors

Contributors