A GUIDE TO MATLAB FOR ME 160

AUSTIN BRAY AND REZA MONTAZAMI

Iowa State University Digital Press
Ames, Iowa

CONTENTS

CHAPTER 1: INTRODUCTION

Individuals employed in mechanical engineering careers attempt to solve engineering problems using the tools and information available to them. In situations where engineering needs to interpret, modify, or use data or other mathematical information efficiently the engineer may turn to a coding language to help them complete a task. A computer code functions by telling a computer a set of instructions created by the individual who wrote the code. In addition to running electronic devices around the world, computer code can be written by an engineer to compute mathematical operations, create a graphical depiction of data, or complete work much faster than the engineer would have been able to do by hand. By learning to code, engineers gain a useful tool to use when solving problems, studying data, or completing calculations within research or industry.

ME 160 at Iowa State introduces students to engineering problem-solving methods. Students will be introduced to the MATLAB program as a tool that can be used to solves problems engineers encounter within their education and their careers. To ensure students are best able to learn to code with MATLAB, this guide has been created as an additional reference that supplements the content being addressed by the instructor or other course texts. The first chapter of this text provides a general history and introduction to coding and the MATLAB program, providing background knowledge for students as they begin to explore the MATLAB program.

The history of MATLAB

The original versions of the modern MATLAB program were developed not as a unique coding language but rather strictly as a tool to compute matrix operations. This is evident as its current name, "MATLAB", is short for matrix laboratory. Initially derived in research papers by J.H. Wilkinson at the National Physical Laboratory, computerized matrix and eigenvalue calculating methods were developed in the late 1960s. Researchers at Argonne National Laboratory later created the software EISPACK and LINPACK, which completed matrix/

eigensystem and linear equation calculations, respectively, based upon the coding language Fortran.

By the 1980s Dr. Cleve Moler and his colleagues wrote the programming language MATLAB to make existing programs including EISPACK and LINPACK accessible without needing to use the coding language Fortran. As MATLAB was introduced to Moler's Stanford classes, Moler collaborated with Stanford graduate student Jack Little to commercialize the software. Since 1984 when the first commercial version of MATLAB debuted at the Institute of Electrical and Electronics Engineers' Conference on Decision and Control, the program's capabilities have been expanded to encompass usages in more applications. Significant mathematical topics which have been added to the MATLAB software include differential equations, an assortment of data collection types, and an improved user interface that includes customizable windows depicting everything the user needs to read, edit, and understand a MATLAB code.

MATLAB's role within ME 160

Within ME 160 you will utilize the MATLAB software to compute basic operations, solve algebraic and matrix operations, and utilize codes that will input data and modify equations based on parameters selected by the user. MATLAB should become a tool that the user feels comfortable using to execute computations and to create codes to solve user specifications. The remainder of this book will address the fundamentals necessary to write code in MATLAB, standard applications of the language, and more involved topics which conclude ME 160. At the end of this book and ME 160, the user should understand the MATLAB coding language enough to apply it to solve problems within their education, internships, or future careers.
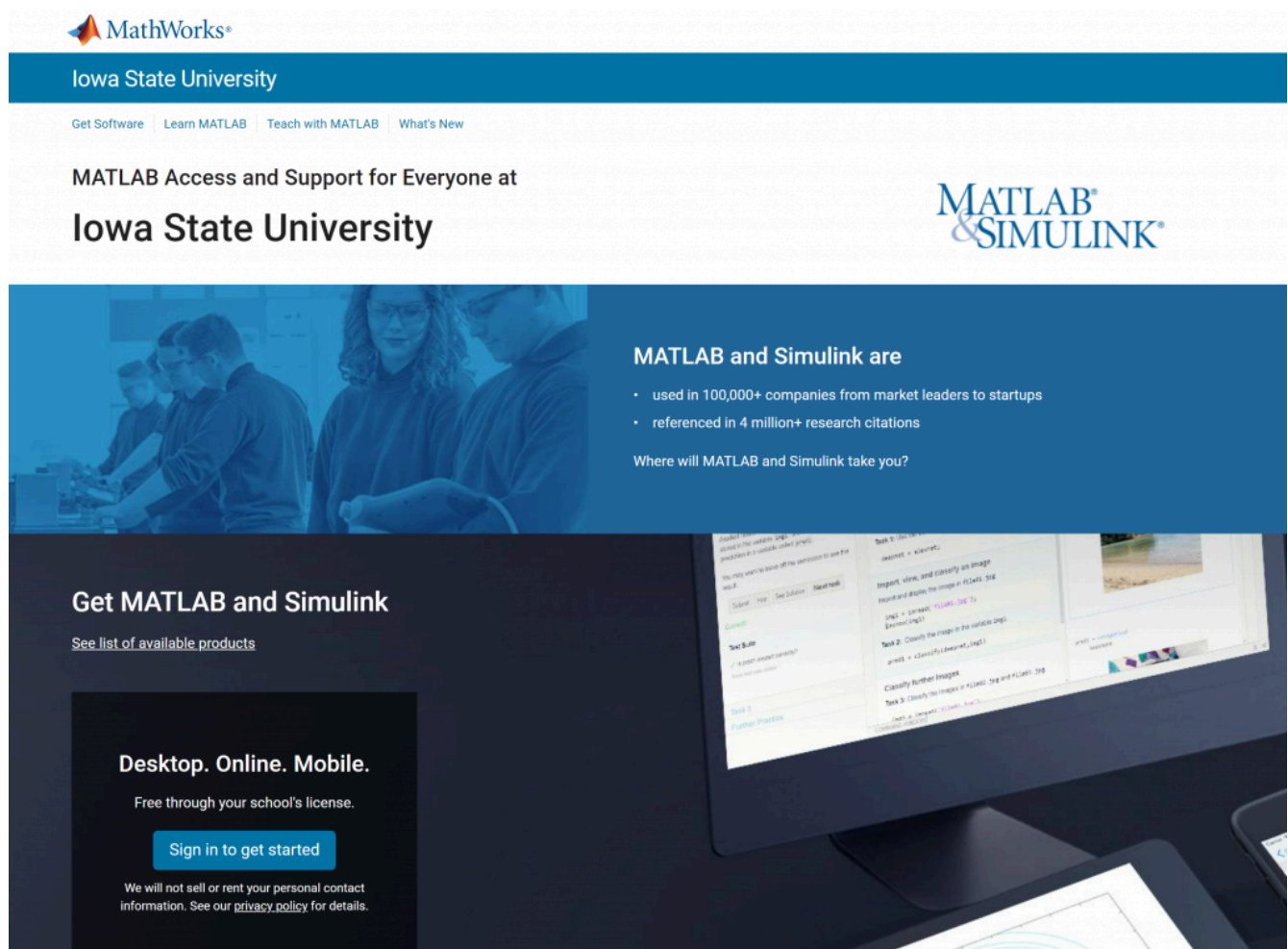
The following list contains examples of situations where MATLAB can be used as a problem-solving tool when working with topics addressed in ME 160. While the list is not exhaustive, it demonstrates a series of skills that the user should have by the completion of ME 160.

- Calculating the values for the resultant of a force system.
- Determine value regarding materials such as sheer, strain, or margins of safety.
- Determine if boats made with different dimensions or materials would sink or float in various fluids.
- Interpreting, calculating, and displaying data sets.

- Development of applications with graphical user interfaces.
- Create repeatable and sharable methods for solving engineering problems.

Accessing MATLAB

The MATLAB program is behind a paywall that requires users or the companies they work for to purchase the software. As students at Iowa State University, you have access to the program without paying out of pocket. To download the software as a student in the College of Engineering, search "Iowa State College of Engineering Installing MATLAB" on a search engine to access Iowa State University's instructions. Following these instructions will direct you to MathWorks Iowa State webpage, where you should create an account using your Iowa State credentials.

After creating an account, read through the instructions on the Iowa State MATLAB installation webpage to download and install the program.

Download MATLAB

Navigate to https://www.mathworks.com/academia/tah-portal/iowa-state-university-1082052.html.

Click **Sign in to get started**.

If you don't already have an account, click **Create Account**.

Fill out the **Email Address** field with your iastate.edu email address. Select **United States** in the **County/Region** drop-down menu. Select the proper title in the **Which best describes you?** drop-down menu. Select the **Yes** radio button next to the **Are you at least 13 years or older?** *if you are indeed over 13 years old*. Click the **Create** button to proceed.

NOTE: You will not be able to download the software using a non iastate.edu email address.

You will receive an email from MathWorks with a link to verify your email address. Click the **Verify your email** button in the email to complete the account creation process.

Fill out all fields and click the **Create** button when all fields are filled out to proceed.

The ISU Matlab license will automatically be assigned to your account. Click the **downward facing arrow** to begin the download process.

Click the **R2019b** (or latest version) button to proceed. Please ask your instructor if there is a specific version you are supposed to use.

Click the button matching your OS to download the MATLAB installer.

Install MATLAB

(**Windows**) Locate the MATLAB installer you downloaded and double-click the exe installer to start the install process.

(**Mac**) Locate the MATLAB installer you downloaded and double-click **InstallForMacOSX** to start the install process.

Select the **Log in with a MathWorks Account** radio button then click the **Next** button to proceed.

Installation instructions provided by Iowa State to download and install MATLAB

Another detail to consider when downloading MATLAB is the version you are downloading. MathWorks releases two versions of MATLAB each year, entitled MATLAB R20XXa or R20XXb, where 20XX corresponds with the year and a or b will denote the first or second version put out that year, respectively. This text was created after R2019b's release, and most images of the software will be taken from this version. If you have a slightly older edition of the software, the most significant differences will be cosmetic modifications and should not cause issues within the scope of ME 160, though I do recommend trying to keep up to date with version updates.

In addition to making MATLAB a freely downloadable software for students, Iowa State University has provided the software on computers located within engineering computer labs. These labs are an excellent resource when students would like to work on MATLAB assignments with peers or would like to bypass the need of downloading software onto a personal device. These labs will update the software to the newest version, which should be the same version used in ME 160.

Learning MATLAB

MATLAB is free and available on campus computers without requiring a download. Search for MATLAB in the start menu to locate the program. Note that it is good practice to ensure that the version of MATLAB available on campus computers is the same version used on any personal device. This can help prevent codes from not being compatible with a new/old version.
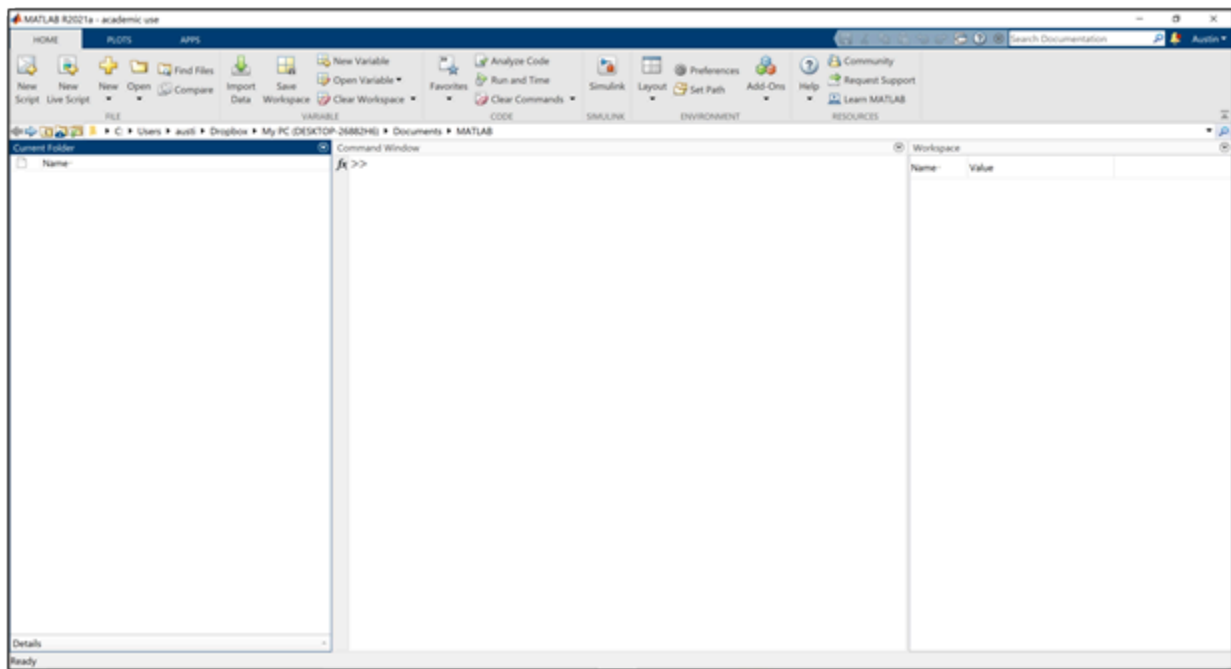
The initial page was viewed after opening MATLAB.

To use MATLAB within ME 160 you must become familiar with the layout of the user interface

of the software. Upon opening the software in Windows, the user will encounter the following page, which is divided into several useful windows. Each window has been colored and labeled in the image below and is discussed in the following section. MATLAB enables the user to make modifications to the position of each window by selecting alternative layouts in the "Layout" button in the toolbar. Different layouts may be more convenient for different users or applications, so feel free to find the layout that is best while coding in ME 160.
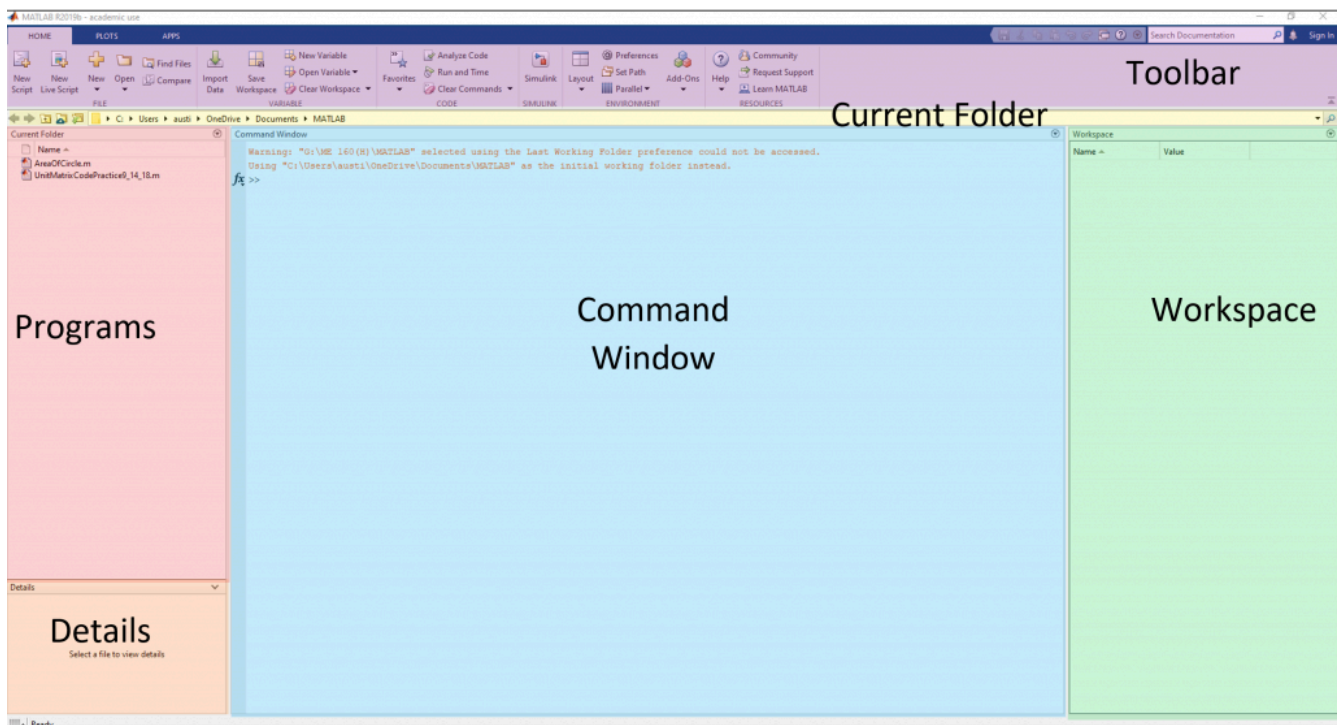
Each element of the MATLAB user interface upon startup. Elements are highlighted for clarity.

The Command Window

The command window (shown in blue) will serve as the primary interface the MATLAB user will interact with when writing and reading simple codes or individual operations. The command window can be used as an interface to type directly into MATLAB and as a place to view the results of these codes. As ME 160 progresses codes will be written in "script" files, which will generate a new window in what is currently shown as the top half of the command window. When running a script, a user will see outputs from the code displayed in the command window and can input data or other information using the command window. The shown example of a command window depicts several mathematical operations that will be addressed at the beginning of ME 160. The highlighted portions show what the user typed into the window, while the remainder is the output by MATLAB.

When the user is done viewing code or is restarting a program, the user must clear anything written in the command window. By using the clear command, clc, the user can reset the command window to its original blank state. For example, completing an arithmetic operation in MATLAB such as 238*5 will result in the window to the right. Typing clc and pressing enter will clear the text in the window.

When writing scripts, you will want to start with a blank command window and reset any operations that the code may have carried out. This can be accomplished by writing two lines of code at the beginning of the script with clc and the close all command. This is demonstrated in the example script below. As a result of these two lines, the text in the command window will reset each time the code is run.

```
SampleCode.m  ✕  +
1 -    close all
2 -    clc
3
4 -    x = 3;
5 -    y = 5;
6
7 -    Hypotenuse = sqrt((x^2)+(y^2))
8
```

```
Command Window

  Hypotenuse =

       5.8310

fx >> |
```
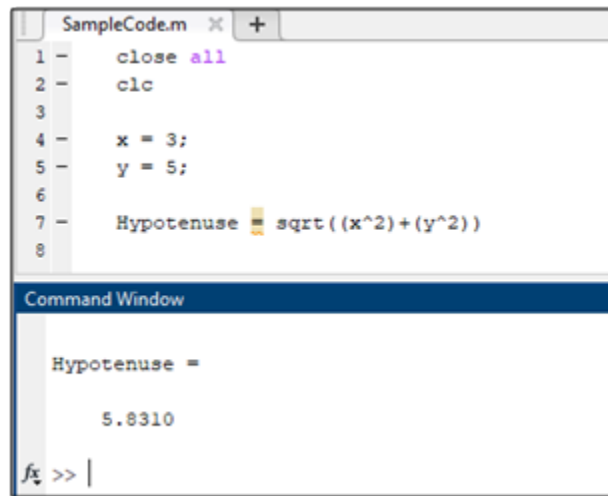
Programs and the Current Folder

Script program files can be saved and referenced later or transferred to other computers, unlike code written directly through the command window. Within ME 160, it will be essential to save all scripts to the same folder, which is referred to as the current folder in MATLAB. The current folder is shown directly below the toolbar and is highlighted in yellow in the above window. By saving your scripts to the same folder you will be able to keep everything in the same place and reference different files without changing folders. Script files, which are saved in .m format, are listed within the red programs section of the MATLAB user interface image above. When a .m file is selected, information about the file will be displayed in the details window, which is depicted in orange in the above image.


```
◄ ► 🔼 📁 📋 | 🗂 ▸ C: ▸ Users ▸ Name ▸ OneDrive ▸ Documents ▸ MATLAB
```



```
Current Folder                        ▼
  📄  Name ▲
  📄 AreaOfCircle.m
  📄 UnitMatrixCodePractice9_14_18.m
```
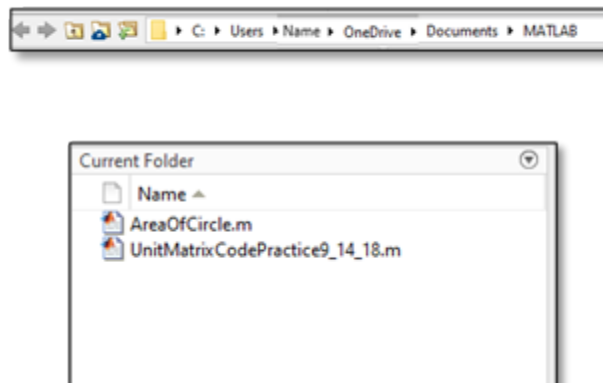
As you continue in ME 160 you will save many files in this section, which will quickly reveal

the importance of careful file naming. Develop a habit of including the name of the program and possibly the date the file was created within the file name to ensure it will be able to be referenced in the future. When naming files in MATLAB, it is important to note that file names cannot include spaces, hyphens, dashes, or numbers as their first digit. To make file names readable, I recommend using underscores and capital letters to make quality file names (i.e. "QuadraticEquation9_23_2019" is much more informative than "untitled5" and will be easy to locate when needed again.

The Editor
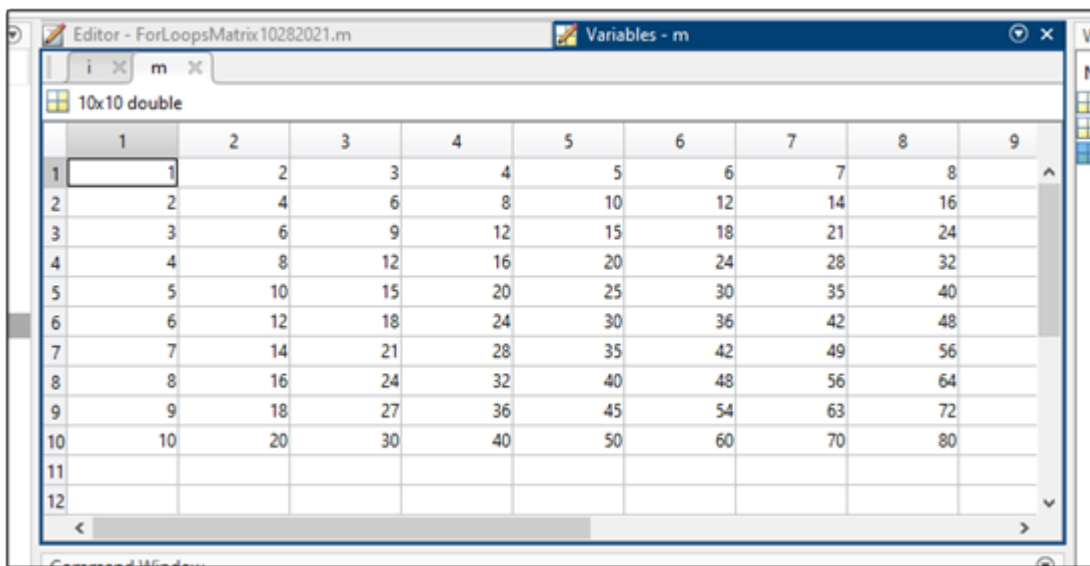
The editor window is the home of MATLAB script files- savable codes that can be shared and saved on a computer for use. MATLAB scripts are the files where all stand-alone programs that are written in ME 160 will be saved. When working in the command window every time an operation is typed and the enter key is pressed, the operation is completed by MATLAB. This fact means that codes are run step-by-step when written in the command folder. This is not the case for scripts within the editor. Codes are written by the user and only ran when commanded by the user by selecting the run button in the editor window of MATLAB.
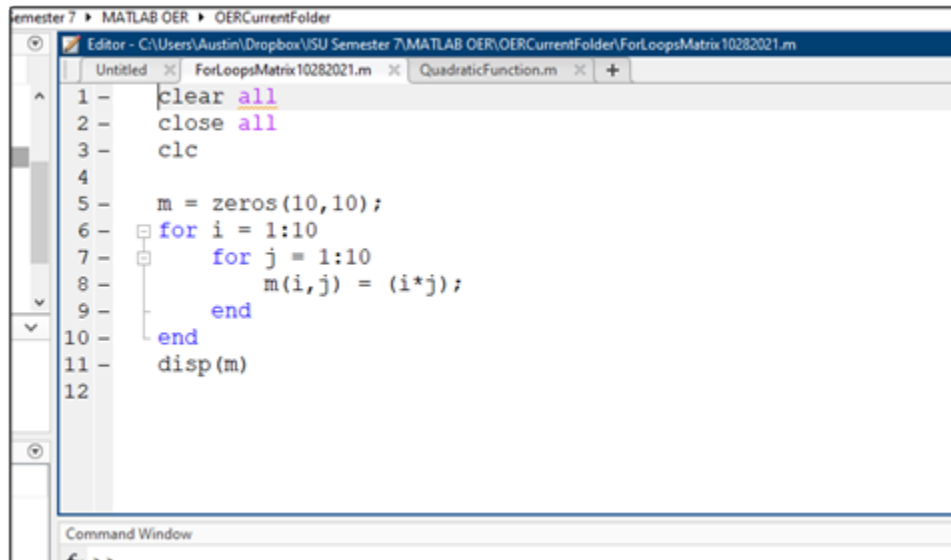
Multiple scripts opened at the same time overlaid with tabs showing the name of each script. Like programs in Microsoft Windows, scripts can be dragged and dropped to be viewed next to

each other or above and below each other, allowing for easier usage when working on multiple scripts.

The editor window also is home to expanded views of variables. When a variable from within the variables window is selected by double-clicking, a window will overlay the editor and any open scripts to show the contents of the variable. This may be a single value, a matrix or array, or a symbolic input. Users may shift back and forth between viewing the contents of variables and scripts using the tabs that appear at the top of the editor window.
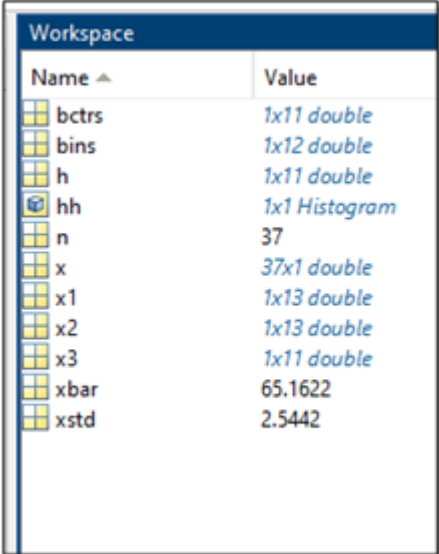
	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	
2	2	4	6	8	10	12	14	16	
3	3	6	9	12	15	18	21	24	
4	4	8	12	16	20	24	28	32	
5	5	10	15	20	25	30	35	40	
6	6	12	18	24	30	36	42	48	
7	7	14	21	28	35	42	49	56	
8	8	16	24	32	40	48	56	64	
9	9	18	27	36	45	54	63	72	
10	10	20	30	40	50	60	70	80	
11									
12									

The Workspace

When writing scripts in MATLAB many variables will be introduced to represent data or to be used in formulas. To assist the user in determining which variables have been assigned in a script, the workspace will show each variable and the values that are assigned to them. This will be helpful when writing long codes using many similar variables and will help keep track of variable meanings.

Various forms of variables that exist within the MATLAB coding language are displayed within the Workspace. This includes numbers, characters, words, and more complicated double objects with arrays and matrices, and histograms. An example of a workspace for a code with

several variable types is included below. Notice how each variable is assigned its own name by the user, allowing for it to be identified within the code.

Workspace	
Name ▲	Value
bctrs	1x11 double
bins	1x12 double
h	1x11 double
hh	1x1 Histogram
n	37
x	37x1 double
x1	1x13 double
x2	1x13 double
x3	1x11 double
xbar	65.1622
xstd	2.5442

CHAPTER 2: BASIC COMMANDS IN MATLAB
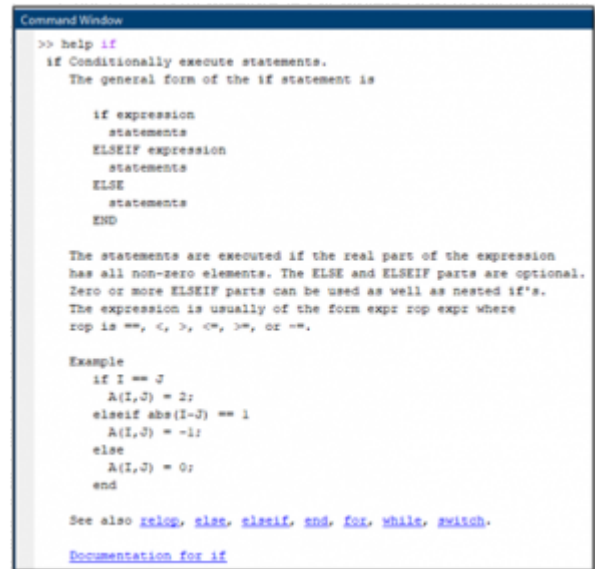
Introduction

To get started writing code in MATLAB, several commands are essential to begin to operate the program. A command is a basic instruction that the user gives the program. These instructions, if given correctly, cause MATLAB to output a response such as printing a solution to a problem, executing a step in a larger process, or relaying information about the code. This chapter details several common commands which will frequently be used when writing scripts. The MATLAB user interface includes a command window designed specifically to facilitate the user executing simple commands or interacting with specific parts of larger codes, which will be addressed in the discussion of scripts. The command window and commands as a concept are fundamental concepts for coding. As a result, the following chapter has been written to instruct users on the variety, effects, and syntax required for command use in the broader context of MATLAB.

Help Resources

MATLAB is a complicated programming tool that contains many functions which can be used in an assortment of applications.

When writing code for ME 160, there are several ways for the user to learn more about functions and their operations while writing code. When the user knows the name of a command and is wanting to learn more about its function or how it should be written into the code, the user should use the help command. For example, to learn more about the if function, the user would time help if into the command window.

Many other resources exist outside of the program within both the MathWorks' webpage and resources produced by third-party vendors. Appendix I of this text discusses many resources that are provided outside of MATLAB software itself. Specifically, resources provided within the documentation page are most valuable within the context of ME 160. The documentation page for a specific function can be reached by following the hyperlink at the bottom of the help page in MATLAB or by searching for the "MathWorks MATLAB Documentation" in a search engine.

```
Command Window
>> help if
 if Conditionally execute statements.
    The general form of the if statement is

        if expression
            statements
        ELSEIF expression
            statements
        ELSE
            statements
        END

    The statements are executed if the real part of the expression
    has all non-zero elements. The ELSE and ELSEIF parts are optional.
    Zero or more ELSEIF parts can be used as well as nested if's.
    The expression is usually of the form expr rop expr where
    rop is ==, <, >, <=, >=, or ~=.

    Example
        if I == J
            A(I,J) = 2;
        elseif abs(I-J) == 1
            A(I,J) = -1;
        else
            A(I,J) = 0;
        end

    See also relop, else, elseif, end, for, while, switch.

    Documentation for if
```

As the reader is most likely just beginning to code in MATLAB, interactive tutorials offered on the MathWorks webpage may prove as valuable supplements to in-class work. Mathworks has provided many interactive MATLAB coding tutorials which enable the user to walk through exercises in a virtual "MATLAB Onramp" course. For more information about MathWorks interactive programs, refer to Appendix I at the end of this text.

Comments

Comments are a built-in way MATLAB has enabled users to make notes within a code without affecting the code's function. To make a comment within a code, the user can type % at the beginning of a line of code. MATLAB will then ignore everything typed after that percent sign, enabling the user to type information regarding the function of the code or what a part of the code does. Comments are essential to writing efficient, long codes that can easily be read and edited by both the initial author and their peers. By placing many comments throughout codes in ME 160, the author will ensure that the code is easily read and edited by themselves, their peers, and the faculty grading final codes. Forming the habit of commenting often throughout a code is critical to ensuring that codes produced within ME 160 are useful and effective.

Comments have also been designed to help users who are editing code and would like to "turn off" a section of code. In the instance that a section of code within a MATLAB script does not work or should not be run, the user could place comments with % signs at the beginning of

each line that should be ignored by the code. In order to speed up the process of "commenting out" large sections of code, MATLAB has created the notation %{ and %} which can be placed on different lines of code and will comment everything written in lines of code between the symbols. An example of standard comments and a block comment follows below. In the example, a portion of code that does not work correctly has been commented out and a note to correct the code has been made. Comments such as the one in line three are essential to ensure that codes are as legible as possible for all users.

```matlab
1   close all
2   clc
3   %This code will generate a plot with two curves
4   x1 = 0:1000:(2*(3.1415));
5   x2 = 0:20:(2*(3.1415));
6
7   y1 = sin(x1);
8   y2 = sin(x2);
9   %{
10  plot(x1,y1,'c',x2,y2,'r*')
11  xlabel('X')
12  ylabel('Y')
13  title('Sine Wave')
14  %}
15  %Above section broken, correct code.
```

Arithmetic in MATLAB

MATLAB is a well-versed tool for completing simple mathematical operations or solving algebraic equations. To complete these operations, the user must be aware of the notation that is required for the computer to understand what was input.

General Operations

To complete addition, subtraction, multiplication, or division the user must be precise with how they write their expressions. The included image depicts how to properly input

operations. The first thing the user should be mindful of is the lack of an equal sign in mathematical operations. MATLAB uses single equal signs to assign a value to a variable, which is not what we want to do when solving an expression. Instead, press enter without typing an equal sign to tell MATLAB to solve the expression. Note that operations do not depend on if spaces are present between the numerical values and the operator. For multiplication and division note the use of an asterisk (*) and a forward slash (/) as their respective operators. Several operations can be done by inputting values with consideration of conventional order of operations rules (think the acronym PEMDAS for parenthesis, exponents, multiplication, division, addition, and subtraction if you forget).

```
>> 5*5

ans =

    25

>> 5*(5^2)

ans =

   125

>> 45/(2/3)

ans =

   67.5000
```

Exponents and roots can be calculated in a similar manner in MATLAB. To compute an operation with an exponent, use a caret (^) followed by the value of the power. To compute roots, the same notation can be used with the root depicted as an exponent. For example, four squared could be written as 4^2 in MATLAB and the square root of four could be written as either 4^(1/2). The sqrt function can also be written using the `sqrt()` function, with all values which would be under the radical being contained within the parenthesis.

Calculating expressions using logarithms and exponentials are easy using MATLAB as well. To calculate a base 10 logarithm, use the function "`log(x)`" with `x` being the numerical value of the logarithm. In a similar manner, values taken to a power of e can be written using "`exp(x)`", which would be input as "`exp(4)`" in MATLAB.

Trigonometry in MATLAB

MATLAB defaults to using radians to calculate degrees, which requires the user to be careful when entering angles to ensure the value is not calculated incorrectly. The functions `degree()` and `radian()` can be used to convert between an angle in degrees and an angle in radians. When computing trigonometry operations such as `sin()` or `cos()` in MATLAB, the alternative function `sind()`, `cosd()`, or `tand()` can be used to inform MATLAB that

the value entered in the trigonometry operation is in degrees. This saves the user the need to convert between radians and degrees.
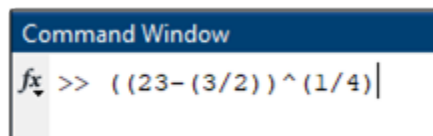
Complex Expressions

MATLAB can calculate complex expressions in the same manner as simple expressions. MATLAB, like any traditional calculator, calculates specifically based on what the user inputs. While expressions are input using the same formatting regardless of their complexity, it is increasingly important to consider the order of operations that the system will use as a result of parenthesis or the lack thereof placed by the user. To get a feel for typing complex expressions within MATLAB's command window, review the following examples and reproduce them within MATLAB yourself.

Examples

The following examples demonstrate how to input complex expressions into the command window.

1. $\sqrt[4]{\left(23 - \dfrac{3}{2}\right)}$

```
Command Window
fx >> ((23-(3/2))^(1/4)|
```

2. $\left[\dfrac{10\exp(3/2)}{\sqrt{23}}\right]$

```
Command Window
fx >> ((10exp(3/2))/(sqrt(23)))|
```

3. $\dfrac{-2 + \sqrt{2^2 - 4(4)}}{2}$

Command Window

f_x >> (-2+(sqrt((2^2)-(4*4))))/2|

4. $14 \times \left(6.022 \times 10^{23}\right)$

Command Window

f_x >> 14*(6.022*(10^23))|

5. $\left[\dfrac{3^{3/2} - \sqrt{3}}{8 + \left(\frac{4}{13}\right)}\right]^3$

Command Window

f_x >> (((3^(3/2))-3^(1/2))/(8+(4/13)))^3|

Problems

Practice writing the following expressions in MATLAB. Be careful to use proper parenthesis and order of operations. For problem 5, show each equation being solved using MATLAB.

1. $e^{\left(\frac{13}{3}\right)}$

2. $\sqrt[3]{23 + 1.5^3}$

3. $\left[\dfrac{\sqrt{72}}{5^{(2/3)} + 13}\right]$

4. $10\left(1 - \left(\dfrac{.06}{.5}\right)^2\right)$

5. The velocity of an object is represented by the following function:

$$V(t) = \sqrt[3]{4t^3 + \frac{1}{2}t^2 + 2t + 100}$$

Using MATLAB, determine the velocity of the object at times t = 10 seconds, 50 seconds, and 100 seconds. (Note: While this problem may be more easily solved using a calculator, MATLAB could be used in the future to calculate the velocity at many different times. To create that code, the author would need to correctly input the function, which is what this problem exercises.)

6. A titanium ball is dropped into a viscous fluid and slowly sinks to the bottom. A 5 mm diameter spherical ball is dropped into the liquid at a constant speed such that it takes 5 seconds for the ball to drop .15m. The density of titanium is $440\frac{kg}{m^3}$ and the density of the fluid is $800\ \frac{kg}{m^3}$. Determine the viscosity of the fluid using the following equations. Note that D is the sphere's diameter, ρ (rho) is density, g is gravitational acceleration, (9.8 m/s), and Δh is the change of the ball's height.

$$V_{sphere} = \frac{\pi(D^3)}{6}$$

$$Weight\ of\ Sphere = W = \rho_{sphere}\, g(\Delta h)$$

$$Weight\ of\ Sphere = W = \rho_{sphere}\, g(\Delta h)$$

$$Weight\ of\ Sphere = W = \rho_{sphere}\, g(\Delta h)$$

$$Viscosity = \mu = \frac{F_D}{3\pi(D)(Velocity)}$$

CHAPTER 3: MATRIX OPERATIONS

Introduction

MATLAB serves as a powerful tool to solve matrices. To use matrices as a tool to solve equations or represent data a fundamental understanding of what a matrix is and how to compute arithmetical operations with it is critical.

What is a Matrix?

A matrix is a rectangular array or grid of values which arranged in rows and columns. Matrices are used to operate on a set of numbers with variations of traditional mathematical operations. Matrices serve valuable rolls within many engineering and mathematic tasks due to their useful ability to effectively store and organize information. Understanding matrices proves valuable when trying to solve systems of equations, organizing data collected during experiments, computing mathematical operations on large quantities of numbers, and complicated applications in linear algebra, machine learning, and optimization.

When describing matrices, we will name them based on the number of rows and columns. For example, the following matrix is a 2×3 matrix as it has two rows and three columns.

$$\begin{bmatrix} 2 & 4 & 65 \\ 3 & 2 & -8.5 \end{bmatrix}$$

And this matrix is a 4×3 matrix:

$$\begin{bmatrix} 1 & -21 \\ 2 & 25 \\ 3 & 12 \\ 4 & -11 \end{bmatrix}$$

Matrix Arithmetic

Matrices are an effective way to modify an entire set of numbers in one operation. Simple ways to modify matrices include addition, subtraction, multiplication, and division by a scalar, or individual number. When completing these operations, complete the calculation with each number in the matrix, as denoted below.

$$\begin{bmatrix} 1 & 2 \\ 4 & 3 \end{bmatrix} + 2 = \begin{bmatrix} 1+2 & 2+2 \\ 4+2 & 3+2 \end{bmatrix} = \begin{bmatrix} 3 & 4 \\ 6 & 5 \end{bmatrix} \leftarrow Answer$$

$$\begin{bmatrix} 2 & -4 \\ 1.5 & 3 \end{bmatrix} * 3 = \begin{bmatrix} 2*3 & -4*3 \\ 1.5*3 & 3*3 \end{bmatrix} = \begin{bmatrix} 6 & -12 \\ 4.5 & 9 \end{bmatrix} \leftarrow Answer$$

Matrices with the same dimensions (i.e. two 2×2 matrices) can have more mathematical operations completed with them. For example, you can add or subtract matrices with the same dimensions by completing operations on the values in each corresponding location in a matrix. The following shows a template for adding or subtracting two matrices.

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} + \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} (a+e) & (b+f) \\ (c+g) & (d+h) \end{bmatrix}$$

Multiplying matrices is more difficult than adding and subtracting and does not follow the format listed above. The process known as element-wise matrix multiplication is shown below. This process for multiplying matrices is a fundamental concept of linear algebra and occurs when working with matrices in MATLAB. Be aware of the general form shown below and that it can be extrapolated to include matrices of different sizes. An alternative method of multiplying two matrices that are the same size is called component-wise multiplication, which would follow the same form as the matrix addition shown above. The procedure for coding these into MATLAB are shown below.

Vectors and Matrices in MATLAB

Inputting Matrices

It is easy to input matrices into MATLAB scripts. To make a standard matrix in the command window, use the following format with values of a matrix listed with spaces between each value. Use a semicolon to separate each line of the matrix. To see how this process looks within MATLAB, refer to the examples at the end of this section.

```
>> [1 2 3;4 5 6;7 8 9]
```

Which produces $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$ in MATLAB.

Note that to create an array list each number in a row separated only by spaces. To move down to a new row, use a semicolon. To save time making a large array, a colon can be used to "list" numbers. For example, 1:5 would create a row containing 1, 2, 3, 4, and 5. For example,

```
>> [1:3;4:6;7:9]
```

creates the same matrix as the first example. If you would like to create a matrix that counts by a unit other than one, add a second colon that denotes what numbers will be included. For example,

```
>> [2:2:10;12:2:20]
```

will create the following 2 row by 5 column matrix which counts by twos between 2 and 10 in the top row and 12 and 20 in the bottom row

Matrix Operations and Concatenating Matrices

Examples

1) Enter the following matrix efficiently into MATLAB.

$$\begin{bmatrix} 1 & 2 & 3 \\ 7 & 8 & 9 \end{bmatrix}$$

2) Enter the following matrix efficiently into MATLAB.

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 7 & 9 & 11 & 13 & 15 & 17 \\ 18 & 18.5 & 19 & 19.5 & 20 & 20.5 \end{bmatrix}$$

3) Use the following matrices in the following parts.

$$a = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \text{ and } b = \begin{bmatrix} 2 & 4 \\ 6 & 8 \end{bmatrix}$$

3a) Input the above matrices into MATLAB. Assign each the variable name shown.

Note that by placing semicolons at the end of the line the output is suppressed. As a result, the actual matrices are not printed in the code, which saves space in this instance.

IMAGE

3b) Add matrix a and b to each other.

IMAGE

3c) Subtract matrix a from matrix b.

IMAGE

3d) Multiply matrix a and matrix b using component-wise multiplication.

IMAGE

3e) Multiply matrix a and matrix b using matrix multiplication.

IMAGE

Problems

Efficiently type the following matrices into MATLAB's command window.

1. $\begin{bmatrix} 1 & 4 & 7 \\ 2 & 3 & 4 \\ -3 & 0 & 3 \end{bmatrix}$

2. $\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 10 & 11 & 12 & 13 & 14 & 15 \end{bmatrix}$

3. $\begin{bmatrix} 1 & 4 & 7 & 10 & 13 & 16 \\ -2 & 0 & 2 & 4 & 6 & 8 \end{bmatrix}$

4. $\begin{bmatrix} 0 & 0.5 & 1 & 1.5 & 2 \\ 1 & 1.25 & 1.5 & 1.75 & 2 \\ 5 & 4 & 3 & 2 & 1 \end{bmatrix}$

Use these matrices to complete the following computations using MATLAB.

$$a = \begin{bmatrix} -8 & 4 \\ 5 & 12 \end{bmatrix}; \quad b = \begin{bmatrix} 3 & 5 \\ 2 & 3 \end{bmatrix}; \quad c = \begin{bmatrix} -2 & 1.5 \\ 12 & -4.25 \end{bmatrix}; \quad d = \begin{bmatrix} -2 & 0 \\ 2 & 4 \end{bmatrix}$$

5. $a + b$

6. $(a + b) * c$

7. $(c * d)$

8. $(a - b) * (c + d)$

CHAPTER 4: WRITING SCRIPTS

Variables

Variables are critical pieces of MATLAB codes that users can assign values to when writing scripts. This chapter will address uses of variables to solve algebraic operations and to analyze data. The following image shows an example of the format for creating a variable. In this example, the variable's name is the letter 'n' and the variable's value is 6.02e23. A variable has value since MATLAB will know that everywhere in a script where 'n' is typed the value 6.02e23 is transferred. This functionality at its simplest form enables users to keep code clean by writing variables instead of a number repeatedly. Importantly, if the user changes the value of the variable where it is initially defined in a code, then the value changes everywhere else in a code.
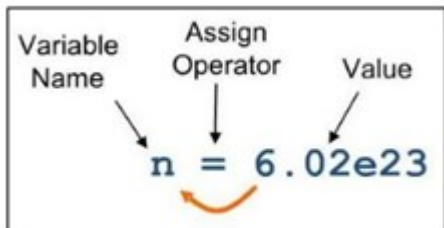
Variable named "n" being assigned the value of 6.02×10^{23}.
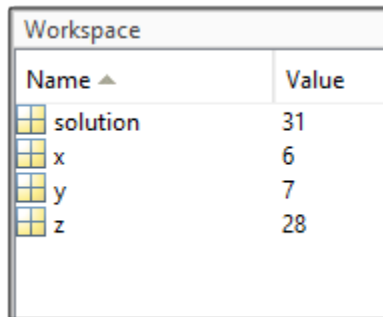
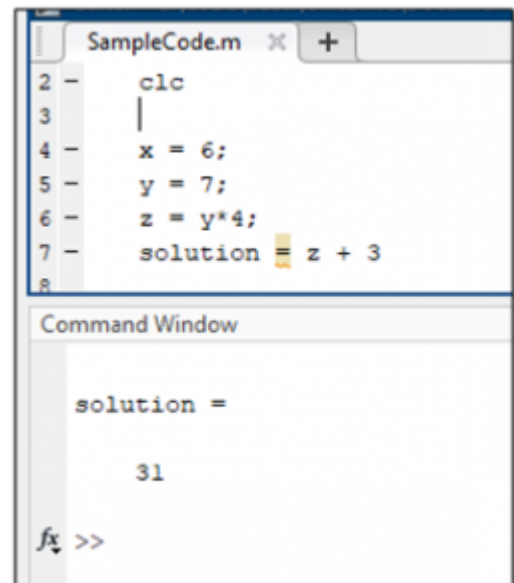Example of variables being assigned in the command window.

Variable Operations

Once variables have values assigned to them, they become useful tools for numeric operations, and data analysis, and will be critical in functions, which will be addressed next. The following

images show examples of how variables can be used in numeric and matrix applications. Once variables have been assigned numeric values, the user can view the workspace window to check the value assigned to each variable. This feature is valuable for long codes with many variables, as the user is able to check the value for each variable and keep track of which variable names have been used.

```
SampleCode.m  ×   +
2 -     clc
3
4 -     x = 6;
5 -     y = 7;
6 -     z = y*4;
7 -     solution = z + 3
8
```

Command Window

```
solution =

    31

fx >>
```

Workspace	
Name ▲	Value
solution	31
x	6
y	7
z	28

When creating equations within MATLAB codes, values assigned to variables can be checked to verify if equations are operating how the user intended them to. For example, the user can manually calculate the expected values generated by an equation and can check actual values displayed in the workspace to ensure that a code functions as intended by the user.

Examples

1. Define two variables a and b which are assigned the values 12 and 3, respectively. Create a code that outputs the values for a divided by b, a multiplied by b, and a to the b power.

```
1 -    close all
2 -    clc
3
4 -    a = 12;
5 -    b = 3;
6
7 -    x = a/b
8 -    y = a*b
9 -    z = a^b
```

```
Command Window

    x =

        4

    y =

        36

    z =

        1728
```
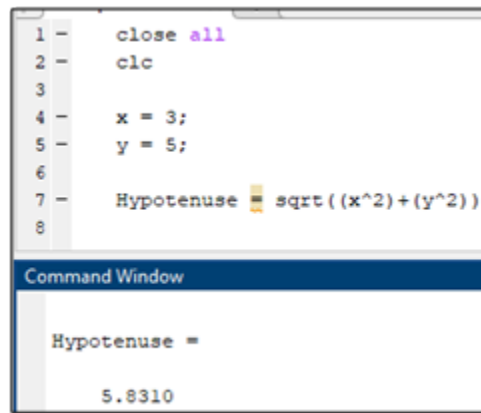
Assign the outputs the values x, y, and z, respectively.

Note that the answer displayed in the command window only depicts values that were not suppressed in the code by semicolons at the end of a line.

2. The Pythagorean theorem can be used to determine the length of the right triangle's sides. Write a MATLAB script that computes the hypotenuse for a right triangle with short sides x = 3 and y = 5. The most useful form of the Pythagorean theorem is provided.

$$Hypotenuse = \sqrt{x^2 + y^2}$$

```
1 -     close all
2 -     clc
3
4 -     x = 3;
5 -     y = 5;
6
7 -     Hypotenuse = sqrt((x^2)+(y^2))
8
```

Command Window

```
Hypotenuse =

       5.8310
```

Problems

1. The following equation can be used to calculate the distance that an object will free fall over some amount of time. Assuming that the object starts at zero velocity and has no air resistance, create a code that can calculate the total distance traveled by an item in free fall for a given gravitational acceleration, g $(\dfrac{meters}{second^2})$, and time, t $(seconds)$.

$$\Delta h = 1/2gt^2$$

2. An engineer would like to easily calculate the volume and surface area of a classroom in a building. Create a script file within MATLAB that can calculate the volume of the room and the surface area of wall #1, wall #2, and the floor, as labeled in the image, by assigning values to variables x, y, and z, and completing arithmetic operations with them. Use comments to label each part of the script.
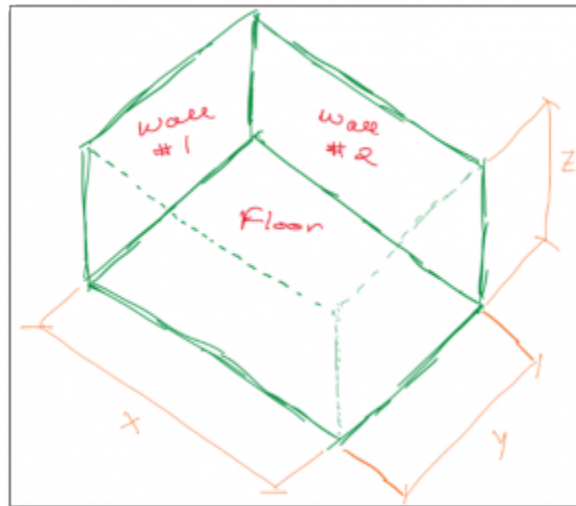
3. A scientist working with high-power electron microscopes would like to convert units from micrometers and nanometers to meters so students can understand how small these units are. Create a MATLAB script that converts a variable with units in meters to micrometers (μm) and nanometers (nm). The necessary conversion factors are provided below. Use comments to label your code.

$$10^6 \mu m = 1m$$

$$10^9 nm = 1m$$
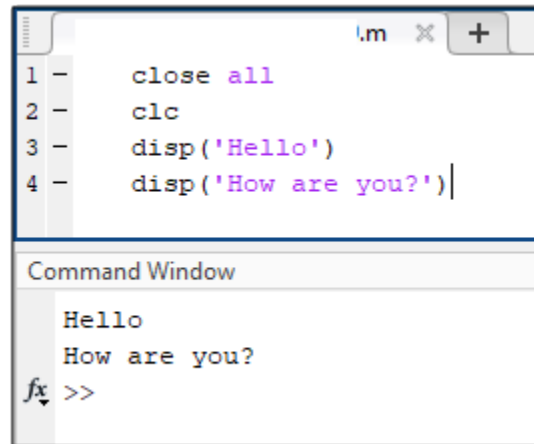
CHAPTER 5: COMMANDS

Introduction

Commands will serve as tools that complete defined tasks within a MATLAB code. Often represented as puzzle pieces in introductory programming guides, commands enable the code to interact with the user, finish tasks in response to user inputs, or serve as a road map within the code. By piecing together commands with instructions a "map" will be created which enables the code to complete the user's desired operation. This chapter will introduce several functions which will be prevalent within ME 160 and will drastically expand the number of applications for MATLAB.

The Display Command (disp)

The display command (typed "disp") allows the user to instruct the program to display a message in the command window. The display command is an excellent way to give instructions or other information in the command window for the user of the code. This can include instructions about the code's function that appear when the user runs the code, greetings for the user, error messages, or conclusions determined by the code.

The display command appears in the following format in MATLAB. To use the display command, type disp followed by a set of parentheses with single quotations inside of them. Everything else in the quotations will be displayed in the command window by the program.

```
disp ('Message Here.')
```

```
                                          l.m    X    +
1 –        close all
2 –        clc
3 –        disp('Hello')
4 –        disp('How are you?')|
```

Command Window

```
       Hello
       How are you?
fx >>
```
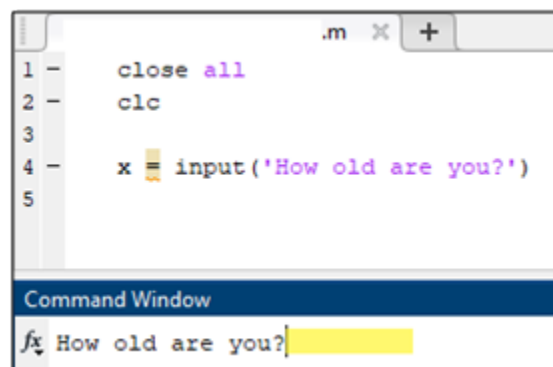
Note that the message being displayed cannot contain apostrophes, as the code will stop reading the message there. Additionally, it is important to understand that this command is not the best way to prompt users to input data or other information into the code, as the input command is best suited for that function.

The Input Command (input)

The input function, typed input, operates as a way for the user to assign a value to a variable. With the input command, the user can have MATLAB provide a prompt in the command window. The user is then able to respond to the prompt by entering in their input directly after the prompt in the command window. An example of the input command is to the right. The user would type their response where the yellow box is next to the prompt. Note that the variable x will be assigned the user input for the remainder of the code unless another function reassigns its value.

```
                                          .m    X    +
1 –        close all
2 –        clc
3
4 –        x = input('How old are you?')
5
```

Command Window

```
fx How old are you?|
```
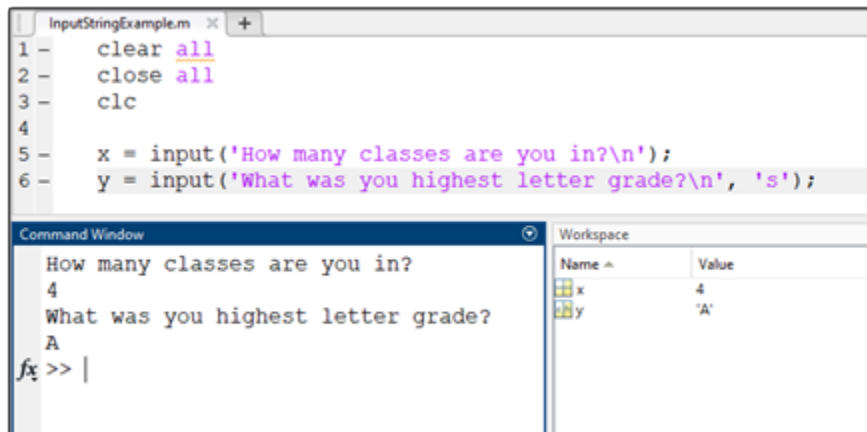
To improve the appearance of this function the user can add \n to the command, which will cause everything following the symbol to appear in the next line. If added to the end of the display command, as shown in the second example, the prompt to type an input appears in the line below the text in the command window and not next to the text, as indicated by the yellow box in the image. This is an easy way to improve the appearance and usability of a code, which will ensure data is properly input and that the user easily operates the code.

The method of using the input command introduced is designed for inputs that are strictly numeric. There are many applications where the user would like to input a value that is not a number and instead is letters or words. Many programming languages refer to a series of characters that consist of letters as "strings". MATLAB follows this convention and has modifications to the input command for string inputs. The example below shows that the input command can receive strings by modifying the end of the command with 's' after a comma after the standard input prompt. Several examples in the MATLAB script shown below compares numeric and string inputs.

```
>> example = input('Prompt typed here', 's')
```

```
InputStringExample.m  ×  +
1 –     clear all
2 –     close all
3 –     clc
4
5 –     x = input('How many classes are you in?\n');
6 –     y = input('What was you highest letter grade?\n', 's');
```

```
Command Window                              ⊙
    How many classes are you in?
    4
    What was you highest letter grade?
    A
fx >> |
```

```
Workspace
Name ▲          Value
 x              4
 y              'A'
```

The fprintf Command

The `disp` and `input` functions enable the user to display information and input information into the code. However, these functions will be insufficient if the user wants to output information that changes based on inputs or other values in a script. As an example, let's say a code was written to find the square of user input. The code may use a display function to inform the user of the code's function and an input function to prompt the user to input a value. However, since the output changes depending on the input, a display value is unable to update to show this calculated output. The function `fprintf` is designed to display the value assigned to a variable inside of a text output message. The format for the fprintf is as follows:

```
>>fprintf ('Text text text %f text text text %f text text text %f
\n',x,y,z)
```

Place a "`%f`" in each location where the value assigned to a variable should be displayed amongst a fixed message. The values of each variable input by the user are listed at the end of the function in a list separated by commas. Note that the included example is with three variables that happen to be named `x`, `y`, and `z`, respectively. `fprintf` follows the same format as the example when different numbers of variables are present.

The following example demonstrates `input`, and `fprintf` functions used together to provide the user instructions, allow the user to input data, and to output the processed data in a text message.

```
clc

clear

a=input('insert a:    ');

b=input('insert b:    ');

p=a*b;

n=a/b;

fprintf('%f multiplied by %f is %f \n and \n %f divided by %f is %f
\n', a,b,p,a,b,n)
```
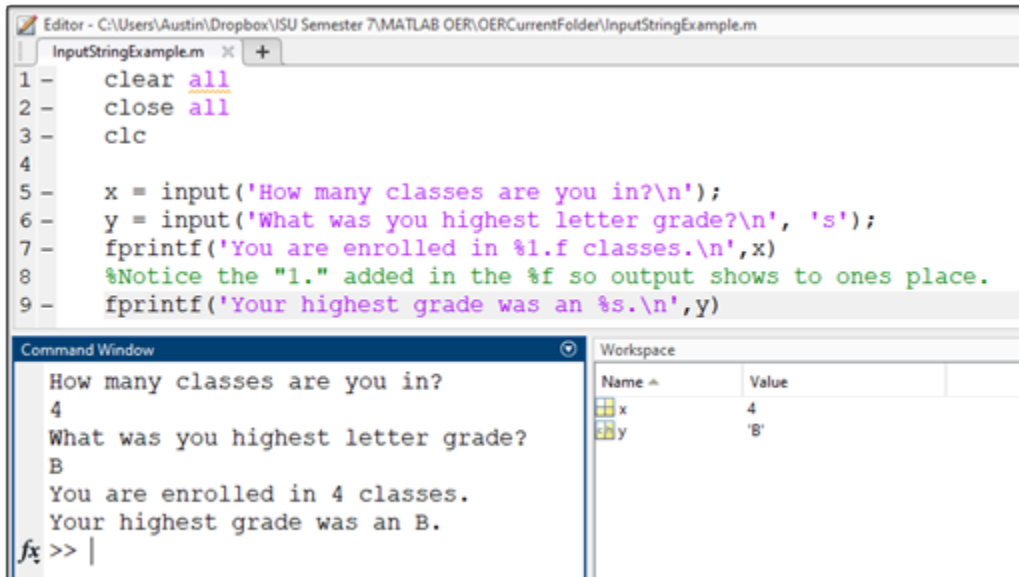
When using fprintf, the %f placeholder in the text output is specific to numeric variables. This means that it can only output numeric values. Alternative placeholders exist for different types of outputs that the user desires. The most common alternative to %f that may be encountered is %s, which outputs a string variable. String variables, as introduced previously, are letters or phrases assigned to a variable. Using %s allows users to output specific words within the defined output message. This is shown in the following example. Alternative placeholders can be viewed within the fprintf documentation on the MathWorks webpage that allows customization of everything from the decimal length in output to scientific notation to integer outputs.

Common Placeholder in fprintf	Usage
%f	Fixed point output (Most commonly used placeholder)
%s	Outputs a series of characters or a string
%i	Outputs an integer
%e or %E	Scientific notation with "e" displayed as a lowercase or uppercase, respectively
%g	Fixed point output (like %f) without trailing zeros.

Modification of the %f operator can allow the user to control the precision of the output

number. By default, the output number will include a large series of zeros at the end, which can be visually distracting and incorrect if that level of precision is not actually known.

```
Editor - C:\Users\Austin\Dropbox\ISU Semester 7\MATLAB OER\OERCurrentFolder\InputStringExample.m
InputStringExample.m  ×  +
1 -    clear all
2 -    close all
3 -    clc
4
5 -    x = input('How many classes are you in?\n');
6 -    y = input('What was you highest letter grade?\n', 's');
7 -    fprintf('You are enrolled in %1.f classes.\n',x)
8      %Notice the "1." added in the %f so output shows to ones place.
9 -    fprintf('Your highest grade was an %s.\n',y)
```

```
Command Window                                    Workspace
How many classes are you in?                      Name ▲      Value
4                                                  x          4
What was you highest letter grade?                y          'B'
B
You are enrolled in 4 classes.
Your highest grade was an B.
fx >> |
```

Rounding

MATLAB has functions developed to round integers or array values within a code. The following functions are useful for rounding with standard rounding conventions, rounding to the floor, and rounding to the ceiling. These functions operate by modifying a variable and creating a new rounded variable. The following example shows how to round some variable a conventionally, to the floor, and to the ceiling and create a variable, b.

```
b = round(a)
```

```
b = floor(a)
```

```
b = ceiling(a)
```

if Statements

if statements enable you to write code that will only be executed if predetermined conditions

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):

$$°C=(°F-32)×5/9$$

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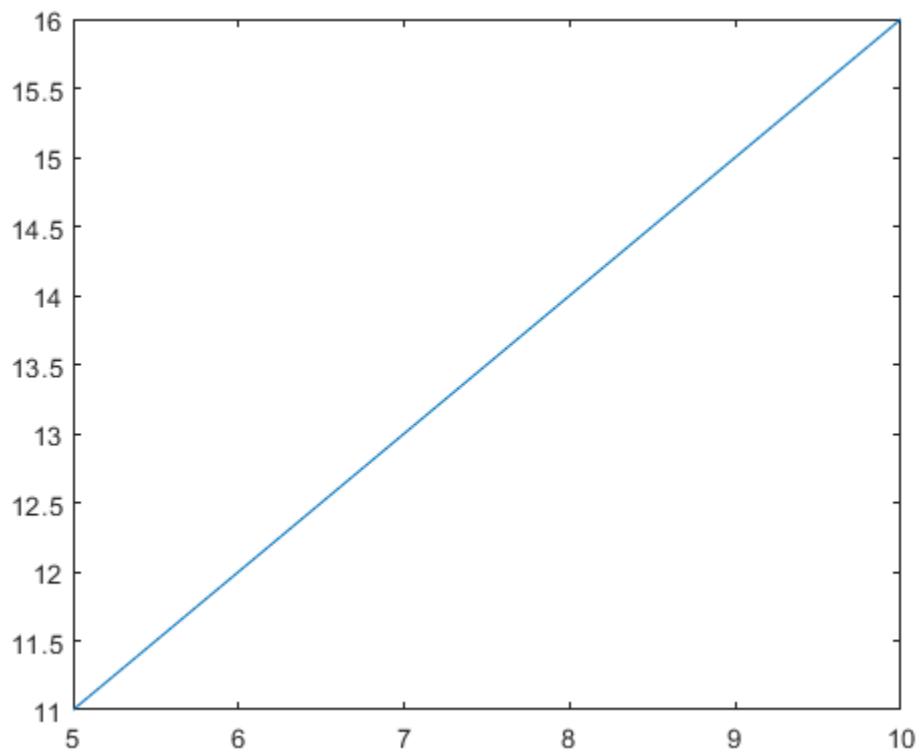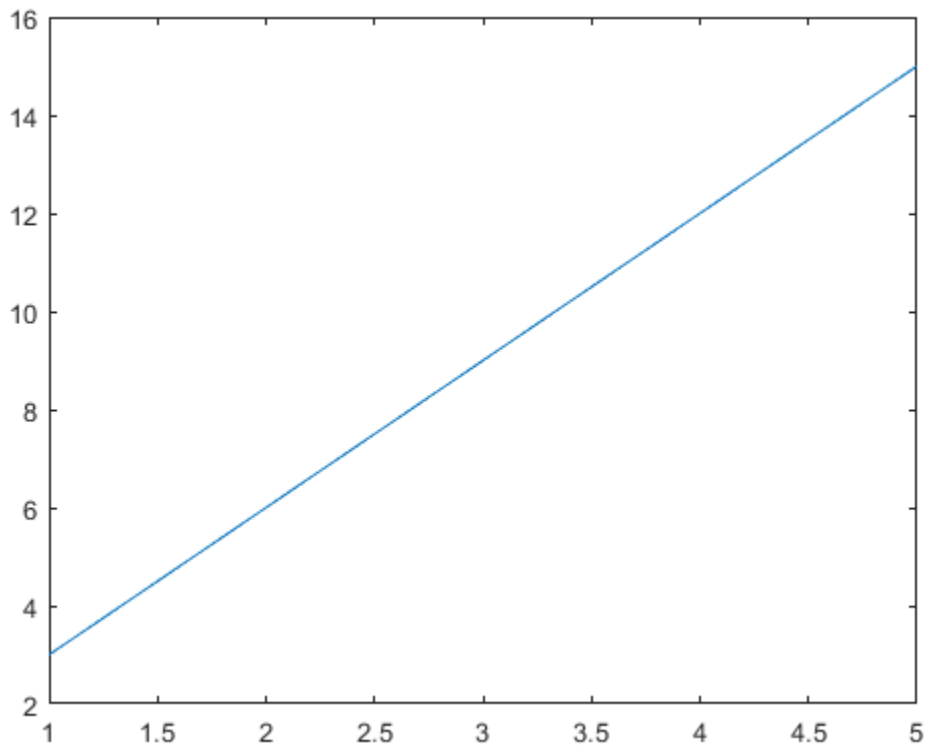
The above technique can be modified to make one variable dependent on the other. For example, if you have a set range of values (let's say x = 1,2,3,4,5 for this example) and another variable that is a multiple of x (say y = 3*x for this example) the following code could be used to plot the above example.

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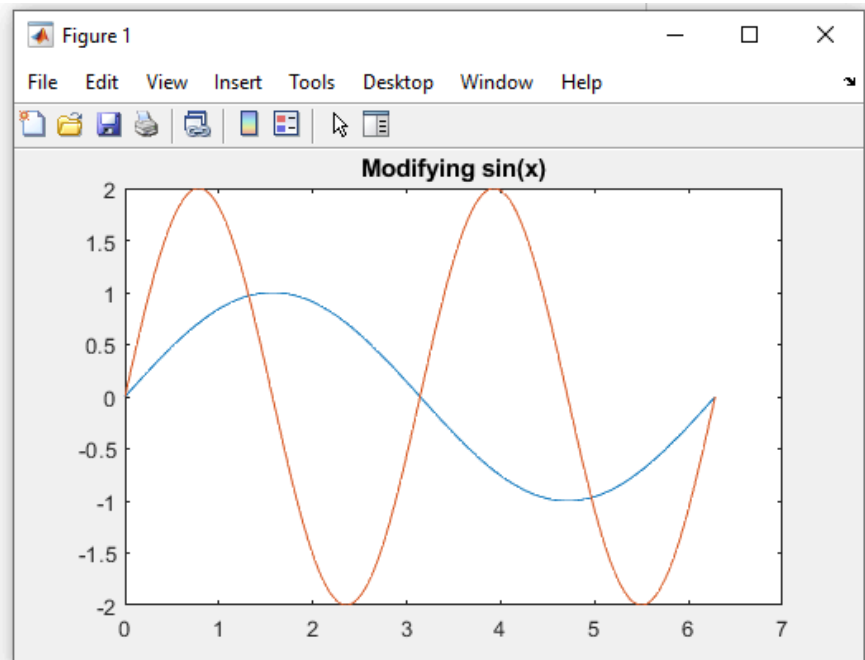
The following image contains concepts addressed above within an actual MATLAB code to create graphs like the one represented in the above examples of commands.

```
1 -    clear all
2 -    close all
3 -    clc
4
5 -    x = linspace(0,2*pi,200);
6 -    a = sin(x);
7 -    b = 2*sin(2*x);
8
9 -    plot(x,a)
10 -   title('Modifying sin(x)')
11 -   hold on
12 -   plot(x,b)
13
```


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

1 -    close all
2 -    clc
3
4 -    x = 1:5;
5 -    y = x.^2;
6 -    z = x.^3;
7 -    plot(x,y,x,z)
8
9 -    legend('x-squared','x-cubed')
10
11 -   xlabel('X Axis Label')
12 -   ylabel('Y Axis Label')
13 -   title('Title','FontSize',14)
14
```
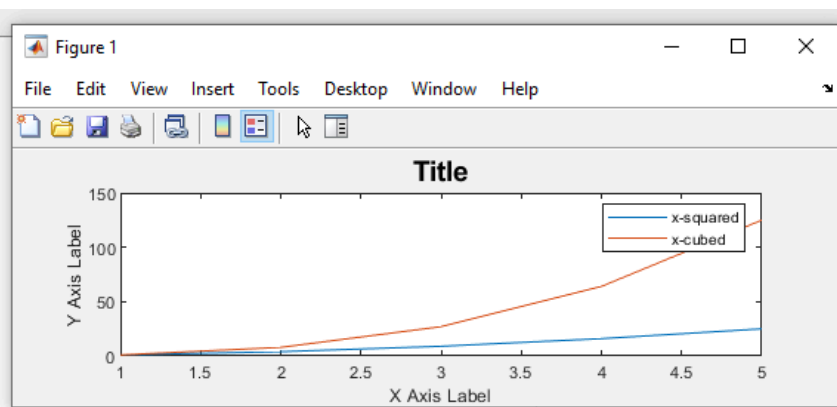

There are many methods available within MATLAB that can assist in producing graphs including scatter plots, line plots, or other non-linear display methods. To create a function that plots a scatter graph instead of a linear graph, use the function scatter in the place of the

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
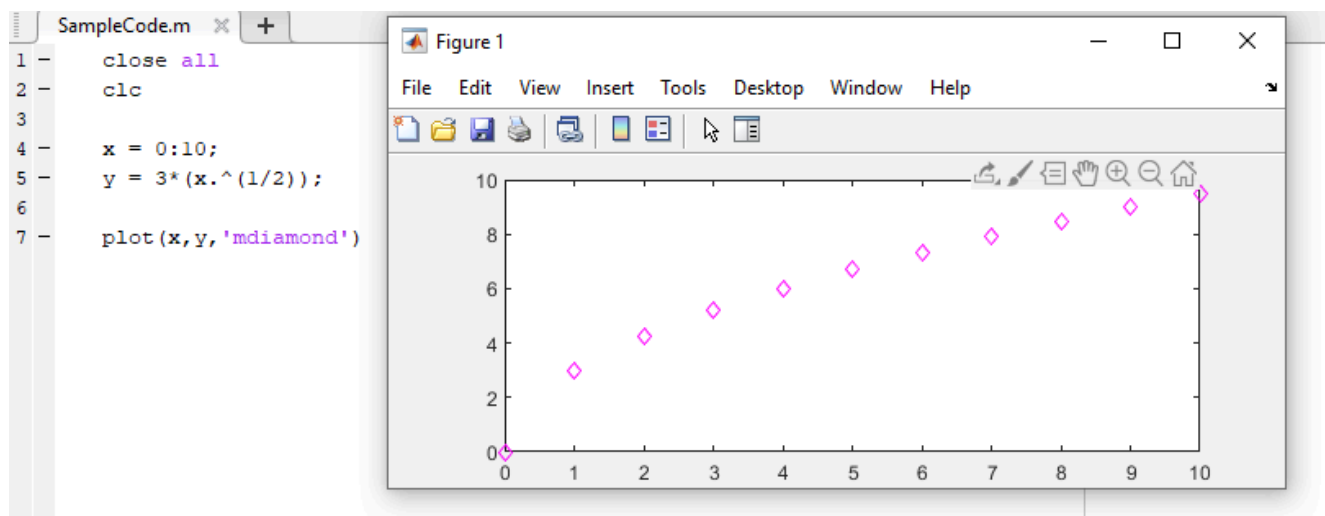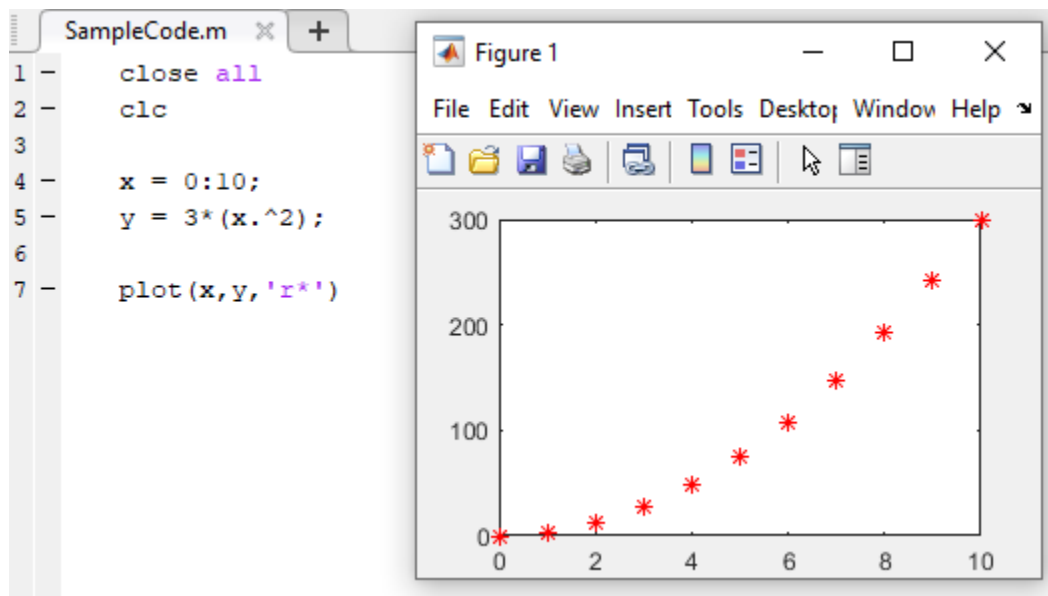
The Plot Tab

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.
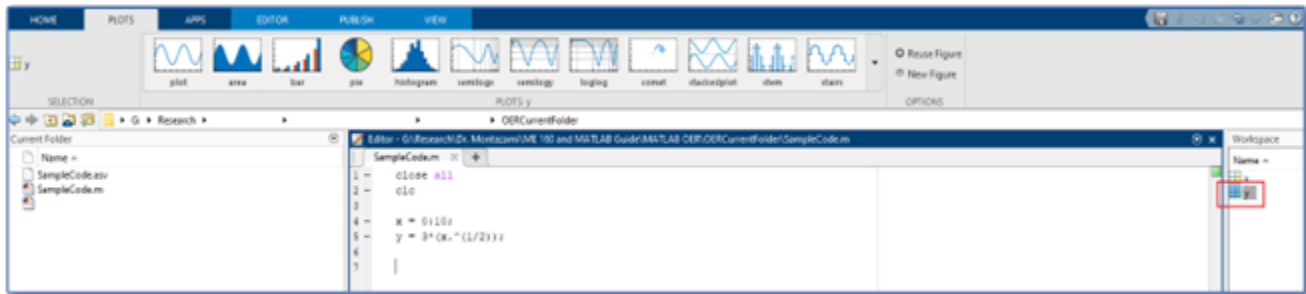
Examples

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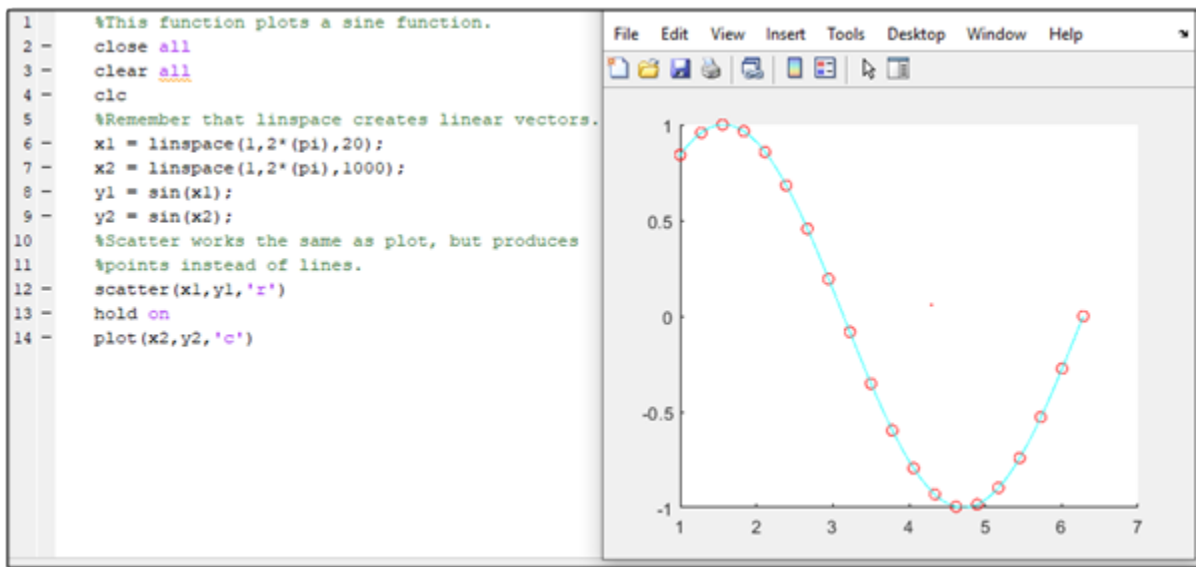
2) The following functions represent the motion of two projectiles. Plot the functions on the same graph. Use proper labels and ensure the graph is sufficiently legible for users. Use x-axis values from 0 to 20. Create a key and some title for the plot.

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

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

```
1 -    clear all
2 -    close all
3 -    clc
4
5 -    x = linspace(1,20);
6 -    y1 = -4*(x.^2)+24*x+100;
7 -    y2 = -6*(x.^2)+36*x+120;
8
9 -    plot(x,y1,x,y2)
10
11 -   title('Equation Comparison')
12 -   legend('Equation 1','Equation 2')
```
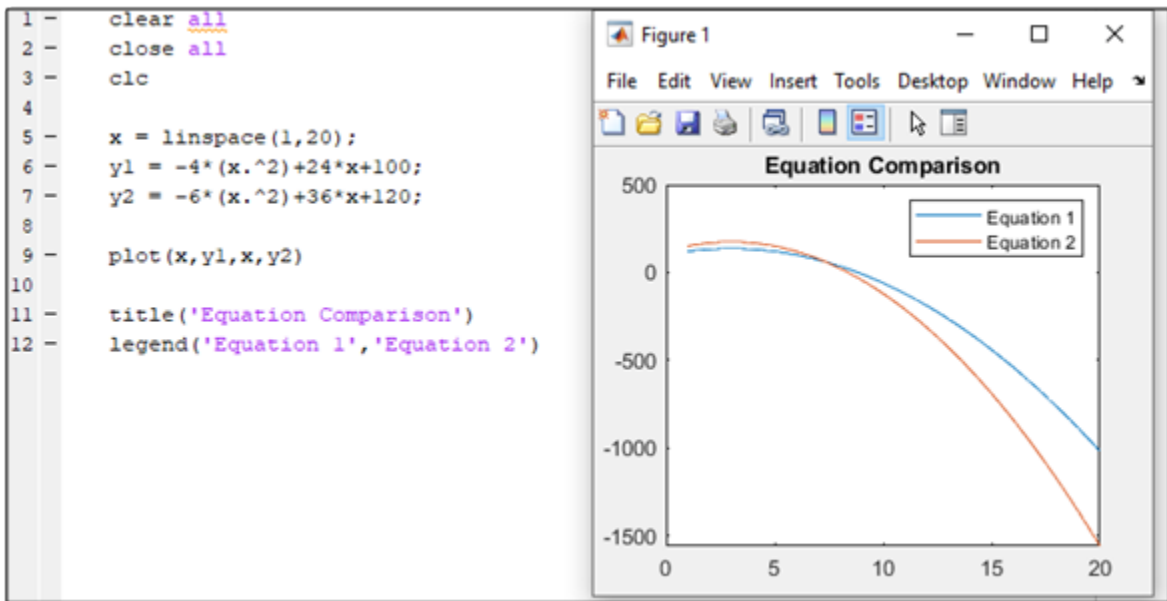

Problems

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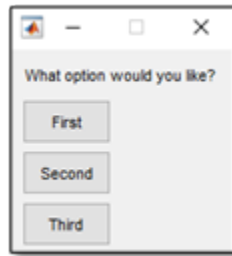
The `listdlg` syntax is similar to the older `menu` command. An example of the `listdlg` is shown below that outputs the same menu as the above example. First, a variable is created which is assigned an array of each list entry that will be in the menu. After the list is defined, the format of the list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

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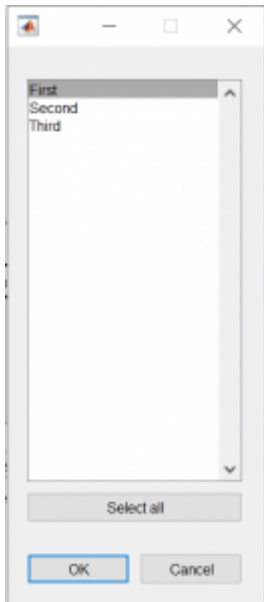

Adding options to the basic "ListString" selection enables further customization of the command. The option "PromptString" allows an instruction to be printed on the output as well. Additionally, the "SelectionMode" function allows for customization of the number of options that can be selected in the 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]);
```
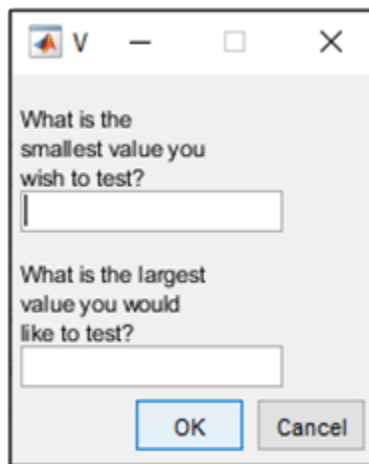
The Message Box (`msgbox`) Command

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');
```
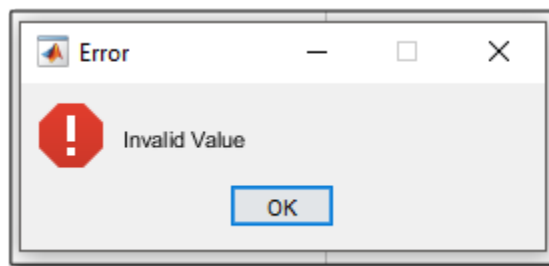

Examples

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.

```matlab
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
```

```matlab
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
13
```

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 ∆L 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 ∆L 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