A GUIDE TO MATLAB *for* ME 160

AUSTIN BRAY &
REZA MONTAZAMI

IOWA STATE UNIVERSITY DIGITAL PRESS

A GUIDE TO MATLAB FOR ME 160

AUSTIN BRAY AND REZA MONTAZAMI

Iowa State University Digital Press
Ames, Iowa

This is a publication of the
Iowa State University Digital Press
701 Morrill Rd, Ames, IA 50011
https://www.iatatedigitalpress.com
digipress@iastate.edu

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.

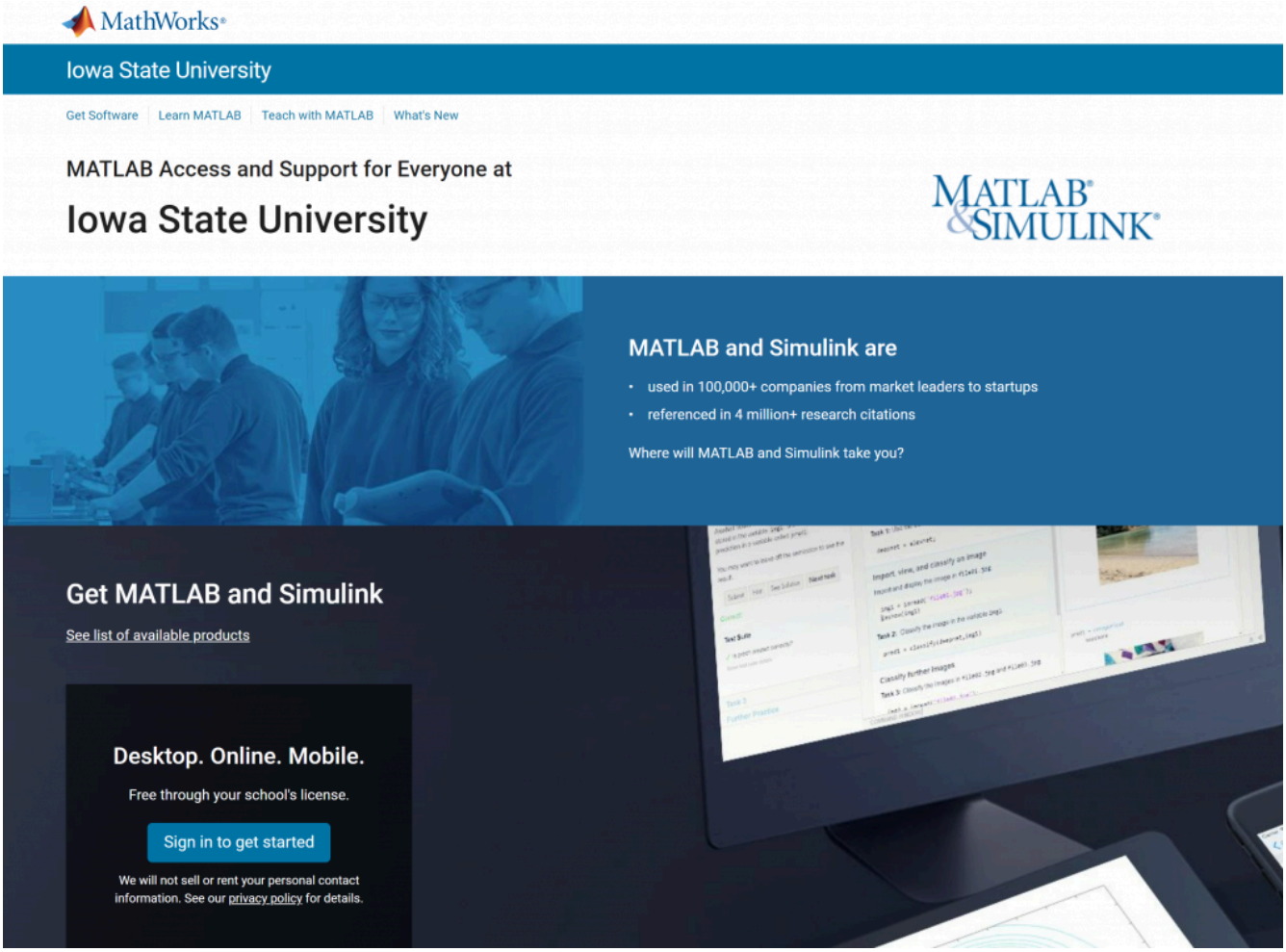
The following list contains examples of situations where MATLAB can be used as a problem-solving tool when working with topics addressed in ME 160. While the list is not exhaustive, it demonstrates a series of skills that the user should have by the completion of ME 160.

- 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.

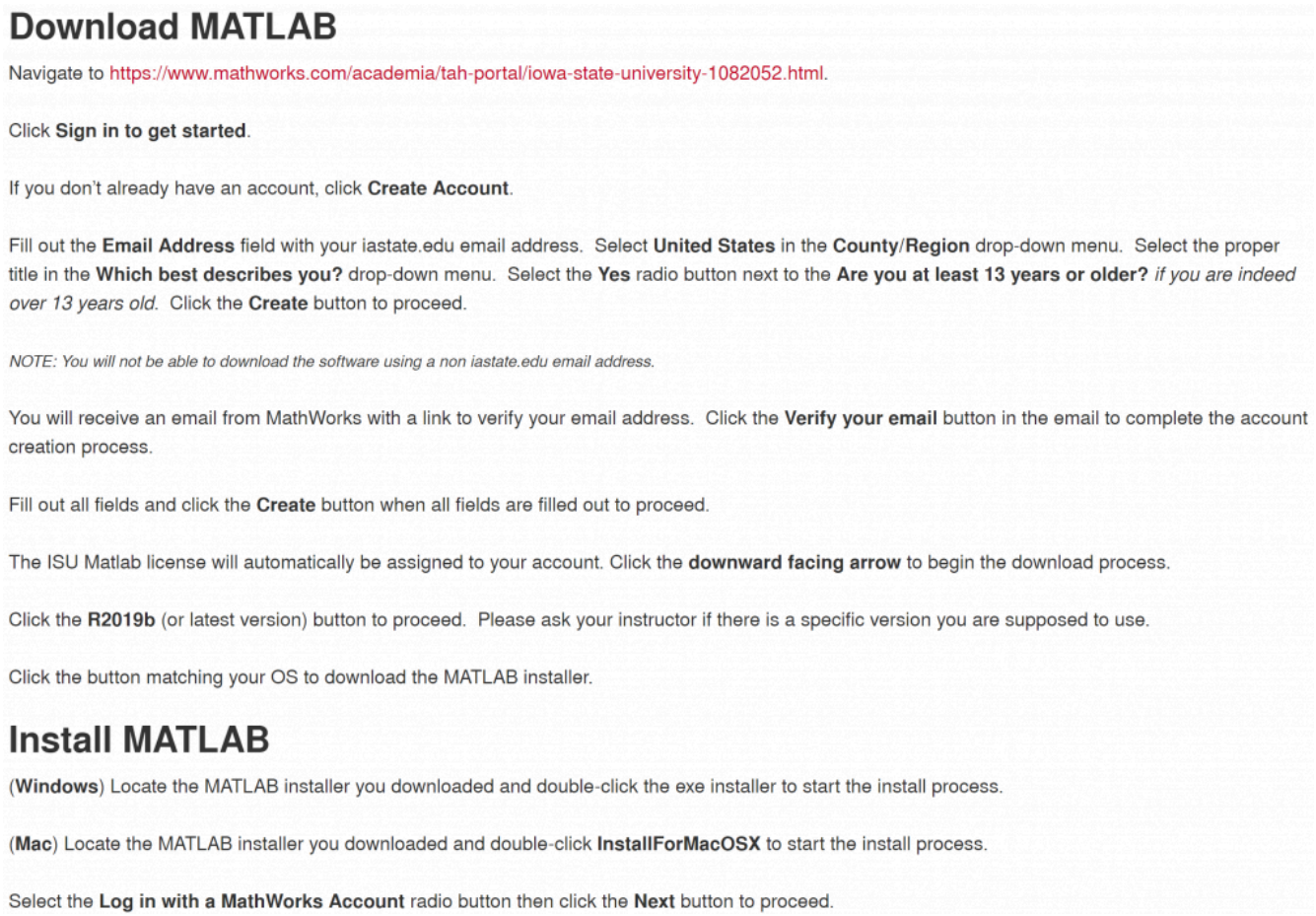
- Development of applications with graphical user interfaces.
- Create repeatable and sharable methods for solving engineering problems.

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.

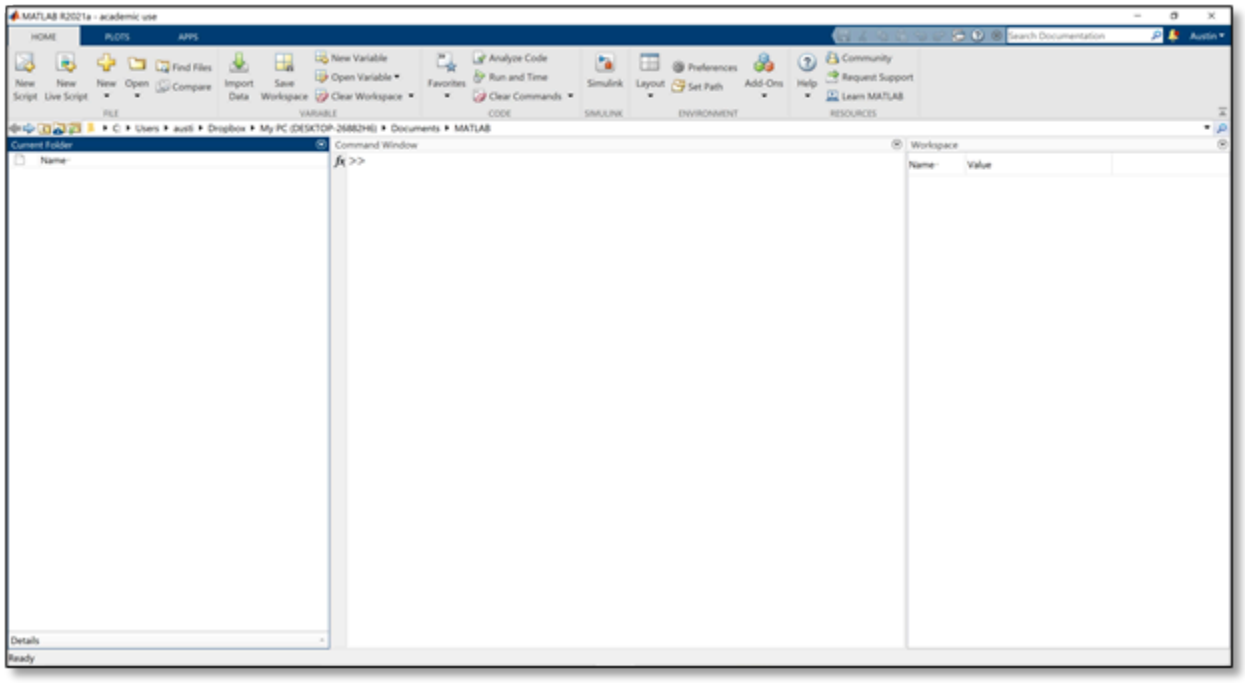
Installation instructions provided by Iowa State to download and install MATLAB

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

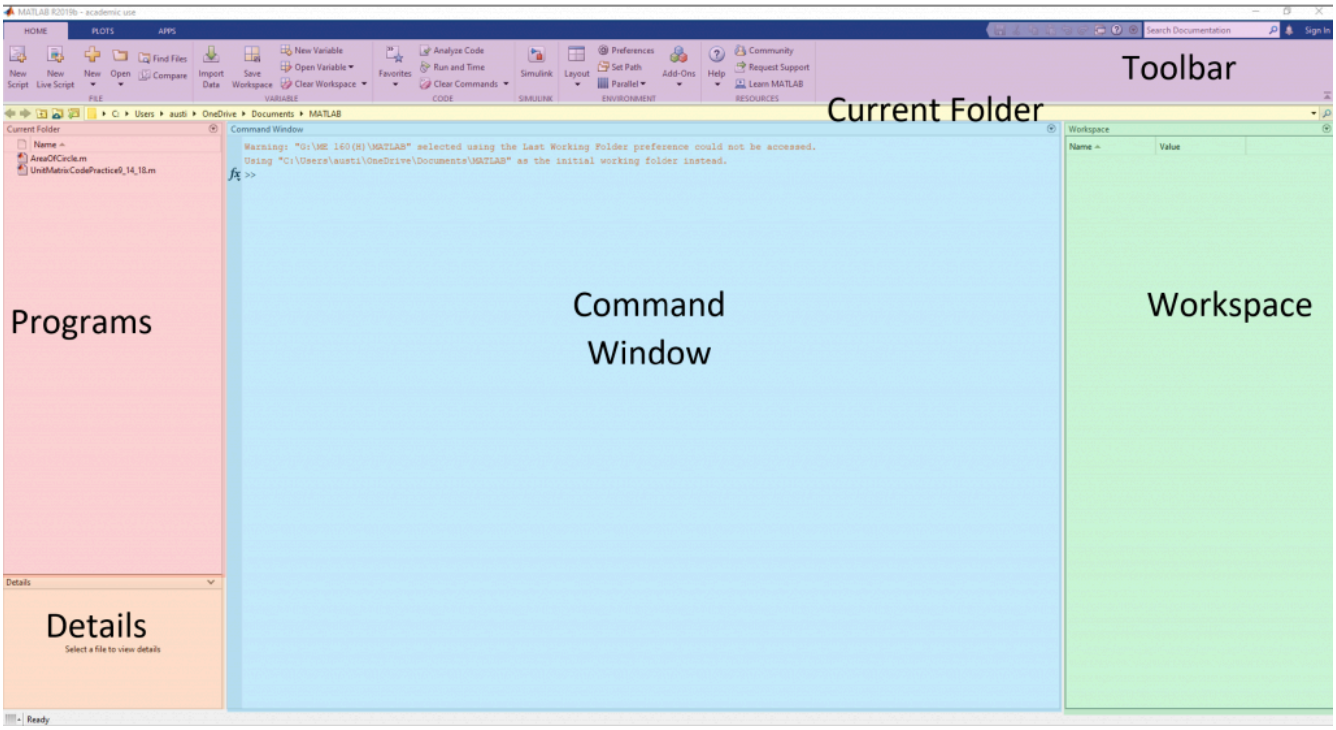
MATLAB is free and available on campus computers without requiring a download. Search for MATLAB in the start menu to locate the program. Note that it is good practice to ensure that the version of MATLAB available on campus computers is the same version used on any personal device. This can help prevent 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

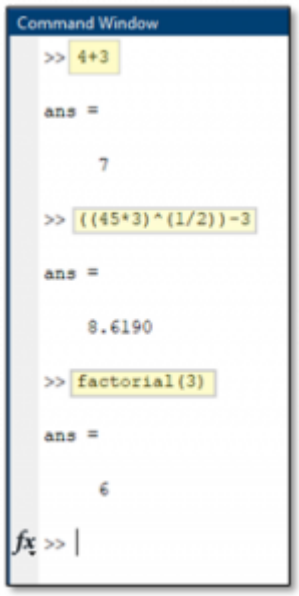
of the software. Upon opening the software in Windows, the user will encounter the following page, which is divided into several useful windows. Each window has been colored and labeled in the image below and is discussed in the following section. MATLAB enables the user to make modifications to the 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.

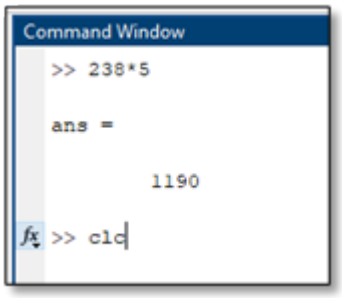


Each element of the MATLAB user interface upon startup. Elements are highlighted for clarity.

The Command Window

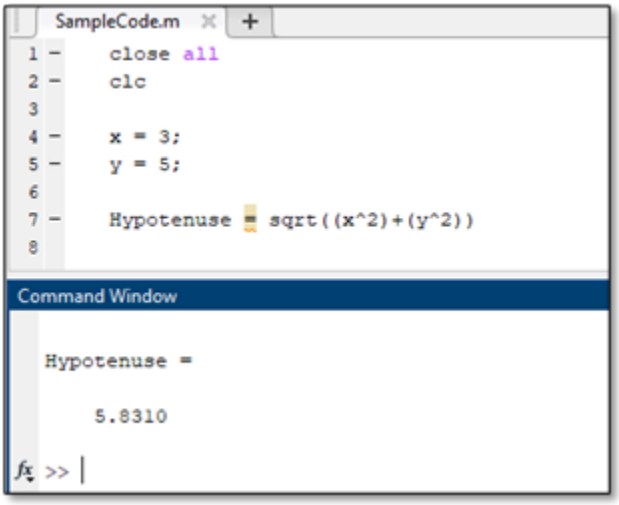
The command window (shown in blue) will serve as the primary interface the MATLAB user will interact with when writing and reading simple codes or individual operations. The command window can be used as an interface to type directly into MATLAB and as a place to view the results of these codes. As ME 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.





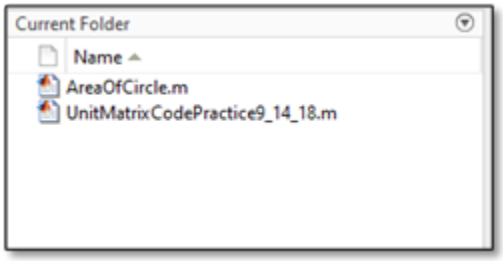
When the user is done viewing code or is restarting a program, the user must clear anything written in the command window. By using the clear command, `clc`, the user can reset the command window to its original blank state. For example, completing an arithmetic operation in MATLAB such as 238*5 will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



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.

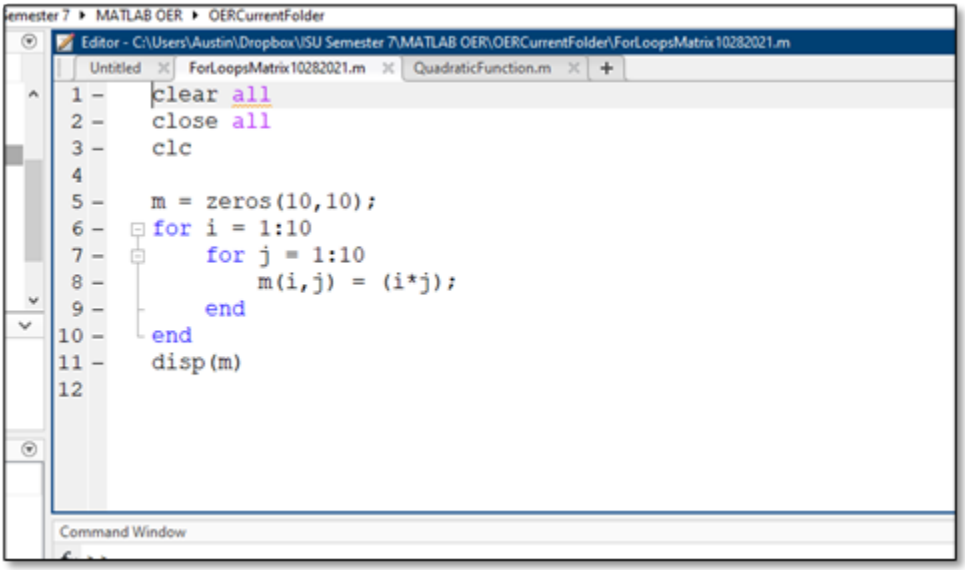


As you continue in ME 160 you will save many files in this section, which will quickly reveal

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

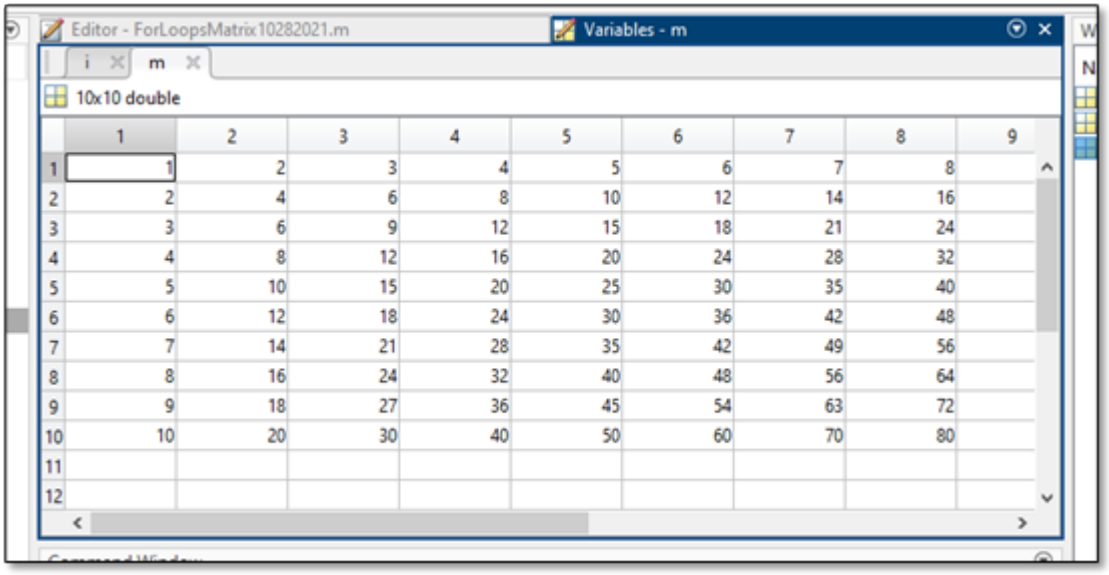
The editor window is the home of MATLAB script files- savable codes that can be shared and saved on a computer for use. MATLAB scripts are the files where all stand-alone programs that are written in ME 160 will be saved. When working in the command window every time an operation is typed and the enter 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.



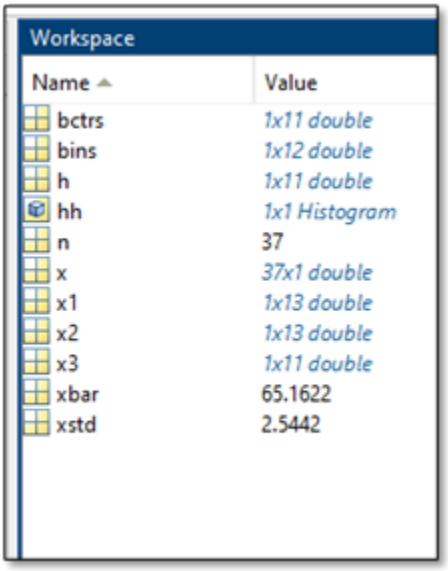
	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	
2	2	4	6	8	10	12	14	16	
3	3	6	9	12	15	18	21	24	
4	4	8	12	16	20	24	28	32	
5	5	10	15	20	25	30	35	40	
6	6	12	18	24	30	36	42	48	
7	7	14	21	28	35	42	49	56	
8	8	16	24	32	40	48	56	64	
9	9	18	27	36	45	54	63	72	
10	10	20	30	40	50	60	70	80	
11									
12									

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.



CHAPTER 2: BASIC COMMANDS IN MATLAB

Introduction

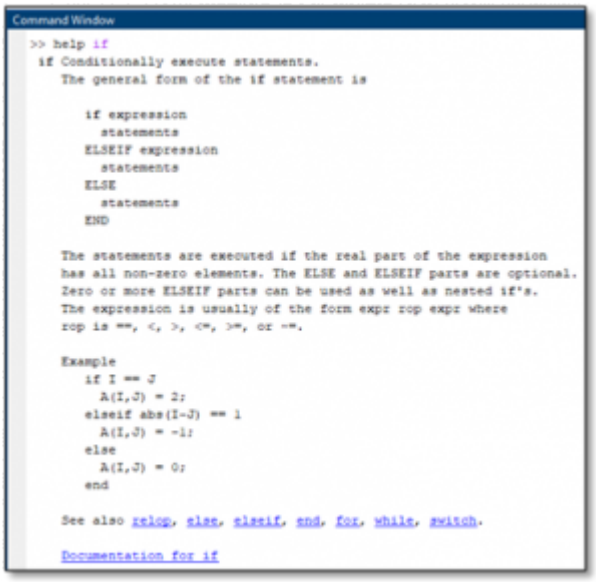
To get started writing code in MATLAB, several commands are essential to begin to operate the program. A command is a basic instruction that the user gives the program. These instructions, if given correctly, cause MATLAB to output a response such as printing a solution to a problem, executing a step 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.



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.

```
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.

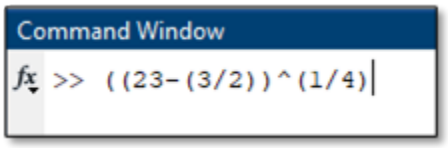
Complex Expressions

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

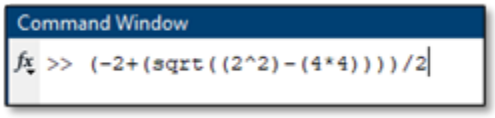
The following examples demonstrate how to input complex expressions into the command window.

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

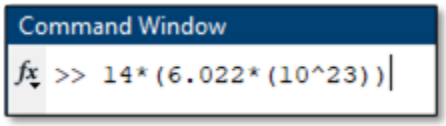


2. $\left[\frac{10\exp(3/2)}{\sqrt{23}}\right]$

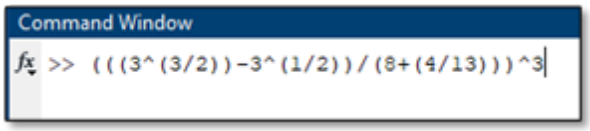
3. $\dfrac{-2+\sqrt{2^2-4(4)}}{2}$



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



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



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

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

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.

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

5. $a + b$

6. $(a + b) * c$

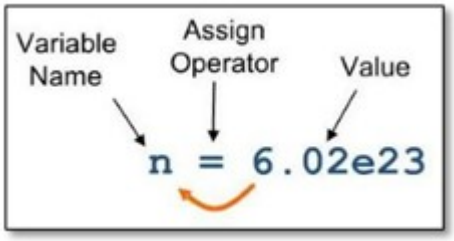
7. $(c * d)$

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

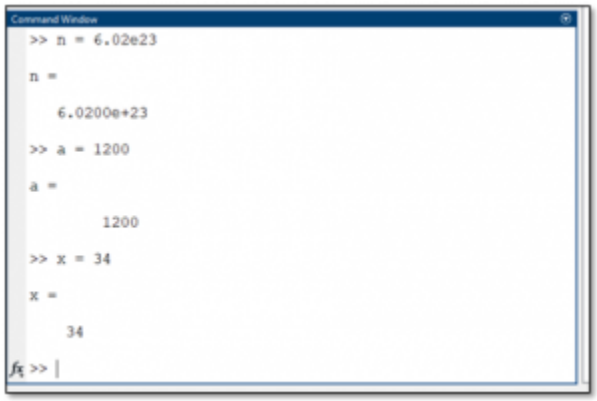
CHAPTER 4: WRITING SCRIPTS

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.



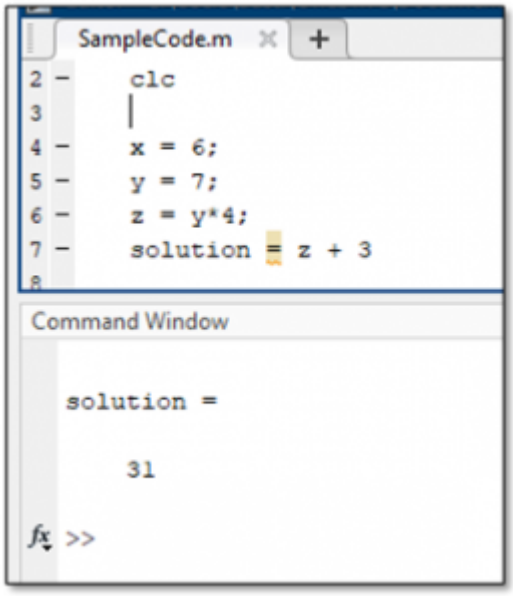
Variable named "n" being assigned the value of 6.02×10^{23}.

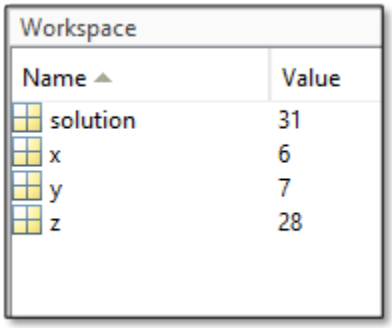


Example of variables being assigned in the command window.

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.

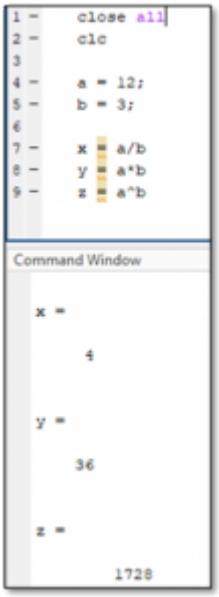




When creating equations within MATLAB codes, values assigned to variables can be checked to verify if equations are operating how the user intended them to. For example, the user can manually calculate the expected values generated by an equation and can check actual values displayed in the workspace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

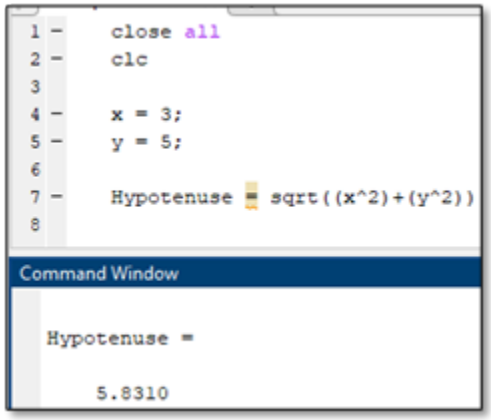
Assign the 

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.

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

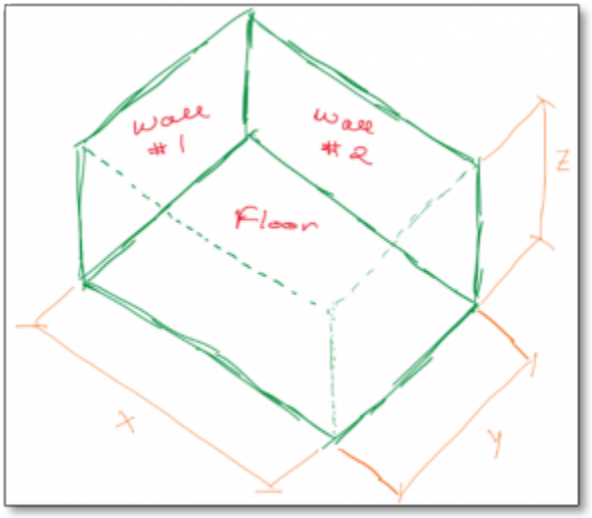


Problems

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 $(\frac{meters}{second^2})$, 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

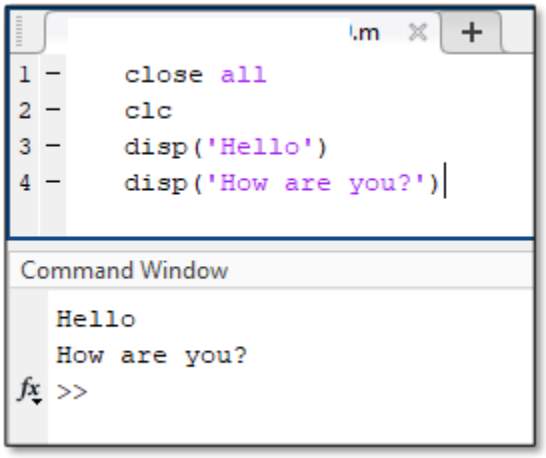
Commands will serve as tools that complete defined tasks within a MATLAB code. Often represented as puzzle pieces in introductory programming guides, commands enable the code to interact with the user, finish tasks in response to user inputs, or serve as a road map within the code. By piecing together 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.

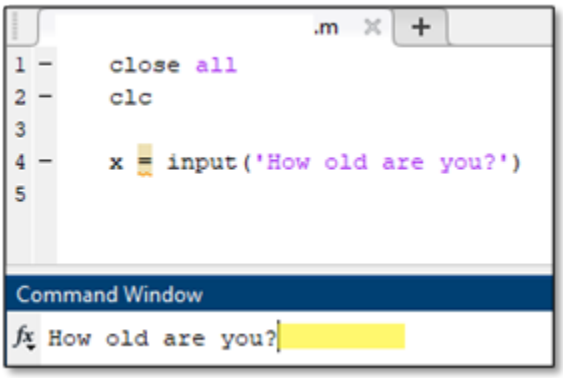
```
disp ('Message Here.')
```

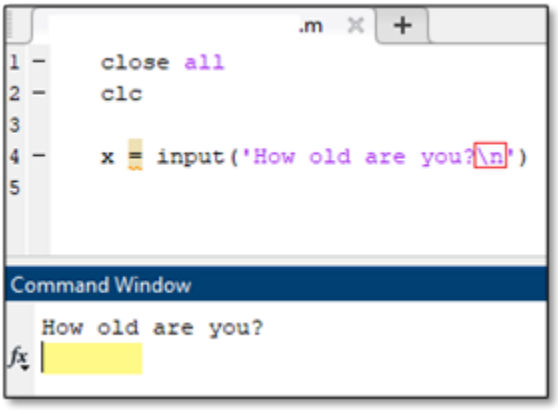


Note that the message being displayed cannot contain apostrophes, as the code will stop reading the message there. Additionally, it is important to understand that this command is not the best way to prompt users to input data or other information into the code, as the input command is best suited for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

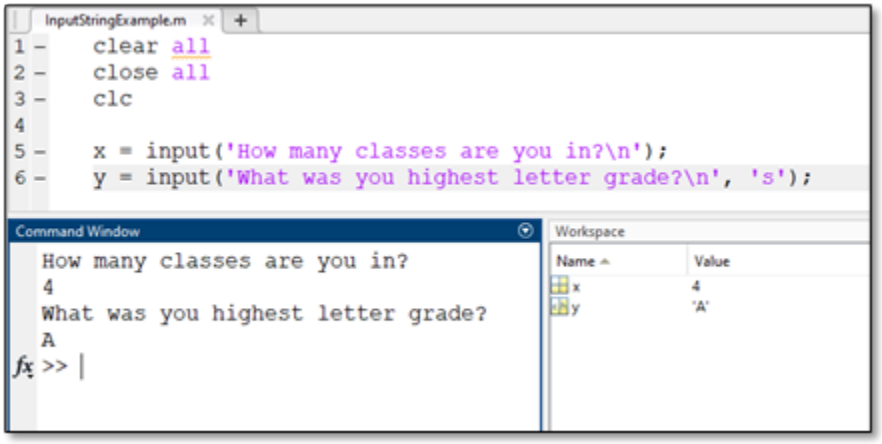




To improve the appearance of this function the user can add \n to the command, which will cause everything following the symbol to appear in the next line. If added to the end of the display command, as shown in the second example, the prompt to type an input appears in the line below the text in the 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')
```



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.

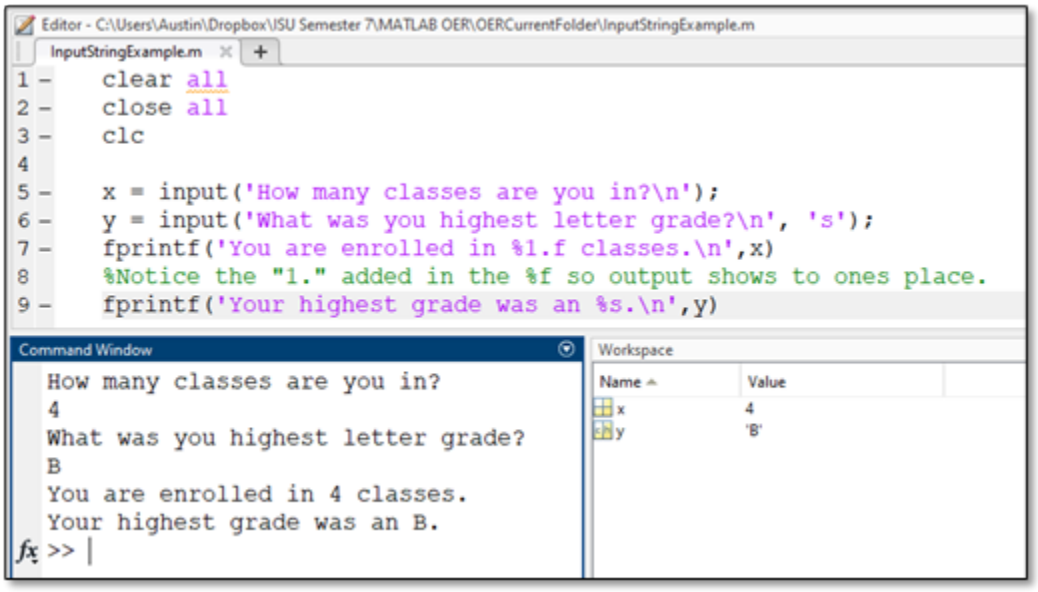
```
clc

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.



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