ECE 8 - Lab 1: Basic MATLAB Use (Part 1)

September 29 2022

1 Introduction

In this lab you will be introduced to the wonderful world of the MATrix LABoratory or MATLAB, a programming language and numerical computing environment built for technical engineering applications. MATLAB combines tools for computation, visualization, and programming all into a single environment allowing for agile engineering development.

1.1 Lab Objectives

By the end of this lab you should be able to:

1. Start and end a MATLAB session.
2. Use the `help` command in MATLAB to identify how various functions work.
3. Identify what a variable is and how to use one in MATLAB.
4. Use the `plot` command in MATLAB to create figures.
5. Use loops to write more efficient code.
6. Save and load MATLAB scripts and data.

1.2 Deliverables

1. A `.pdf` version of your `.m` file using the `publish` function. This is due October 7th, at 6:00 PM.

1.3 References and Background Reading

`https://www.mathworks.com/help/matlab/getting-started-with-matlab.html`
`https://www.mathworks.com/learn/tutorials/matlab-onramp.html`

1.4 Necessary Files, Tools, and Equipment

- MATLAB installed.

1.5 Grading Rubric

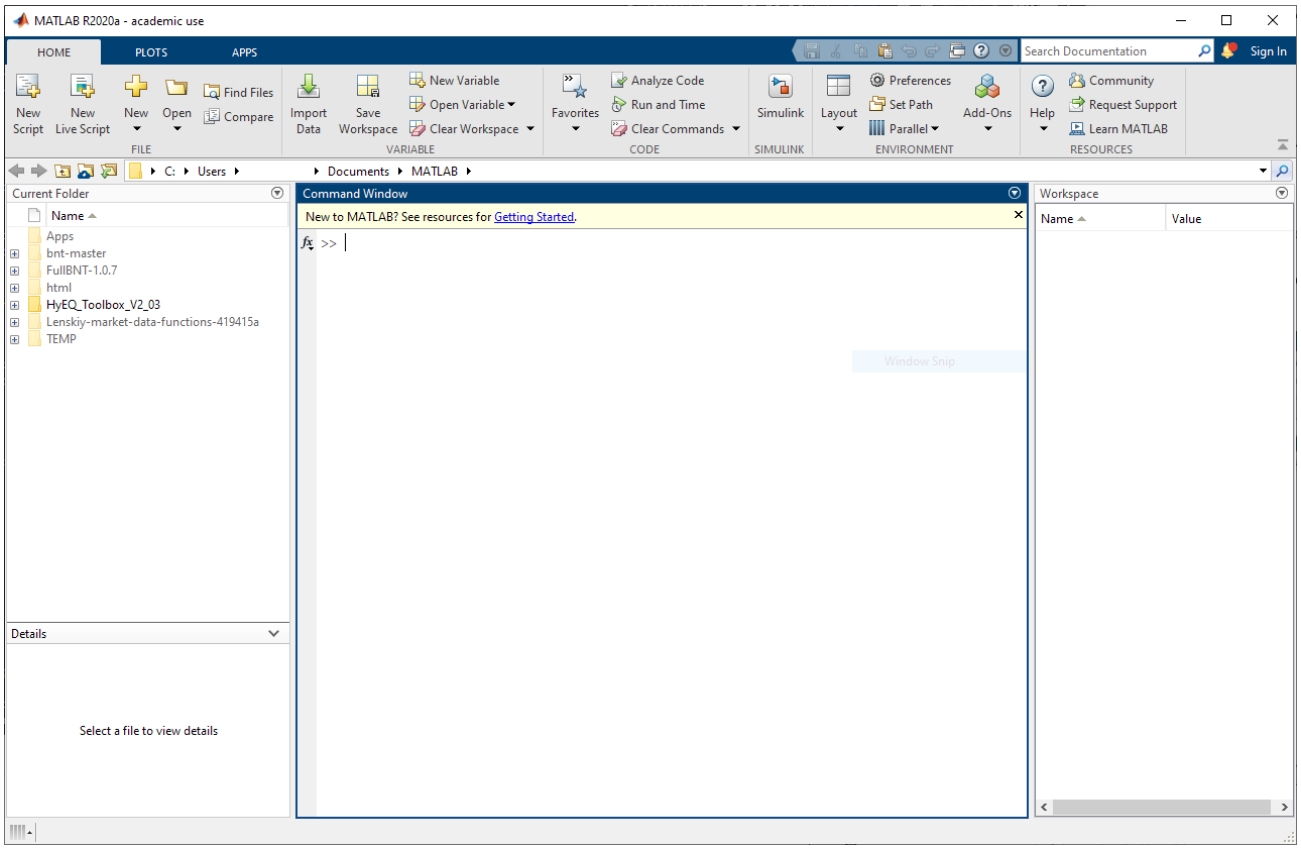
This assignment consists of 10 points.

- Exercise 1 is worth 2 points.
- Exercise 2 is worth 2 points.
- Exercise 3 is worth 3 points.
- Properly commented code is worth 1 point per exercise.

2 Getting Started with MATLAB

Starting a MATLAB session

Start by opening up MATLAB on your macOS or Windows machine. When MATLAB is loaded, a window with the MATLAB environment or desktop appears. The desktop window contains all of the available tools and associated windows to execute a MATLAB session. The default desktop should appear as follows



Within the window environment, you will find the following windows:

- **Command Window** - This window is where you can execute simple arithmetic expressions, MATLAB functions, and scripts.
- **Workspace** - This window contains the data from your current MATLAB session.

- **Current Folder** - This window contains the files and folders inside of your current working directory.
- **Details** - This window contains a preview of the contents of a highlighted file or folder from the **Current Folder** window.

Command Window

As previously mentioned, the command window is where you can execute simple arithmetic expressions, MATLAB functions, and scripts. As an example, let's try using MATLAB as a simple calculator. Suppose you want to calculate the expression $6 \times 3 + 7$, in the **Command Window** you would type `>> 6*3+7` and the result should be displayed in the window as follows,

```
>> 6*3+7
ans = 25
```

Creating your Working Directory

Creating your own working directory lets you keep all your work in one spot and enables you to easily backup your work (See Robot Automation handbook Section 2.2.1). MATLAB can only execute programs that are in its current working directory, so if you write a program but save it somewhere besides your working directory, MATLAB will not be able to find it and issue an error message.

The `help` Command

The `help` command is very useful for learning how various functions and commands work in MATLAB. While many functions in MATLAB work as you would expect them to, there are many subtleties that can trip you up. For example, `cos(x)` is universally used to calculate the cosine of an angle, however it is important to know if the angle should be given in radians or degrees. You can use the command `help cos` to learn how the cos function works in MATLAB as follows,

```
>> help cos
 cos    Cosine of argument in radians.
    cos(X) is the cosine of the elements of X.

    See also acos, cosd, cospi.

    Documentation for cos
    Other functions named cos
```

MATLAB data types

Variables

A variable in MATLAB is essentially a tag that you assign to a value while that value remains in memory. The tag gives you a way to reference the value in memory so that your programs can read it, operate on it, and save it back to memory. When creating variables it is important to remember that the expression on the right hand side of the equal sign is evaluated first and then assigned to the variable on the left hand side. For example, suppose you want to create the variable `xyz` and assign it the value 5. You would simply type:

```
xyz = 5
```

The expression on the right hand side of the equal sign can range from a single value (e.g. `x = 5`) to a complex equation (e.g. `z = 5*x + sin(y)`). You can also use the variable on the left hand side of the equal sign in the expression on the right hand side. For example:

```
x = 5+3
x = x*2
```

The first statement adds 5 plus 3 and assigns the value 8 to the variable `x`. The second statement multiplies the value of `x` (which is 8) by 2 and assigns the new value 16 to `x` overwriting the previous value.

Vectors and Matrices

Variables in MATLAB can also be assigned an array of values instead of just a single value. When an array has dimension $1 \times n$, i.e., in a *row* or of dimension $m \times 1$, i.e., in a *column*, the variable is, respectively, referred to as a *row* or *column* **vector**. In order to create either vector, an array of values enclosed by square brackets is listed on the right hand side of the equal sign. For example, if a *row* vector is desired, you can assign the values 1, 2, 3 and 4 to the variable `y` by typing:

```
y = [1 2 3 4]
```

alternatively, if a *column* vector is desired we type

```
y = [1; 2; 3; 4;]
```

One can always transpose a vector from *row* to *column* or vice versa by including a `'` after the second bracket, so if we type

```
y = [1 2 3 4]'
```

`y` will be defined as a *column* vector.

Another way to create a vector is to use vector notation to create an array of values. In vector notation, an array of values is defined by a starting point, an increment size, and an ending point separated by colons. For example, to assign the values 1, 2, 3 and 4 to the variable y using vector notation you would type:

```
y = 1:1:4
```

The first `1` indicates the start of the vector, the second `1` indicates the increment size, and the `4` indicates the end of the vector. So in the example above, the variable `y` is assigned the value `1` along with every integer up to the value `4` in order.

When an array of values has dimension $m \times n$ we refer to such a variable as a **matrix**. To create a matrix we begin with a square bracket (`[`), separate elements in a row with spaces or commas (`,`), use a semicolon (`;`) to separate rows, and then end the matrix with another square bracket (`]`. As an example, if we wanted to create the following matrix

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

we would type

```
A = [1, 2, 3; 4, 5, 6; 7, 8, 9]
```

When a matrix is stored and remembered in the *Workspace*, we can refer to it simply using the assigned variable name. Thus for the above matrix, it will be referred to as matrix `A`. If we wanted to view a particular element in the matrix, we can specifying its location by typing

```
>> A(2,1)
ans = 4
```

A_{21} is an element located in the second row and first column. Its value is 4.

Scripts and publishing

Scripts

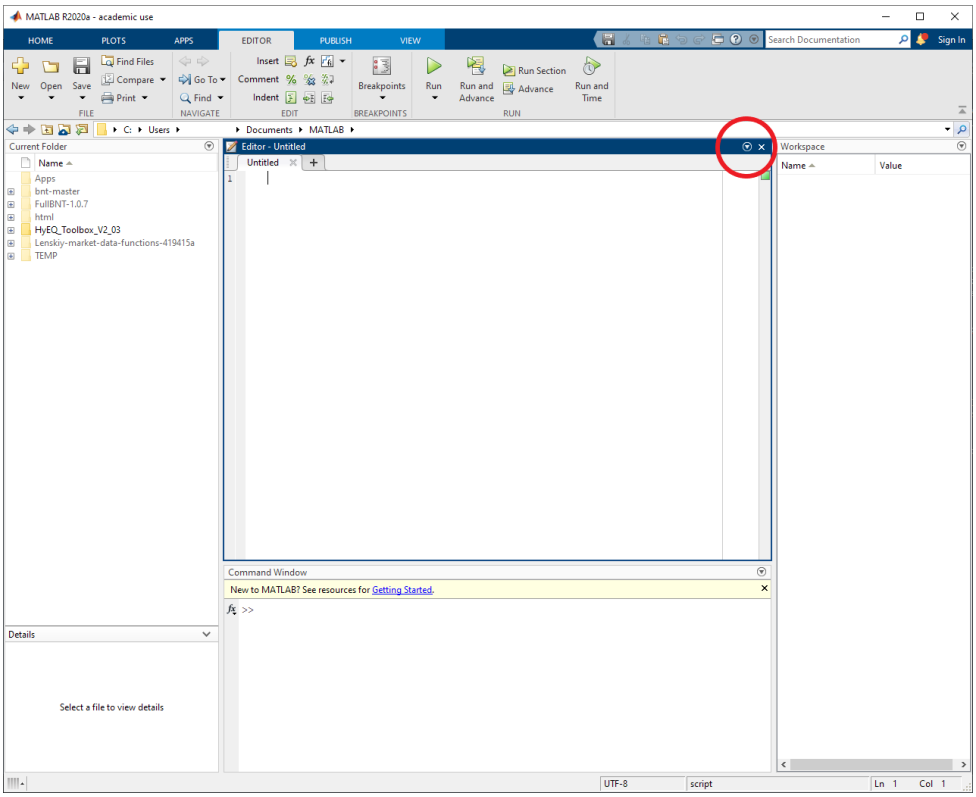
Suppose we want to execute a number of statements at once but don't want to type them into the command line individually, such as

$$5 \times 32 + 79 + 3 = ?$$
$$89/3 + 37 \times 2 = ?$$
$$1 + 12 + 123 + 1234 = ?$$

Suppose further that we want save these statements so that we can execute them again at a later point in time. Fortunately, we can utilize something called a *script* that allows us to both execute and save such statements.

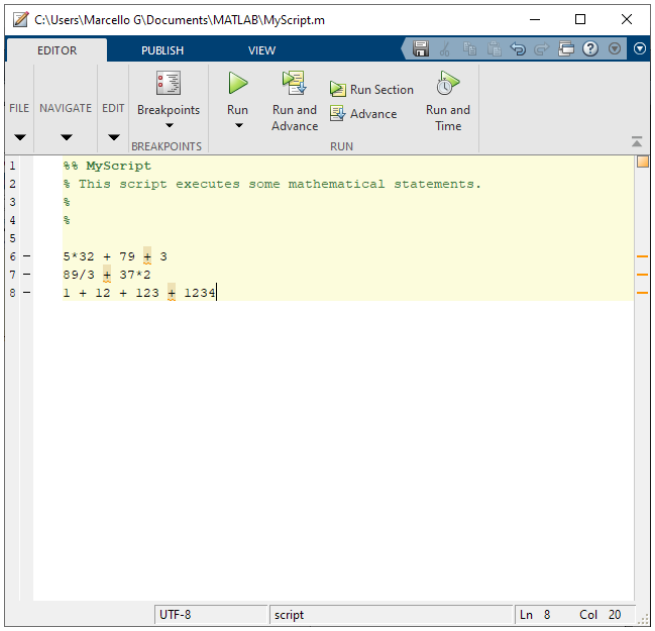
A *script* is a special type of file used in scripting languages, such as MATLAB, that contains a sequence of computational statements or instructions. In MATLAB, scripts use the filename extension `.m` and are often referred to as "M-files." M-files can be scripts that simply execute a series of MATLAB statements, or they can be functions that can accept arguments and can produce one or more outputs.

To create a script click the 'New' button marked by a '+' sign in the upper left-hand corner of the MATLAB session window or press "Ctrl + n" on your keyboard. Your MATLAB session should now appear as follows.

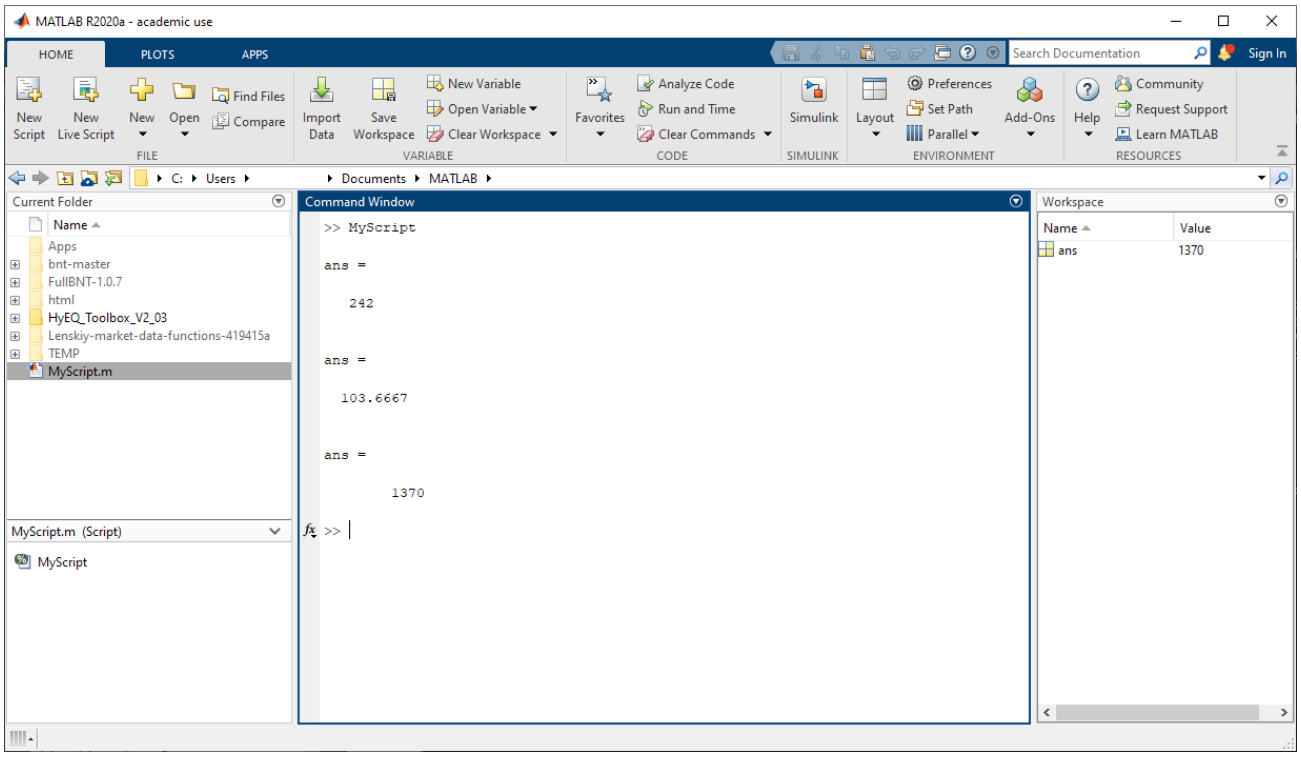


NOTE: You can 'undock' the editor from the main window, by clicking the drop down menu arrow in the upper right-hand corner of the editor window (as highlighted by the red circle in the above image).

Then we can type in the above mathematical statements as follows (note that the script has been undocked from the main window). Whenever you write code, it is a good practice to add comments that describe the code. Comments enable others to understand your code and can refresh your memory when you return to it later. Add comments using the percent (%) symbol and use the percent symbol twice (%%) when you want give it a header comment or to partition the script into sections.

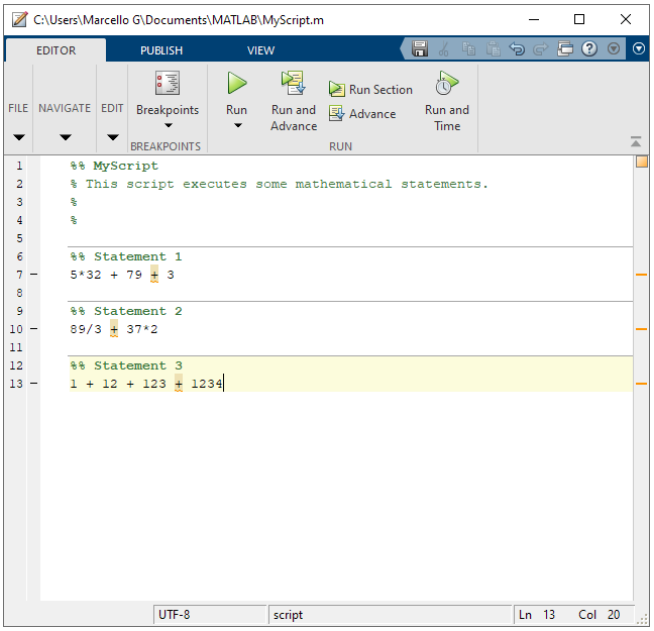


Then to execute the script we simply click the 'Run' button marked by the green play symbol or we can press "F5" which yields the following output from the command window

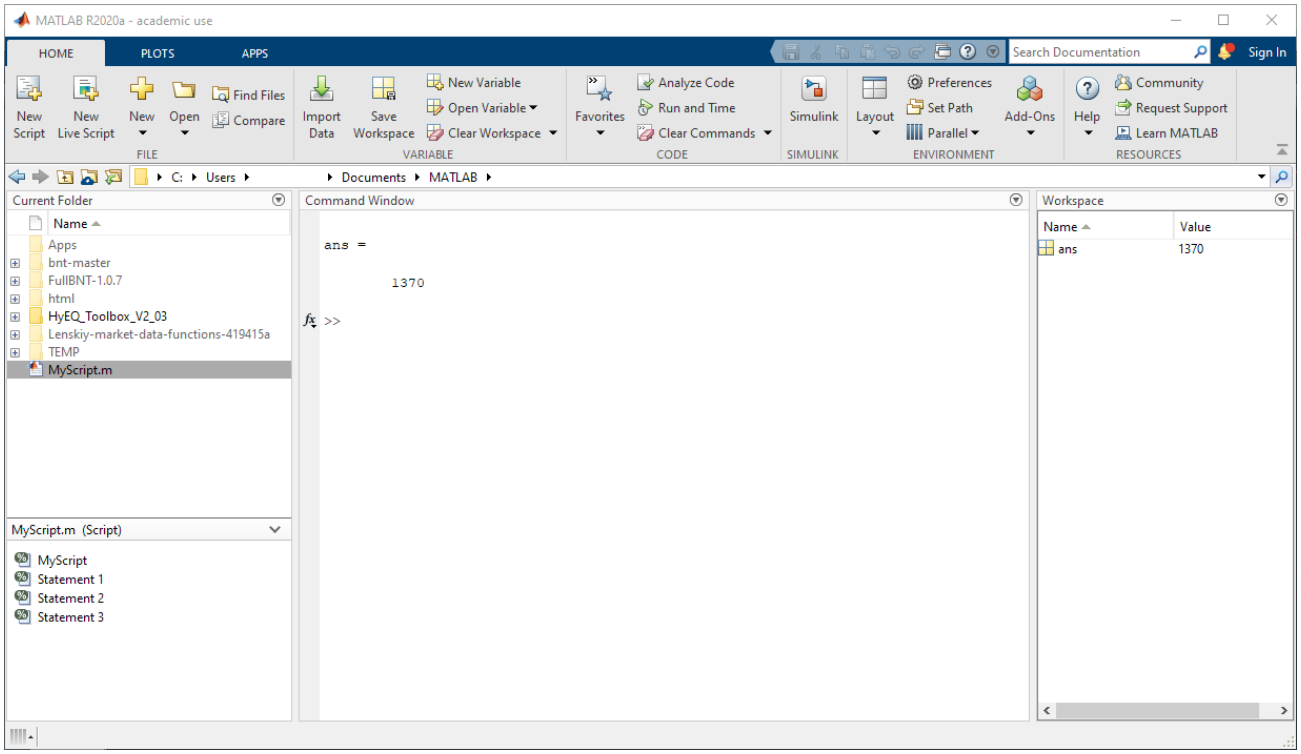


Script partitioning

As previously mentioned, we can use the percent symbol twice (%%) to partition our scripts this allows us to execute sections of the script separately rather than having to execute the entire script. This is especially helpful when we have very long scripts where we are only interested in a piece of the code. Using our previous script as an example, we can partition each statement as follows



Then to run each individual section, we simply click on the section we're interested in running such that section is highlighted in yellow as shown above and then click the "Run Section" button to the right of the "Run and Advance" button. Thus if we click the section titled "Statment 3", our output should be as follows



The `publish` function

With our script organized in this fashion, we can generate a .pdf or .html file in order to share with others. So if we have a script in our working directory called 'MyScript.m' we can simply execute `publish('MyScript.m','pdf')` in the command window where 'MyScript.m' indicates the script name and 'pdf' indicates the desired file format.

MATLAB functions and constant values

MATLAB offers many predefined mathematical functions for technical computing which contains a large set of mathematical functions. Typing `help elfun` and `help specfun` calls up full lists of elementary and special functions respectively. There is a long list of mathematical functions that are built into MATLAB. These functions are called built-ins. Many standard mathematical functions, such as $sin(x)$, $cos(x)$, $tan(x)$, e^x, $ln(x)$, are evaluated by the functions `sin`, `cos`, `tan`, `exp`, and `log` respectively in MATLAB.

In addition to the elementary functions, MATLAB includes a number of predefined constant values such as π given as `pi`.

The `plot` Command

Plotting figures in MATLAB is easy using the `plot` command. This command allows you to plot everything from very simple equations (e.g. $x = 2 * y$) to more complex equations (e.g. $z = 5 * x^2 + 12 * sin(y)$) and also allows you to plot multiple curves on the same figure. Once you are familiar with

how to use some elementary functions, plotting them will help you visualize your answer and better understand the effects of different variables on those functions.

3 Exercises

Write a single MATLAB script to complete the following exercises, separate each exercise using appropriate MATLAB comments. Upon completion of your script use the `publish()` function to generate a .pdf of your script for file submission.

1. Calculate the following expressions (pay attention to the dimensions of the matrices and vectors!)

   (a) $AB + C$

   (b) $Ax + Bu$

   (c) $\frac{1}{3}A(x + u)$

   where $A = \begin{bmatrix} 8 & 1 & 6 \\ 3 & 5 & 7 \\ 4 & 9 & 2 \end{bmatrix}$, $B = \begin{bmatrix} 29 & 44 & 86 \\ 1 & 66 & 37 \\ 84 & 78 & 5 \end{bmatrix}$, $C = \begin{bmatrix} 7 & 3 & 9 \\ 6 & 3 & 5 \\ 7 & 6 & 5 \end{bmatrix}$, $x = \begin{bmatrix} 2 \\ 7 \\ 9 \end{bmatrix}$, and $u = \begin{bmatrix} 9 \\ 38 \\ 45 \end{bmatrix}$

2. For the following functions and the given domains, you will produce four figures: one plot for each function defined in a., b., and c., and another plot with all of the functions plotted together on the same figure

   a. $y = 0.1x$ where `x = 0:2:100`

   b. y = sin(x) where `x = 0:0.1:2*pi`

   c. y = cos(x) where `x = 0:0.1:2*pi`

3. We have an experiment involving a car that is aware of its own position and orientation on a 2D plane. Its goal is to reach a point on the same plane – which we call **the target**. It does this by using a *wonky* controller (it tries to move efficiently as possible to the point, but can overshoot or undershoot). You can see an example of the experiment here.

   The experiment has already been completed and you are asked to produce one figure with the following information:

   (a) **plot of the initial location** with your favorite marker;

   (b) **plot of the vehicle position** using your favorite color and appropriate line style; and

   (c) **plot of the target** with your favorite marker.

   All of these plots need to be on the same figure including labels on the axes and a legend. The data that needs to be plotted is provided through Canvas. The units of the experiment are in 'meters'. The target sits at the $(2m, 0m)$ point.

Note: It's very IMPORTANT to comment your code. Please make sure to include a brief explanation of the code used to complete each of the exercises.