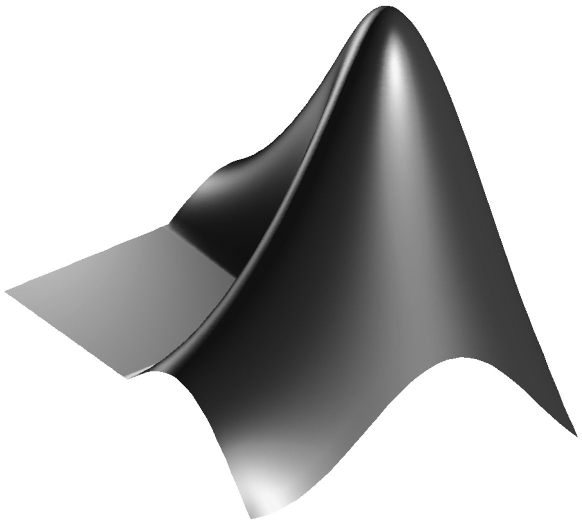
An Introduction to Numerical Methods

Using MATLAB®

Khyruddin Akbar Ansari, Ph.D., P.E.
Bonni Dichone, Ph.D.

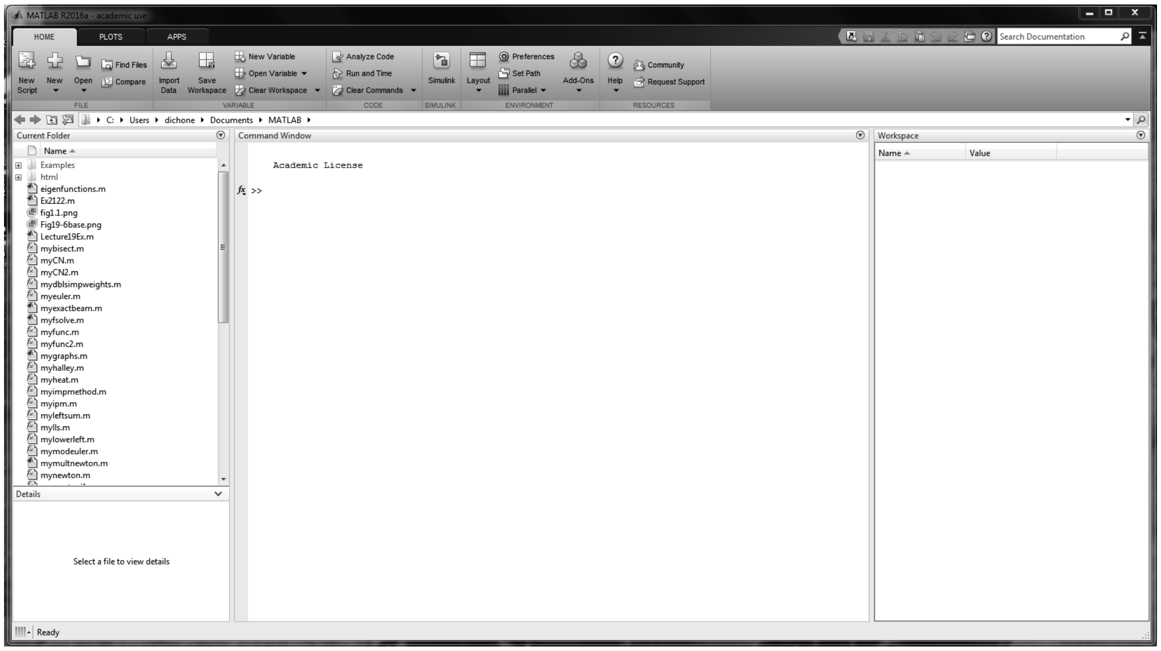
SDC PUBLICATIONS
Better Textbooks. Lower Prices.
www.SDCpublications.com

Chapter 1
Basics of MATLAB



1.1 Introduction

MATLAB is a programming language developed by MathWorks. It is primarily used for mathematical operations involving matrices and arrays. With MATLAB, we can plot functions and data, implement algorithms, and create iterative processes. In this text, we will use MATLAB primarily for numerical computing; however MATLAB does have some symbolic computing ability which we will use lightly.



```
Academic License
>>
```

Fig. 1.1: MATLAB Screen

1.2 The MATLAB Screen

When you first open MATLAB, you will see some version of the screen shown in Figure 1.1. We will begin by entering commands in the Command Window.

1.3 Entering Vectors

In MATLAB, data is typically entered as a matrix, or array of numbers. Vectors are examples of matrices with only one row or one column. A row vector with n elements has dimension $1 \times n$ and can be entered as follows:

```
v = [1 2 3 4 5 6]
```

When you hit "enter", MATLAB will print out the vector in the Command Window:

```
v =
     1     2     3     4     5     6
```

Notice that we separate the elements of a row with spaces. You can also separate the elements with commas.

A column vector with m elements has dimension $m \times 1$ and can be entered as follows:

```
u = [7; 8; 9; 10]

u =
     7
     8
     9
    10
```

Notice that we separate the elements of a column with semicolons.

You can access an entry in a vector with:

```
u(3)

ans =
     9
```

You can change the value of an entry with:

```
u(3) = -14

u =
     7
     8
   -14
    10
```

You can extract a “slice” of a vector with:

```
u(2:4)

ans =
     8
   -14
    10
```

You can transpose a row vector into a column vector and a column vector into a row vector:

```
w = v'

w =
     1
     2
     3
     4
     5
     6

z = u'

z =
     7     8   -14    10
```

There are shortcuts to create vectors whose elements are a set distance away from each other:

```
x = -2:.1:-1

x =
   -2.0000   -1.9000   -1.8000   -1.7000   -1.6000   -1.5000
         -1.4000   -1.3000   -1.2000   -1.1000   -1.0000
```

For this definition of a vector, the first number is the starting element, the middle number is the step size between elements, and the last number is the ending element.

Also:

```
y = linspace(0,1,9)

y =
    0    0.1250    0.2500    0.3750    0.5000    0.6250
       0.7500    0.8750    1.0000
```

In the *linspace()* command, the first input is the starting element, the second input is the ending element, and the last input is the desired number of elements in the vector. MATLAB will then use an appropriate step size to evenly divide the given range into the given number of elements.

1.4 Plotting Data

Let us enter some data:

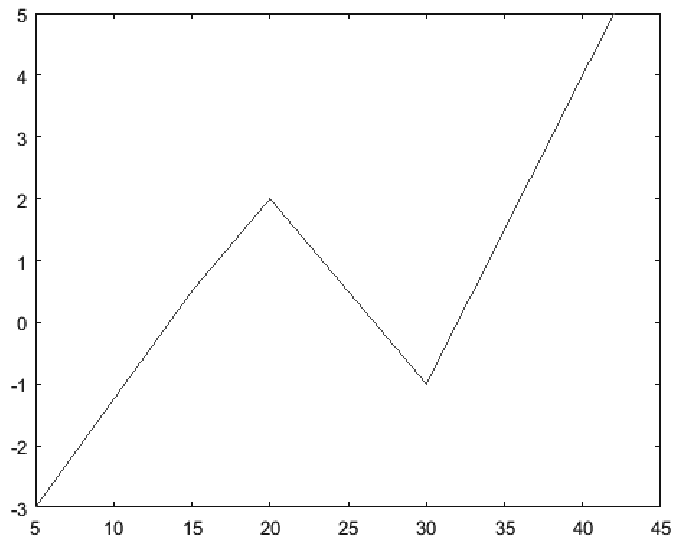
```
x = [5 15 20 30 42];
y = [-3 0.5 2 -1 5];
```

Notice that by putting a semicolon after the definition of our vectors, MATLAB doesn't print out the vectors.

To plot the data:

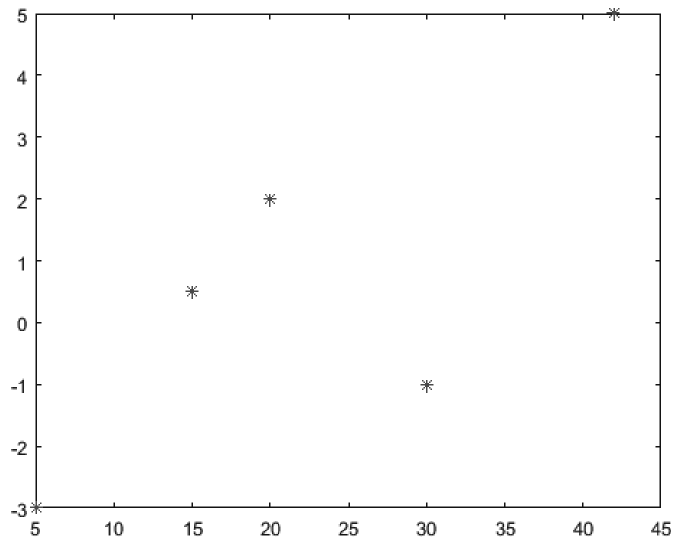
```
plot(x,y)
```

Which produces the following figure:



Notice MATLAB produces a graph with the data connected by lines. This is not always the best way to represent discrete data, so you can plot the data as individual points, represented by symbols:

```
plot(x,y,'*')
```



There are other symbols that can be used, like ’o’, ’.’, ’d’ and others. For a complete list and explanation, you can enter:

```
help plot
```

which will display an explanation of the command *plot* and its inputs.

1.5 Built-in Functions

MATLAB has many built-in common functions, like *sin()*, *cos()*, etc. We can use these to find values of these functions:

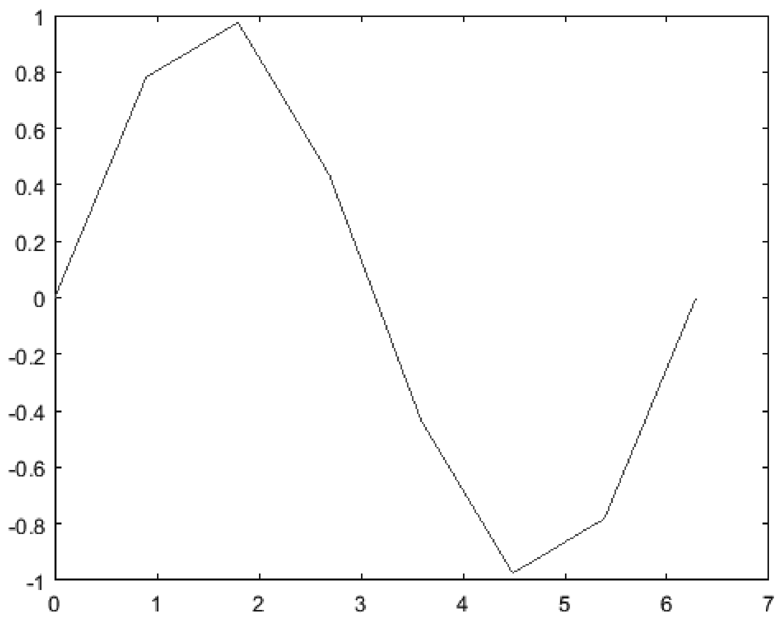
```
sin(0)

ans =
     0

cos(pi)

ans =
    -1

exp(0)

ans =
     1
```

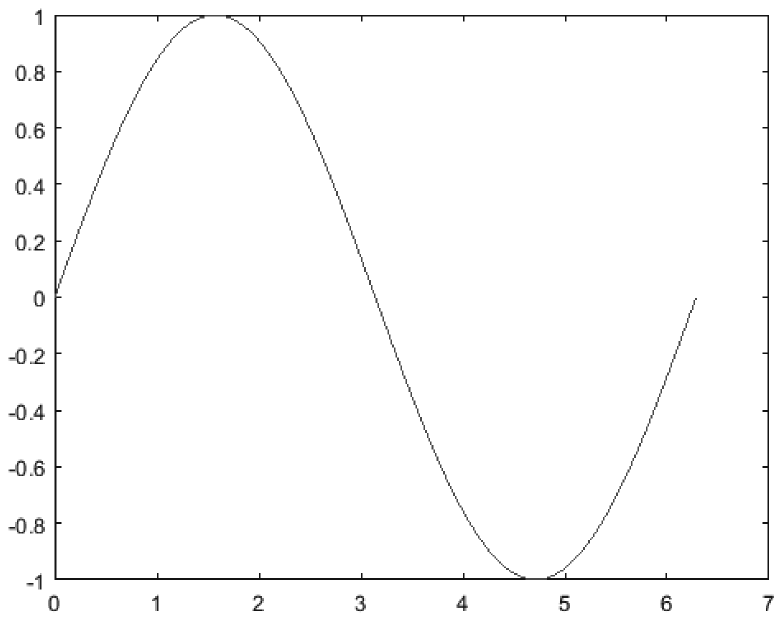
The built-in functions can also operate on vectors, which can then be used to produce plots (see next page):

```
x = linspace(0,2*pi,8);
y = sin(x);
plot(x,y)
```

Notice that this plot is not as smooth as we know the plot of a sine function should be. Remember, MATLAB is simply plotting a discrete number of data points and then connecting the data with straight lines. To get a more smooth and continuous curve, we should increase the number of data points, like this:

```
x = linspace(0,2*pi,100);
y = sin(x);
plot(x,y)
```


1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
1
2
3
4
5
6
7


1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
1
2
3
4
5
6
7

1.6 User-Defined Functions

Should we wish to define a function that is a polynomial or a combination of built-in functions, we will use an “Anonymous Function”. An anonymous function is not stored as a program file. It can accept inputs and returns outputs, just as a built-in function does.

We enter an anonymous function as follows:

```
f = @(x) 2*x.^2 - 3*x + 1
```

Notice the syntax. We name the function *f*, followed by the equals sign (=). That is followed by the at symbol (@) and then the input of the function in parentheses (x). Finally, we have the function itself, in this case, the polynomial $2x^2 - 3x + 1$.

We can evaluate the function at a single value or for a vector of values.

```
f(2.4)

ans =
    5.3200

x = -1:.5:1;
f(x)

ans =
     6     3     1     0     0
```

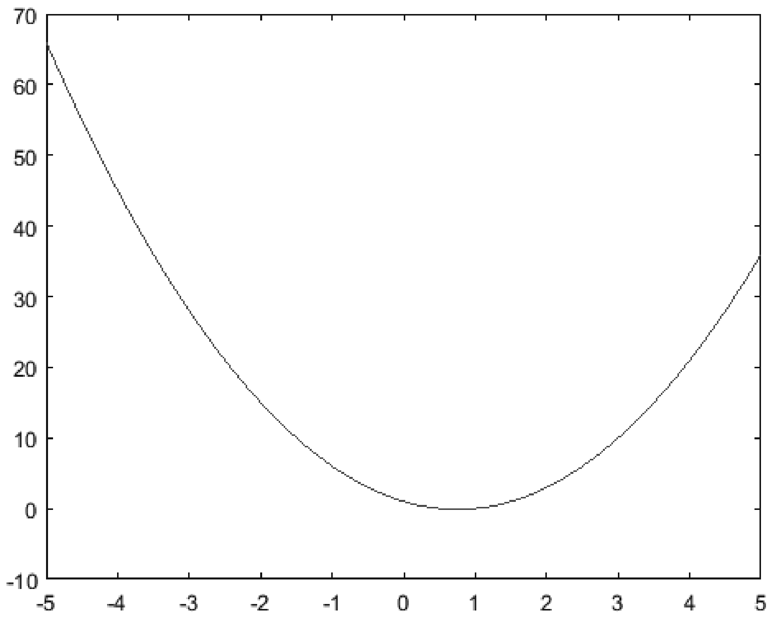
You may have noticed in the definition of our function, we used $*$ symbols every time we multiplied, but also, more importantly there was a . in before the ^ This is not a typo, but a very important indicator to MATLAB that it is not a scalar operation, but a vector operation. This is what allowed us to evaluate the function for an entire vector. By using a . in front of different operators, like ^, $*$, and /, MATLAB will perform those operations entry-wise for each component of the vector. For example, consider the following two operations, one with the . and one without:

```
[1 2 4 5].^2

ans =
     1     4    16    25

[1 2 4 5]^2

Error using  ^
Inputs must be a scalar and a square matrix.
To compute elementwise POWER, use POWER (.^) instead.
```

The . example performed the squaring operation on each element. The ^ example produced an error because it was attempting to multiply the 1×4 row vector by itself, which is not possible because the inner dimensions do not match. (In order to take the product of the vector times itself, one would need to multiply using the transpose.)

Lastly, we can use the anonymous function to plot, just as with the built-in functions.

```
x = -5:.1:5;
y = f(x);
plot(x,y)
```

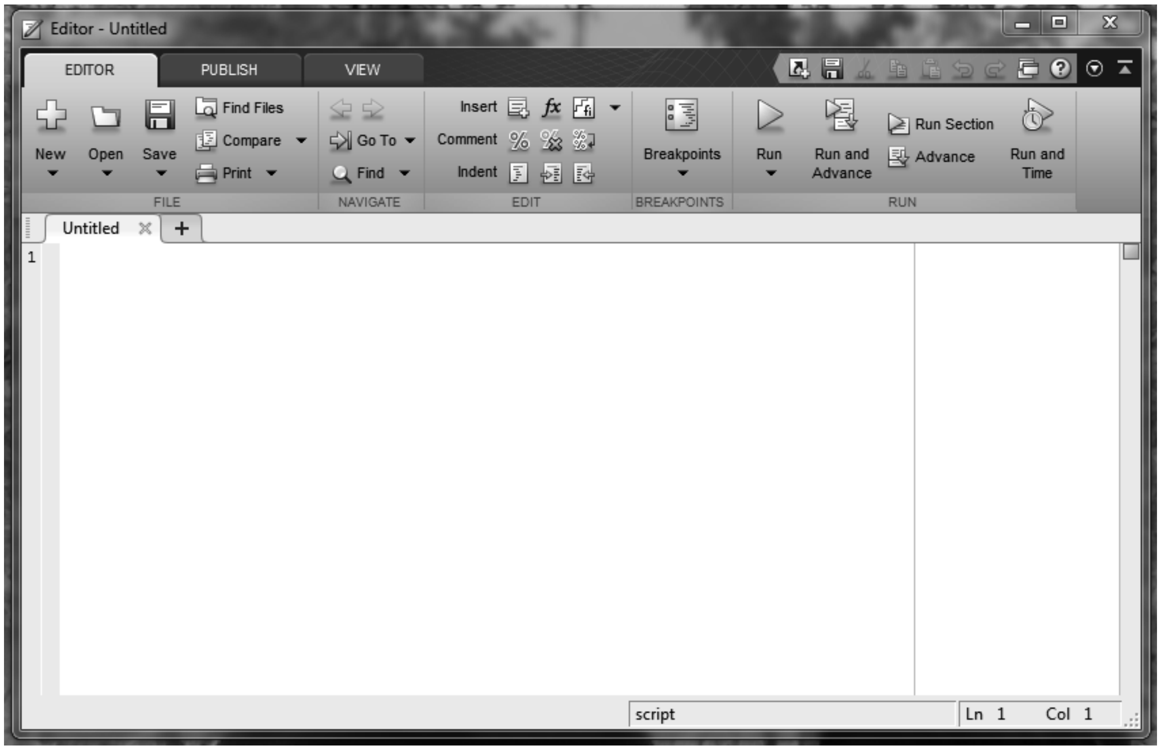


1.7 M-Files

Programs in MATLAB can be saved in files called *M-files* and have the extension .m. Generally speaking, there are two main types of M-files: *functions* and *scripts*.

1.7.1 Function Programs

Click on the new script icon, located in the upper left hand corner of the command window interface. This will open up a new window, that looks like this:



Before we create a function file, there are four important things we should know about function files:

- All function files begin with the word “function”.
- There must be an input and an output.
- The output, name of the function, and input all follow immediately after the word “function” in the first line, like this:

```
function output = FunctionName(input)
```

- Somewhere in the body of the program, the output must be assigned a value.

With these items in mind, let’s create out first function file:

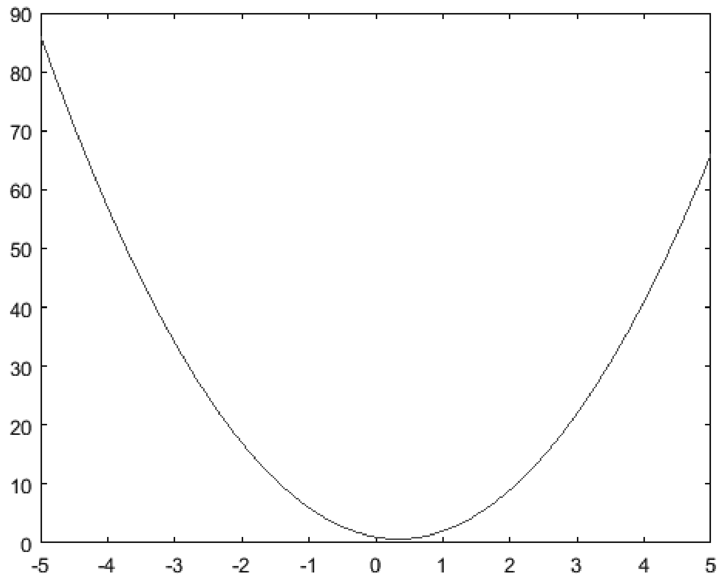
```
function y = myfunc(x)

y = 3*x.^2 - 2*x + 1;
```

Notice the output is “y”, the function name is “myfunc” and the input is “x”.

Save the file as myfunc.m in your working directory. Cautiously, you must save your M-file with the same name as what follows the equals sign in the first line of your function. This will be important when we call the functions from the command window (or another script or function file).

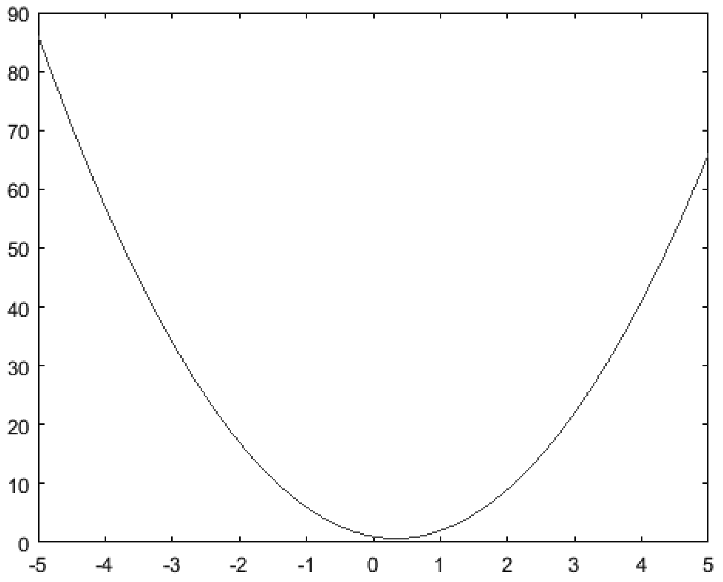
Now, let’s use your function file, by calling it in the command window. Make sure that the path in your command window points to the same working directory in which you saved your file. If it is not the right path, you will get errors. Now, let’s call the function and plot the results:

```
x = -5:.1:5;
y = myfunc(x);
plot(x,y)
```



In the command window, we can name the input and output anything we want. It does not have to match the names used in the definition of the function file. Notice that the following yields the same results:

```
z = -5:.1:5;
cat = myfunc(z);
plot(z,cat)
```



Functions can be multivariate in both input and output. Consider the following two function files, dblinput.m and mypowers.m:

```
function z = dblinput(x,y)

z = 2*exp(x.*y) + x.^2.*y.^2;
```

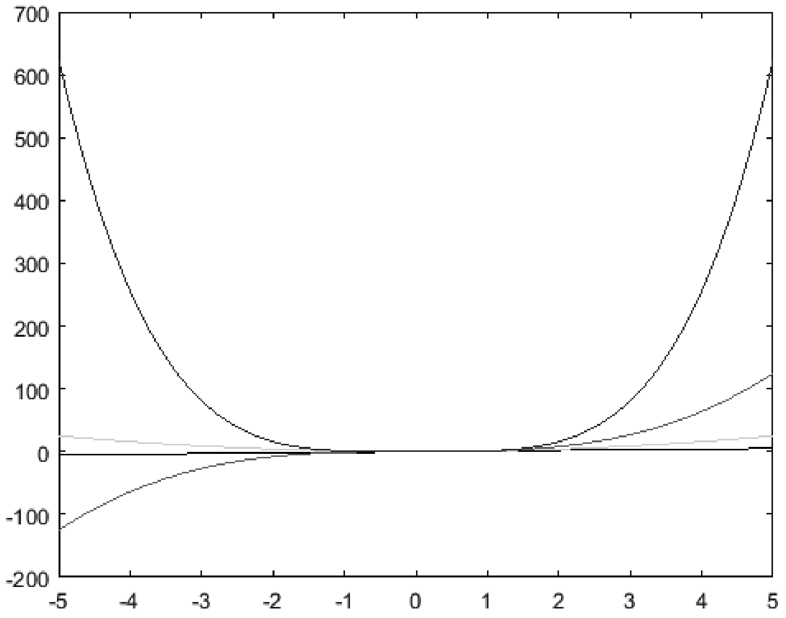
```
function [x1 x2 x3 x4] = mypowers(x)

x1 = x;
x2 = x.^2;
x3 = x.^3;
x4 = x.^4;
```

The first has two inputs (x,y) with a single output z. The second has one input x and outputs a vector [x1 x2 x3 x4].

In the command window, we see how these files are called, executed, and used.

```
z = dblinput(0,1)

z =
     2

x = 3;
```

```
[x1 x2 x3 x4] = mypowers(x)

x1 =
     3

x2 =
     9

x3 =
    27

x4 =
    81

x = -5:.1:5;
[x1 x2 x3 x4] = mypowers(x);
plot(x,x1,'black',x,x2,'green',x,x3,'red',x,x4,'blue')
```



Notice in the last plot, we could plot each of the different vectors separately, but repeating the input with various parameters for each of the vectors.

1.7.2 Script Programs

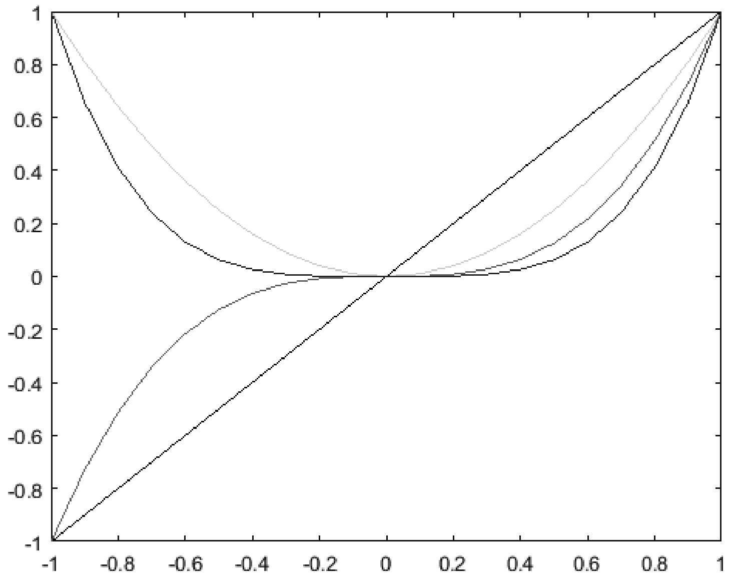
Script programs differ from functions in several ways. The key difference is that scripts do not have inputs or outputs. Also, a script can use and change variables in the workspace.

Let us begin by creating a script that essentially does the same thing as the mypowers.m function we created previously. We name it mygraphs.m.

```
x1 = x;
x2 = x.^2;
x3 = x.^3;
x4 = x.^4;

plot(x,x1,'black',x,x2,'green',x,x3,'red',x,x4,'blue')
```

Now, in the command window, we can call our script. Notice, first we define a range for x:

```
x = -1:.1:1;
mygraphs
```



Both our function and the script produced the same result; however, the script is reusable. We could pass it a different range for x without having to retype all of the commands to find the powers or to plot. For this reason, scripts are often used for routine calculations that require typing multiple commands in the command window.

1.7.3 Comments and Documentation

Now that we can save both functions and script files, it is important that we properly document our code, by "commenting" on what different commands within our programs perform. In MATLAB, anything that follows the percentage symbol (%) is considered to be a comment and will not be ignored by the compiler.

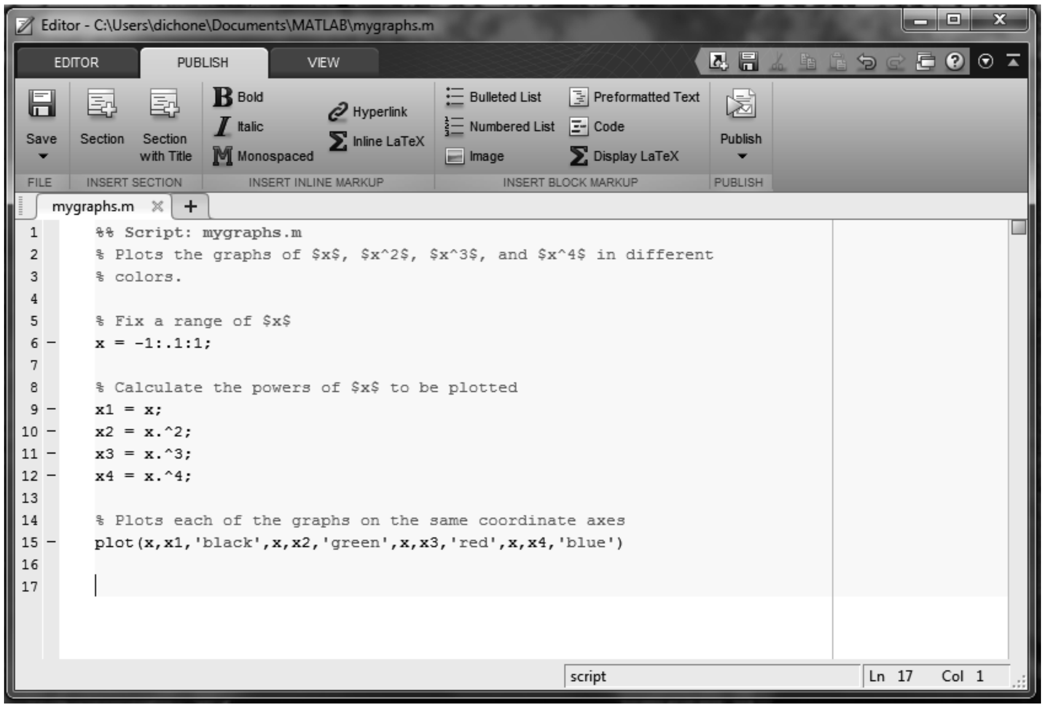
You should always include comments at the top of any .m file that includes the name of the file, a basic explanation of what the file does, and, for a function, what the inputs and outputs are. It also helps to add comments after different commands, explaining what a certain line of code executes. This is an important and helpful tool so that later, if you come back to a program you can easily remind yourself what the program does.

The following is an example of how we could comment the script we wrote in the previous subsection.

```
%% Script: mygraphs.m
% Plots the graphs of x, x^2, x^3, and x^4 in different
   colors

%Fix a range of x
x = -1:.1:1;

%Calculate the powers of x to be plotted
x1 = x;
x2 = x.^2;
x3 = x.^3;
x4 = x.^4;

%Plots each of the graphs on the same coordinate axes
plot(x,x1,'black',x,x2,'green',x,x3,'red',x,x4,'blue')
```

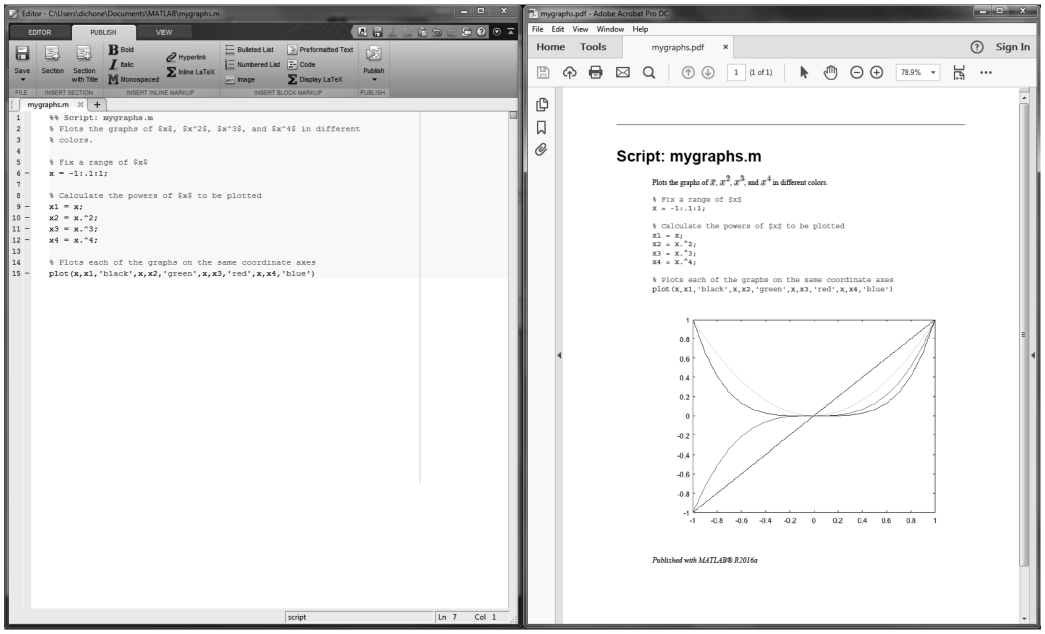
1.8 Publish MATLAB Code

At the top of the editor window, where you create scripts and functions, you will notice a tab named “Publish”. Click on it and you will see a new menu bar at the top.



```
%% Script: mygraphs.m
% Plots the graphs of $x$, $x^2$, $x^3$, and $x^4$ in different
% colors.

% Fix a range of $x$
x = -1:.1:1;

% Calculate the powers of $x$ to be plotted
x1 = x;
x2 = x.^2;
x3 = x.^3;
x4 = x.^4;

% Plots each of the graphs on the same coordinate axes
plot(x,x1,'black',x,x2,'green',x,x3,'red',x,x4,'blue')

```

We will now publish our script mygraphs.m by clicking “Publish”. In the figure, you can see our code (with some modified LaTeX) on the left and on the right is the resulting published PDF file.



```
%% Script: mygraphs.m
% Plots the graphs of $x$, $x^2$, $x^3$, and $x^4$ in different
% colors.

% Fix a range of $x$
x = -1:.1:1;

% Calculate the powers of $x$ to be plotted
x1 = x;
x2 = x.^2;
x3 = x.^3;
x4 = x.^4;

% Plots each of the graphs on the same coordinate axes
plot(x,x1,'black',x,x2,'green',x,x3,'red',x,x4,'blue')
```

Script: mygraphs.m

Plots the graphs of x, x^2, x^3, and x^4 in different colors.

```
% Fix a range of $x$
x = -1:.1:1;

% Calculate the powers of $x$ to be plotted
x1 = x;
x2 = x.^2;
x3 = x.^3;
x4 = x.^4;

% Plots each of the graphs on the same coordinate axes
plot(x,x1,'black',x,x2,'green',x,x3,'red',x,x4,'blue')
```

Published with MATLAB® R2016a

The first time, your file may publish in a different format, like HTML or XPS. You can change the output in the “Edit Publishing Options” menu. We will not go into great detail here about all the different features of Publish. As you work through this text and produce examples and programs of your own, you will discover the different features necessary if and when you choose to use the Publish feature in MATLAB.

Problems

1.1. Create a table of data. Input it as vectors and plot it. Turn in the graph.

1.2. Create an anonymous function $f(x) = x + \cos(x^2)$. Plot it on the domain $-5 \le x \le 5$, with a step size of 1. In what way does the graph incorrectly represent the function? What can you do to make the graph properly represent the function? Fix it and turn in both plots.

1.3. Write a function program for the function $x^2 e^{-x^2}$. Include adequate comments in the program. Plot the function on the interval $[-5, 5]$. Turn in the program and graph.

1.4. Write a script program that graphs the functions $\sin(x)$, $\sin(2x)$, $\sin(3x)$, $\sin(4x)$, $\sin(5x)$, and $\sin(6x)$ on the interval $[0, 2\pi]$ on the same set of coordinate axes. Include comments in the program. Turn in the program and the graph.