

MATLAB CRASH COURSE

A quickstart guide to master MATLAB basics

by Aatmaj K. Mhatre

MATLAB CRASH COURSE

Forward

This book covers all essential basics of MATLAB along with relevant code examples. This book is a quickstart guide towards learning MATLAB and not a comprehensive reference encyclopedia. This book focuses more on speed and pace rather than technical completeness.

MATLAB is very easy to learn language. Using this book, you can master MATLAB in at most three hours. This book is designed in such a manner that it will give you a very clear idea of what MATLAB is and help you cover most of the basics in a very short span of time.

Before you proceed to learn MATLAB, however I would like to share a few words of caution. MATLAB is neither free nor open. Unless you are a student affiliated to any university MATLAB might not be the best fit for you. However, there are good alternatives to MATLAB. Among them options, I would advise two alternatives- Python (with scientific add-ons) and the Julia language. Julia is a better alternative to MATLAB with syntax very similar to it. However one point of advantage of learning MATLAB is that the syntax of MATLAB is very similar to that of GNU Octave, a free alternative to MATLAB.

I hope that you will find the contents of this book at your advantage.

Prerequisites- Basic knowledge of programming. (Variables, Control statements, etc)

Summary of the Book

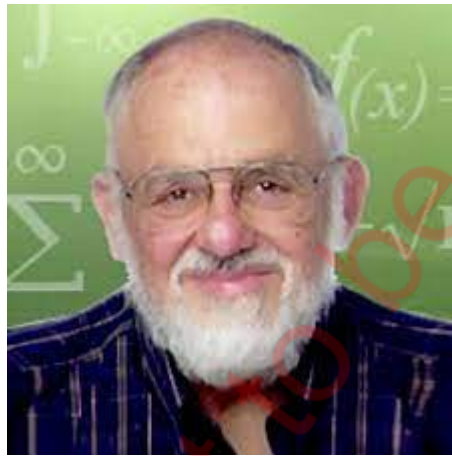
- **Chapter One** : An introduction to what MATLAB is and where is it used. MATLAB stands for MATrix LABoratory. MATLAB is a scientific programming language used for simulations and data driven research. We will write our hello world code in this part. After that, we will learn all about MATLAB variables which we can declare using the '=' sign, that is the assignment operator.
- **Chapter Two**: We will learn about some predefined functions in MATLAB like `rand()` `scatter()` and `min()` . MATLAB contains loads of other functions, whose documentation can be found out on https://in.mathworks.com/help/matlab/elementary-math.html?s_tid=CRUX_lftnav. After that we will learn about control statements in MATLAB including the syntax for the while loop, for loop and if-else-elseif statements.
- **Chapter Three**: We will study about row and column vectors in MATLAB. Sometimes, however we may require to create row or column vectors with large number of equidistant points. We will learn how to do that automatically in MATLAB. At the end we also will learn how to take the transpose of any matrix.
- **Chapter Four**: We will have had understood the syntax for the generation of matrices at the end of this chapter. In a similar way of making matrices, we can combine them together and fuse two matrices. Besides, there also exist inbuilt functions for matrix generation, which we have covered in this chapter.
- **Chapter Five**: We will check out how to operate on matrices in MATLAB. We will have a look at matrix multiplications. Matrix multiplication can be done easily by just multiplying two matrices together. Many times we need to find mean, mode or median of a vector. Such a statistical analysis of the data can be done easily in MATLAB using the inbuilt functions we cover.
- **Chapter Six**: This part is reserved for accessing vectors and matrices. Many times, we need to access the elements in middle of a vector or matrix. MATLAB provides easy methods to extract out the elements. Similarly we can easily access matrices

At the end of the book, you will find yourselves familiar with the basic MATLAB syntax, operations and matrices. That's a promise!

Chapter One: Introduction

What is MATLAB?-

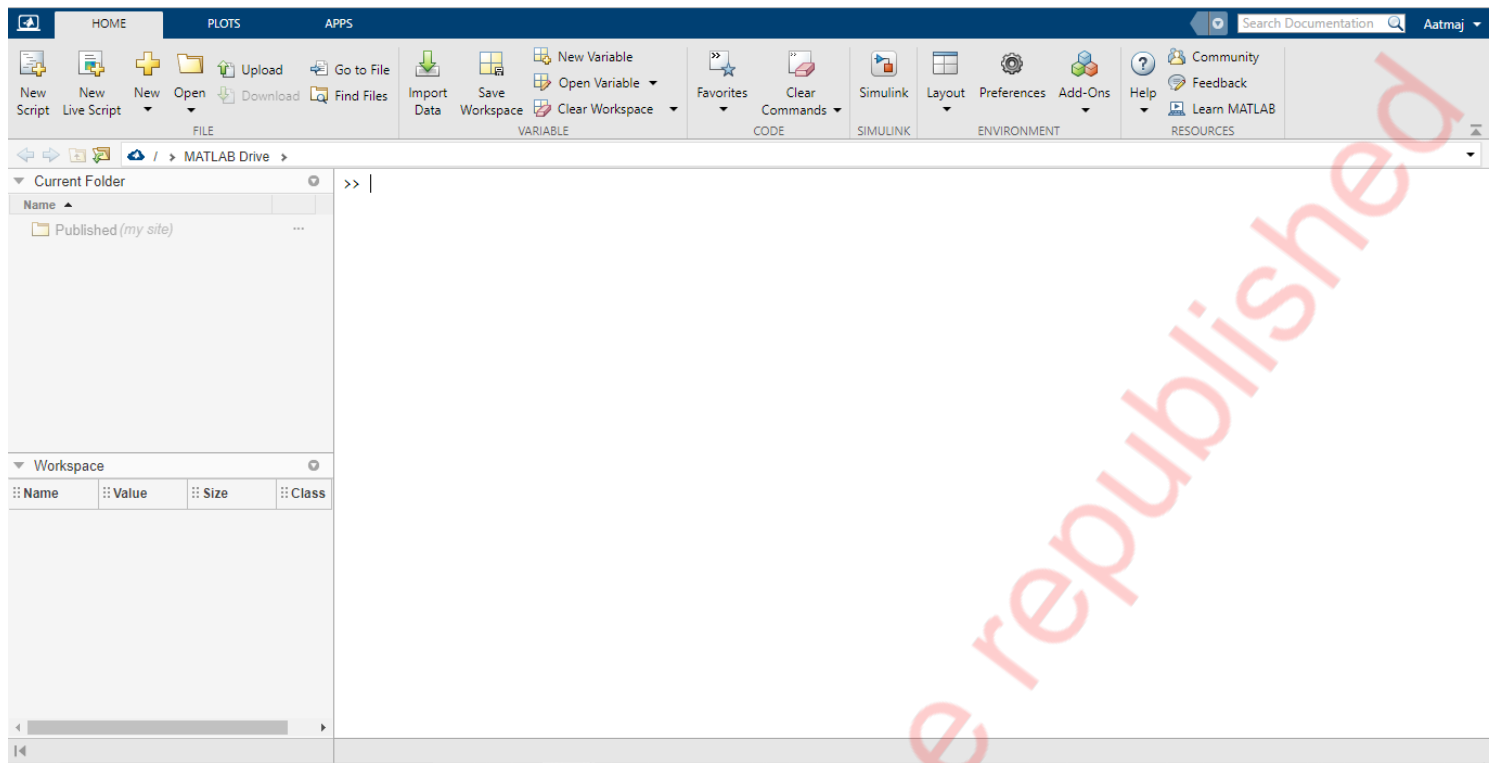
MATLAB stands for MATrix LABoratory. MATLAB is a scientific programming language. ✨ It is mainly used for solving mathematical equations and data-visualization. Complicated systems can be better understood with this software. Speed and ease of writing are the salient features of MATLAB. Professor Cleve Moler designed MATLAB. He did so with an intention to reduce heavy FORTRAN coding. The aim of the language was making an engineer friendly syntax.



MATLAB is used by students and engineers in many disciplines like Robotics, Astrology, Machine Learning, Image Processing and Biology.

MATLAB web browser

The MATLAB web browser let's us run MATLAB through the web. First, login (create a new account) using your university account. If you do not have a university account, you can use a free 30- day trial too!



The command prompt

MATLAB online contains the command prompt central to the page, where we can write our code. Similar to python, MATLAB let's us calculate numbers at the command prompt, and the answer is displayed in the command window.

← → ↗ ↘ ☁ / > MATLAB Drive >

▼ Current Folder ⓘ

Name ▲

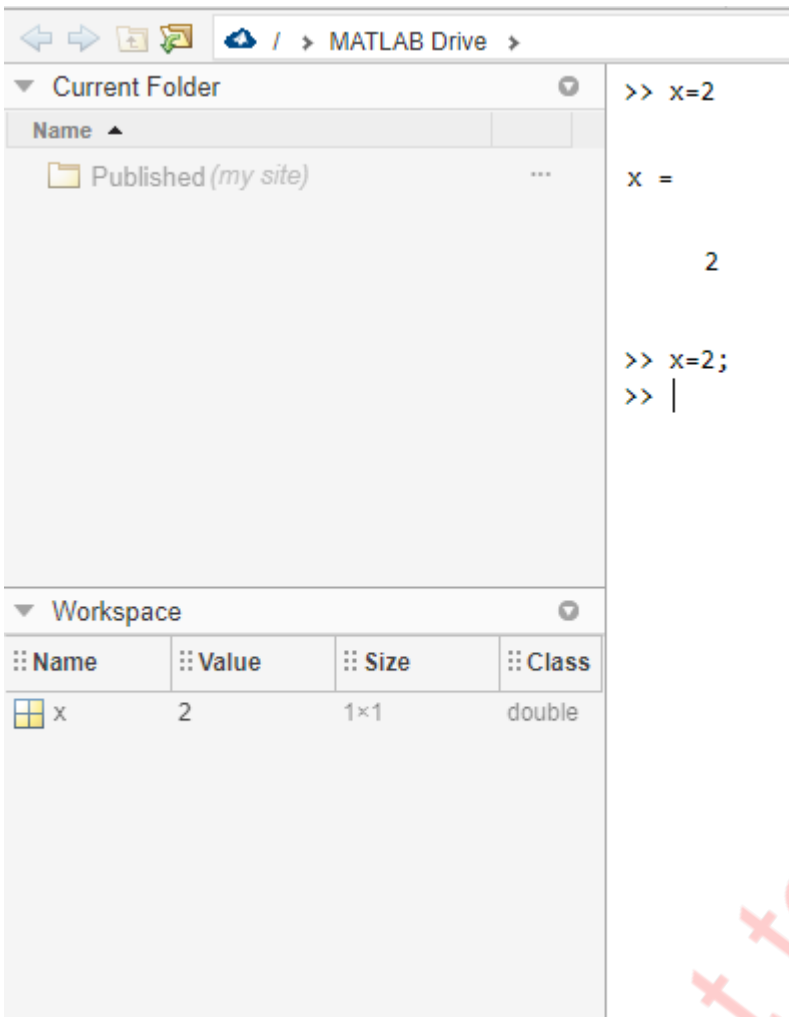
Published (my site) ...

▼ Workspace ⓘ

Name	Value	Size	Class
ans	8.3333e-04	1×1	double

```
>> 22/7  
  
ans =  
  
3.1429  
  
>> 1+9  
  
ans =  
  
10  
  
>> 1/1200  
  
ans =  
  
8.3333e-04  
  
>>
```

Adding a semicolon at the end of the statement prevents output from being displayed.



Similar to many languages, too small or too large values like the last one are displayed using scientific notation.

MATLAB operators follow this order-

1. Parenthesis
2. Exponentiation
3. Product and Division
4. Addition and Subtraction

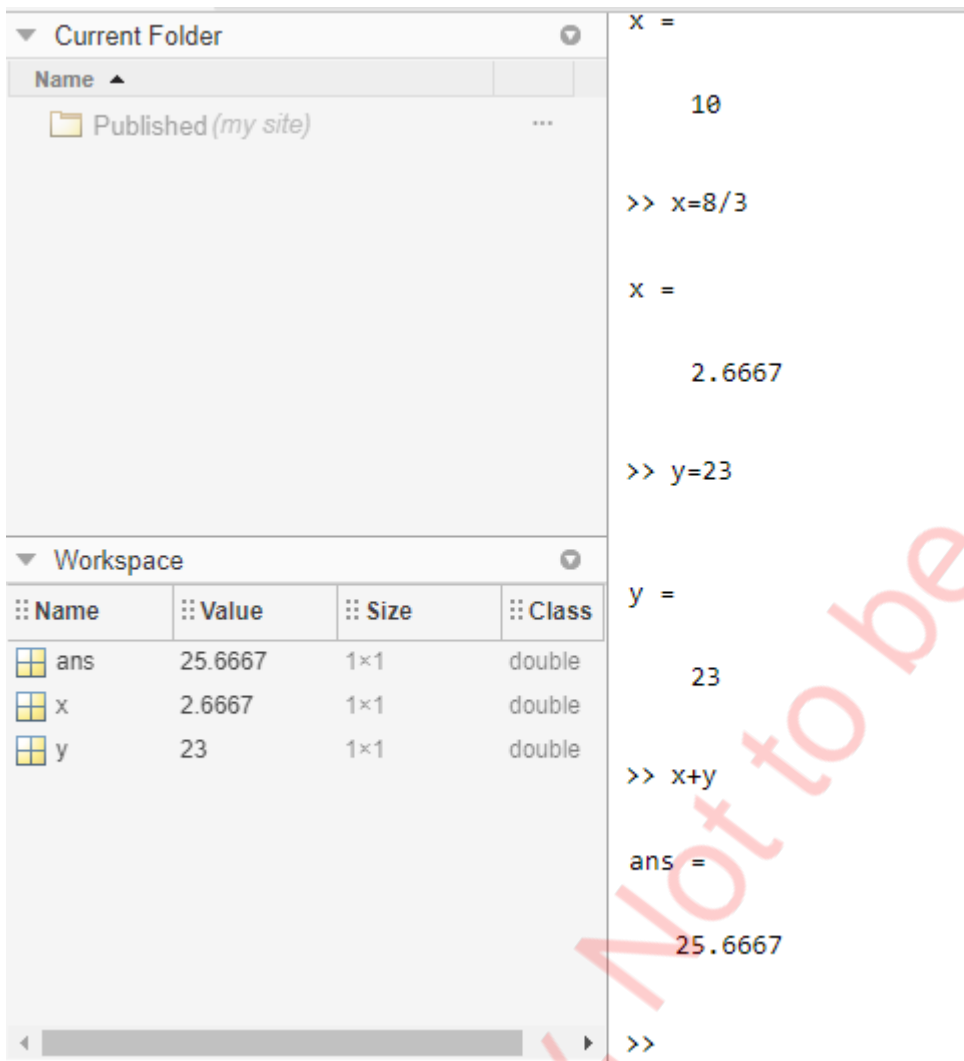
So for complicated calculations, it is important to specify the brackets.

The command prompt can be quit using the command "exit"

MATLAB VARIABLES We can declare variables using the '=' sign, that is the assignment operator. The following are the rules for variable declaration in MATLAB are same as Python as below

Variables must begin with a letter, are case sensitive, may contain only letters numbers or underscores

The variables declared are located at the left bottom side of the page. (WORKSPACE) The values of the variables are also displayed along with their class. Note that MATLAB treats 'ans' as a variable too. So whenever operations are performed, the variable 'ans' is overridden.



The image shows a MATLAB interface with three main panels. The 'Current Folder' panel on the left shows a folder named 'Published (my site)'. The 'Workspace' panel below it lists three variables: 'ans' (value 25.6667, size 1x1, class double), 'x' (value 2.6667, size 1x1, class double), and 'y' (value 23, size 1x1, class double). The 'Command Window' on the right shows the following sequence of commands and outputs:

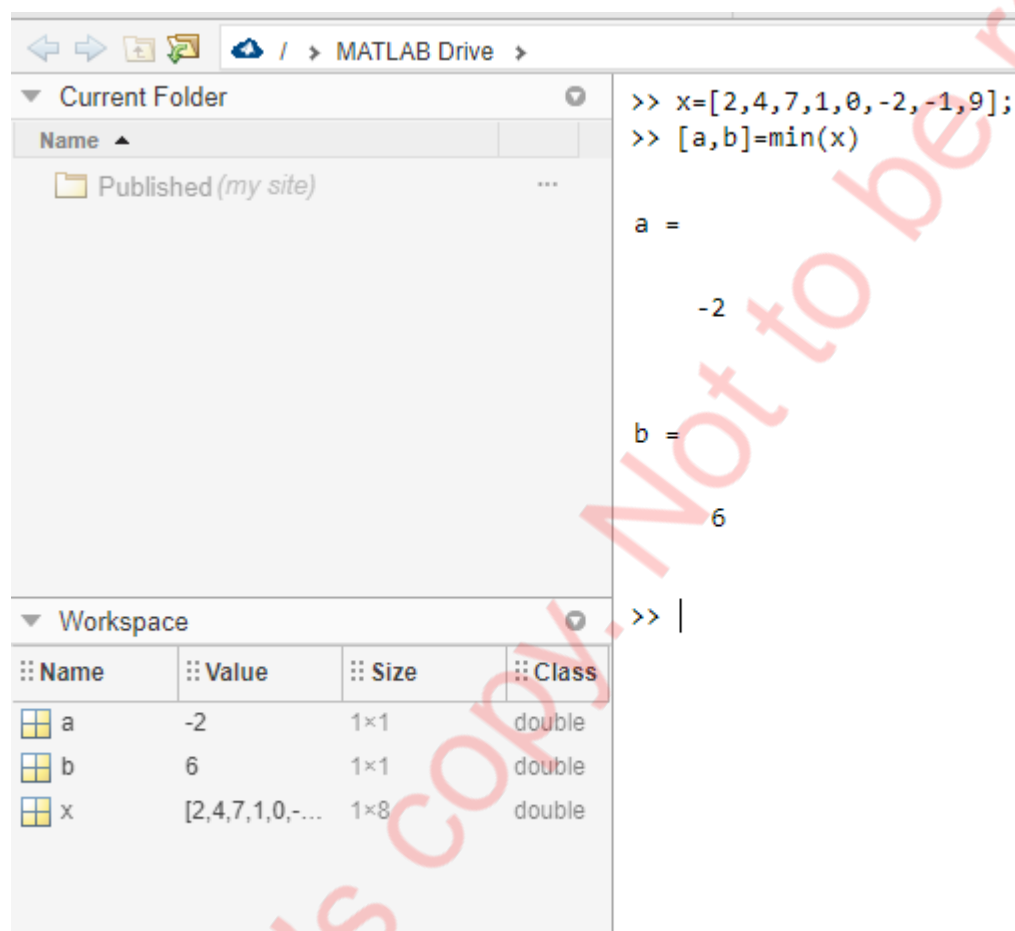
```
x =  
10  
  
>> x=8/3  
  
x =  
2.6667  
  
>> y=23  
  
y =  
23  
  
>> x+y  
  
ans =  
25.6667  
  
>>
```

A large diagonal watermark reading 'Author's copy. Not to be republished' is overlaid across the entire image.

Chapter Two: Inbuilt Functions and Control Statements

In built Functions in MATLAB

MATLAB supports standard mathematical functions like sin, cos, log, exponentiation, square root, and many more. However, unlike many languages, input for these functions can be an integer or even an matrix vectors. Many MATLAB functions take in multiple inputs and return multiple outputs. e.g. the min function can take in an array of numbers and output both, the smallest number and the position of the smallest number.



```
>> x=[2,4,7,1,0,-2,-1,9];
>> [a,b]=min(x)

a =

    -2

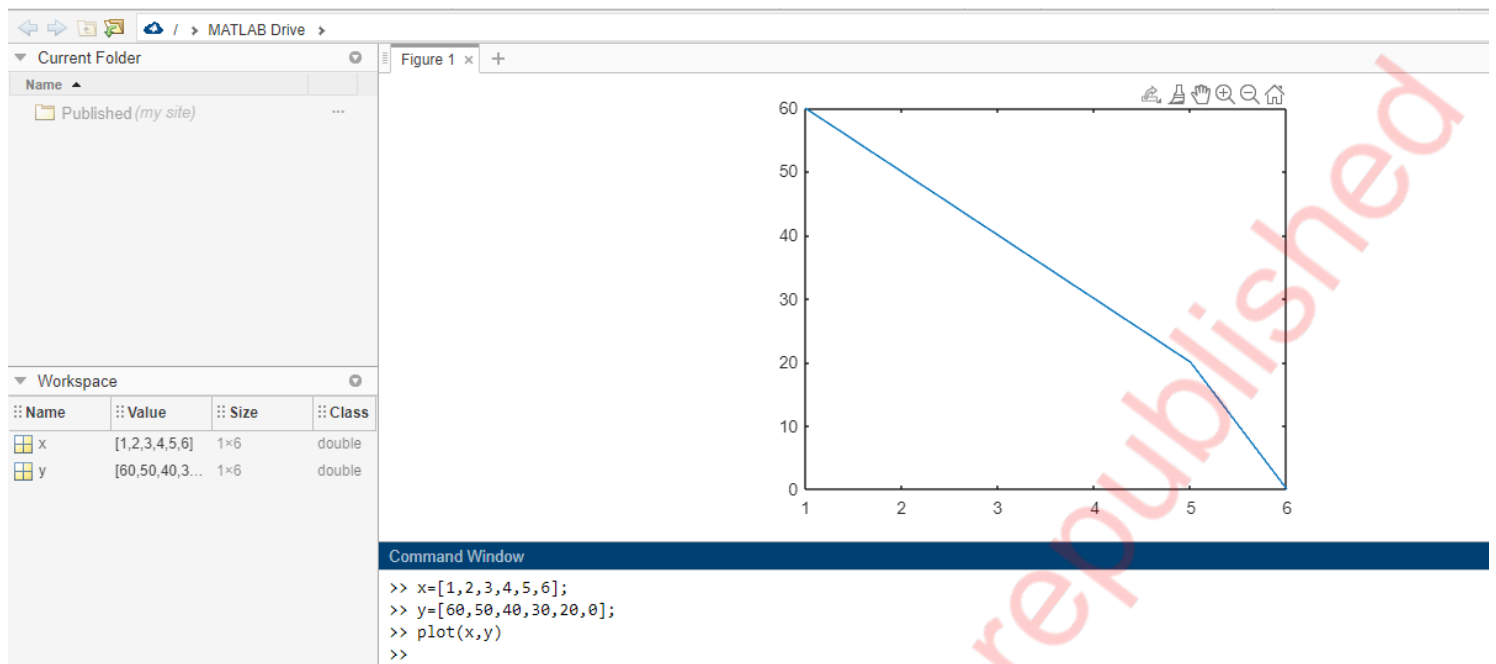
b =

     6

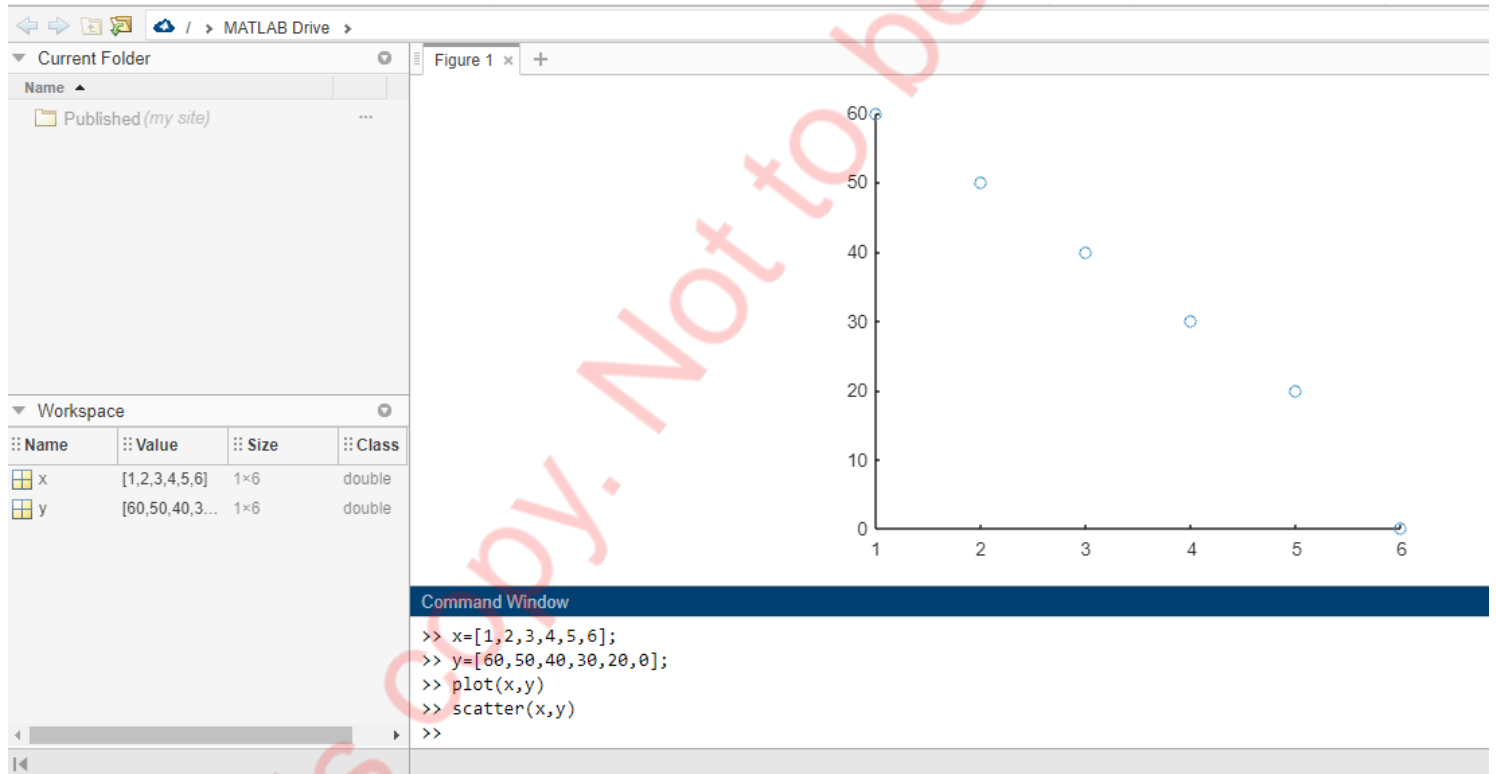
>> |
```

Name	Value	Size	Class
a	-2	1×1	double
b	6	1×1	double
x	[2,4,7,1,0,-2,-1,9]	1×8	double

Similarly, the function plot takes in two matrices and plots their values. The plot connects all the points represented by the x,y pairs.



Another similar function is `scatter()` which generates the scatterplot of the functions.



MATLAB contains loads of other functions, whose documentation can be found out here -

https://in.mathworks.com/help/matlab/elementary-math.html?s_tid=CRUX_lftnav

One more valuable function is `rand()`, which returns a random value.

```
>> x=rand()
```

```
x =
```

Program control statements

MATLAB has the for loop syntax as follows

```
for i=a:b
    c=d
end
```

Here the running variable i assumes values from 0,99 Python equivalent of this statement is

```
for i in range(a,b):
    #function
```

Here is a sample program-

Name	Value	Size	Class
i	8	1×1	double
x	512	1×1	double

```
>> x=1;
>> for i=0:8
x=x*2;
>> end
>> x

x =

512
```

The if else statement has the following syntax

```
if a<b
    a=b
elseif a>b
    a=0
else
    b=0
end
```

The Python equivalent is

```
if a<b:
    a=b
elif a>b:
    a=0
else:
    b=0
```

Here is a sample

```
>> x=10;
>> if x<10
x=3
>> elseif x==10
>> x=pi
>> else
>> x=100
>> end
x =

    3.1416
```

While loop

MATLAB also has the while loop with syntax as

```
while condition
statement
end
```

For an infinite loop,

```
while true
statement
end
```

Here is an example which uses while loop- the Collatz conjecture

```
>> for k=1:100
x=k;
while x>1
if mod(x,2)==0
x=x/2
else
x=3*x+1
```

end
end
end

Note that % operator in python is replaced by the mod() function in MATLAB. mod(a,b) is equivalent to a%b

Chapter Three: Row and Column Vectors

Row and Column Vectors in MATLAB

We can create row vector in matlab by this syntax-

```
x=[-2,-1,0,-1,-2]
```

Similarly we can make column vectors using ':' instead of ','

```
x=[-2;-1;0;-1;-2]
```

Here is the output-

```
>> x=[2;3;4;5];
```

```
>> x
```

```
x =
```

```
2
3
4
5
```

```
>> x=[2,3,4,5,];
```

```
>> x
```

```
x =
```

```
2    3    4    5
```

Sometimes, however we may require to create row or column vectors with large number of equidistant points. Example, for plotting values from -2 to 2, we may require to plot at steps of small units like 0.01. In that case, MATLAB provides a feature to enable creation of "Uniformly Based Vectors" We can create such vectors by inputting startvalue, spacing and end value -

```
x=startvalue: spacing: endvalue
```

Here is an example-

```
>> x=-1:0.17:+1
```

```
x =  
-1.0000 -0.8300 -0.6600 -0.4900 -0.3200 -0.1500 0.0200 0.1900 0.3600 0.5
```

Note- Vector ends on the largest value within range. That is, after adding up consecutive spacing values, vector ends in the greatest value smaller than the endvalue

If the spacing value is not mentioned, it defaults to one-

```
>> y=[-4:5]
```

```
y =  
-4 -3 -2 -1 0 1 2 3 4 5
```

Similarly, all this can be done for column vectors as well

The transpose operator

The row vectors can be transformed into column vectors using the transpose operator

```
x=(Startvalue:spacing:endvalue)'
```

This operator transposes any matrix (flips over rows and columns)

```
>> z=(-3:1)
```

```
z =  
-3 -2 -1 0 1
```

```
>> x=(z)'
```

```
x =  
-3  
-2  
-1
```

Author's copy. Not to be republished

Chapter Four: Matrices in MATLAB

Making matrices in MATLAB

We will now learn how to create Matrices in MATLAB. Similer to row vectors, the elements in a row are separated by a ',' and column by a ';'. First, type in the values for the first column separated by commas. When the row ends, type a semicolon and the proceed on to the next row. Repeat until done, and end with the square brackets.

```
>> x=[1,2,3,4,5;6,7,8,9,10;11,12,13,14,15;16,17,18,19,20]
```

```
x =
```

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20

Mismatch of elements in rows or columns generates this error-

```
>> x=[1,2,3;4,5,6;7,8]
```

Error using [vertcat](#)

Dimensions of arrays being concatenated are not consistent.

Fusing two Matrices

In a similar way of making matrices, we can combine them together with this syntax

```
a=[ ]  
b=[ ]  
c=[a;b]
```

This is vertical concatenation. The two matrices will be placed one on top of the other and joined together

For horizontal concatenation, use the syntax `c=[a,b]`

```
>> a=[1,2,3;4,5,6;7,8,9];
>> b=[10,11,12;13,14,15;16,17,18];
>> c=[a;b]
```

$$C =$$

1	2	3
4	5	6
7	8	9
10	11	12
13	14	15
16	17	18

```
>> c=[a,b]
```

$$C =$$

1	2	3	10	11	12
4	5	6	13	14	15
7	8	9	16	17	18

Note, For horizontal Concatenation, the number of rows for both matrices must be the same, and for vertical concatenation, the number of columns must be equal. If this rule is violated, Concatenation error is generated.

Matrix generation functions

There are a few matrix generation functions in MATLAB like `eye()` this function generates an Identity matrix of the size we input. example-

```
>> I=eye(10)
```

$$I =$$
[illegible]

The zeros() function creates a square matrix of zeros. For rectangular matrices, two arguments can be used.

```
>> z=zeros(3)
```

```
z =
```

0	0	0
0	0	0
0	0	0

```
>> z=zeros(3,5)
```

```
z =
```

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

We can use the exact same syntax for the ones function and the rand function.

```
>> x=rand(5,6)
```

```
x =
```

0.8147	0.0975	0.1576	0.1419	0.6557	0.7577
0.9058	0.2785	0.9706	0.4218	0.0357	0.7431
0.1270	0.5469	0.9572	0.9157	0.8491	0.3922
0.9134	0.9575	0.4854	0.7922	0.9340	0.6555
0.6324	0.9649	0.8003	0.9595	0.6787	0.1712

```
>> x=ones(5,6)
```

```
x =
```

1	1	1	1	1	1
1	1	1	1	1	1
1	1	1	1	1	1
1	1	1	1	1	1
1	1	1	1	1	1

Similarly there is the diag function for a diagonal matrix

```
>> x=diag(1:7)
```

```
x =
```

1	0	0	0	0	0	0
0	2	0	0	0	0	0
0	0	3	0	0	0	0
0	0	0	4	0	0	0
0	0	0	0	5	0	0
0	0	0	0	0	6	0
0	0	0	0	0	0	7

Author's copy. Not to be republished

Chapter Five: Calculations on Matrices

Matrix calculations

We may at times require to manipulate the individual elements of matrices. We can make these matrix calculations similar to variable operations. Example we want to make a matrix y such that each element in y is twice that of each in x . We can do so by $y=2*x$

```
>> x=[1,2,3,4,5];
>> y=2*x

y =

     2     4     6     8    10
```

Now, if we want to make another vector z which is the addition in x and y , $z=x+y$

```
>> z=x+y

z =

     3     6     9    12    15

>> z=x-y

z =

    -1    -2    -3    -4    -5
```

However, when we multiply the two, an error occurs.

```
>> z=x*y
Error using .*
```

Incorrect dimensions for matrix multiplication. Check that the number of columns in the first matrix matches the number of rows in the second matrix. To perform elementwise multiplication, use '.*'.

This is because the `*` operator means matrix multiplication and not element wise multiplication. For element wise multiplication, we use `.*`. Similar for division `./` and exponentiation `.^`

```
>> z=y.*x

z =
```

```
2      8      18      32      50
```

```
>> z=x./y
```

```
z =
```

```
0.5000    0.5000    0.5000    0.5000    0.5000
```

```
>> z=x.^y
```

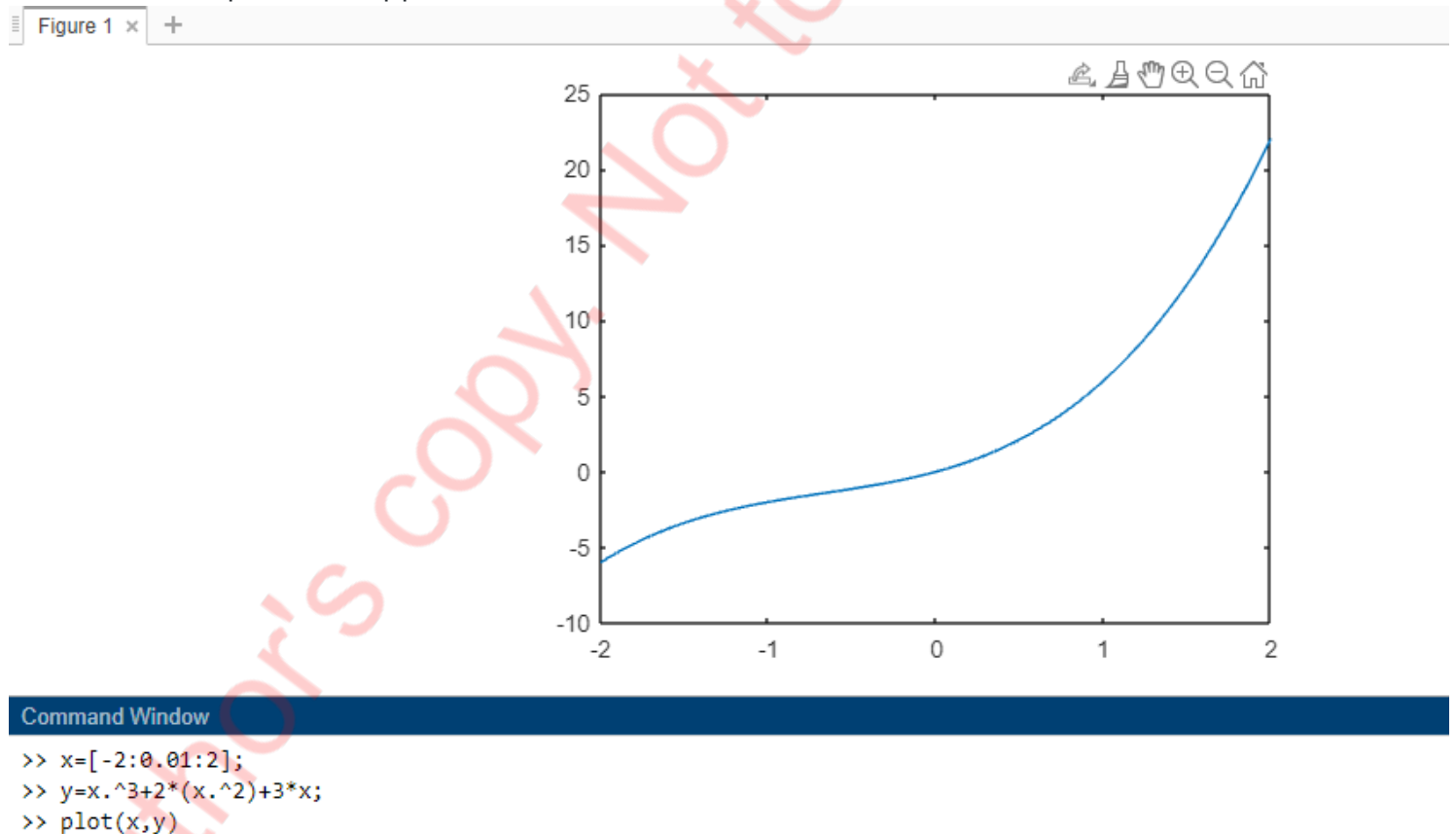
```
z =
```

```
1         16        729       65536      9765625
```

When dealing with two matrices, always add a dot before operators for making element wise operators `.*` , `./` and `.^` . This is because normal operators are reserved for matrix operations.

Note, the matrix operations are only valid for matrices of same sizes and dimensions. If this is not followed, this error pops up- **Arrays have incompatible sizes for this operation.**

Here is an example of the application of what we learnt so far-



Try plotting a few equations yourself....

Matrix multiplication

Matrix multiplication can be done easily by just multiplying the two together. However, be careful that the dimensions for matrix multiplication match.

```
>> x=[1,2,3;4,5,6]
```

```
x =
```

```
    1    2    3
    4    5    6
```

```
>> y=[1,2;3,4;5,6]
```

```
y =
```

```
    1    2
    3    4
    5    6
```

```
>> x*y
```

```
ans =
```

```
    22    28
    49    64
```

Statistical functions

Many times we need to find mean, mode or median of a vector. Such a statistical analysis of the data can be done easily in MATLAB using inbuilt functions. The `mean()` function can find the mean of the vector elements, while `median()` returns the central number. The `mode()` gives out the mode of the vector, while `var()` and `std()` return the variance and the standard deviations respectively. `rms()` indicated the root mean square. Apart from these widely used functions, you will find many more in the MATLAB documentation.

```
>> x=[1,2,3,4,5,4,3,2,1,1];
```

```
>> mean(x)
```

```
ans =
```

```
    2.6000
```

```
>> mode(x)
```

```
ans =
```

1

```
>> var(x)
```

```
ans =
```

```
2.0444
```

```
>> std(x)
```

```
ans =
```

```
1.4298
```

```
>> median(x)
```

```
ans =
```

```
2.5000
```

```
>> y=median(x).*x
```

```
y =
```

```
2.5000 5.0000 7.5000 10.0000 12.5000 10.0000 7.5000 5.0000 2.5000 2.5
```

```
>> rms(x)
```

```
ans =
```

```
2.9326
```


Chapter Six Accessing Vector Elements

Many times, we need to access the elements in middle of a vector or matrix. MATLAB provides easy methods to extract out the elements.

Accessing vector elements-

We can access and the vector elements as shown in the example

```
>> x=[1,2,3,4,0,6];  
>> x(5)
```

```
ans =
```

```
0
```

```
>> x(5)=5;  
>> x
```

```
x =
```

```
1    2    3    4    5    6
```

MATLAB also allows us to access many elements at once. The result is a 'n' element vector

```
>> x=[2,4,6,8];  
>> y=x([2,3])
```

```
y =
```

```
4    6
```

```
>> y=x([1,3,4])
```

```
y =
```

```
2    6    8
```

An alternative may be

```
>> x=[2,4,6,8];  
>> data=1:3;  
>> y=x(data)
```

```
y =
```

2 4 6

We can access the last element of the vector using the "end" keyword.

```
>> x=[2,4,6,8];
```

```
>> x(end)
```

```
ans =
```

```
8
```

```
>> x(end-1)
```

```
ans =
```

```
6
```

Accessing matrix values- Matrix values require both the row and the column number to be accessed

```
>> x=rand(5,6)
```

```
x =
```

0.9058	0.2785	0.9706	0.4218	0.0357	0.7431
0.1270	0.5469	0.9572	0.9157	0.8491	0.3922
0.9134	0.9575	0.4854	0.7922	0.9340	0.6555
0.6324	0.9649	0.8003	0.9595	0.6787	0.1712
0.0975	0.1576	0.1419	0.6557	0.7577	0.7060

```
>> x(2,3)
```

```
ans =
```

```
0.9572
```

We can also extract a whole matrix using the syntax `x(rows,columns)`, where `rows` is an array of rows while `columns` is array of columns. Here are two examples that will make things very clear.

```
>> x=rand(7,8)
```

```
x =
```

	0.9961	0.8173	0.9106	0.5499	0.0760	0.9027	0.1112	0.9421
	0.0782	0.8687	0.1818	0.1450	0.2399	0.9448	0.7803	0.9561
3	0.4427	0.0844	0.2638	0.8530	0.1233	0.4909	0.3897	0.5752
	0.1067	0.3998	0.1455	0.6221	0.1839	0.4893	0.2417	0.0598
	0.9619	0.2599	0.1361	0.3510	0.2400	0.3377	0.4039	0.2348
6	0.0046	0.8001	0.8693	0.5132	0.4173	0.9001	0.0965	0.3532
	0.7749	0.4314	0.5797	0.4018	0.0497	0.3692	0.1320	0.8212

```
>> x([3,6],[2,4])
```

```
ans =
```

0.0844	0.8530
0.8001	0.5132

Note that 2:4 means [2,3,4] while comma separated 2,4 means [2,4]

```
>> x=rand(7,7)
```

```
x =
```

	0.7791	0.0305	0.8594	0.4899	0.6820	0.7224	0.4538
2	0.7150	0.7441	0.8055	0.1679	0.0424	0.1499	0.4324
3	0.9037	0.5000	0.5767	0.9787	0.0714	0.6596	0.8253
4	0.8909	0.4799	0.1829	0.7127	0.5216	0.5186	0.0835
	0.3342	0.9047	0.2399	0.5005	0.0967	0.9730	0.1332
	0.6987	0.6099	0.8865	0.4711	0.8181	0.6490	0.1734
	0.1978	0.6177	0.0287	0.0596	0.8175	0.8003	0.3909

```
>> x([2:4],[5,7])
```

```
ans =
```

0.0424	0.4324
0.0714	0.8253
0.5216	0.0835

In order to take an entire column or row, we can just place a semicolon ':' for it. Here is an example-

```
>> x=rand(5,5)
```

x =

0.8314	0.4168	0.0155	0.1981	0.0527
0.8034	0.6569	0.9841	0.4897	0.7379
0.0605	0.6280	0.1672	0.3395	0.2691
0.3993	0.2920	0.1062	0.9516	0.4228
0.5269	0.4317	0.3724	0.9203	0.5479

>> x(:,3)

ans =

0.0155
0.9841
0.1672
0.1062
0.3724

>> x(3,:)

ans =

0.0605 0.6280 0.1672 0.3395 0.2691