**MATLAB/Octave - Basic Operations**

**Elementary Math Operations**

**#addition**

>> 5+6

ans =

11

**#subtraction**

>> 3-1

ans =

2

**#multiplication**

>> 4\*5

ans =

20

**#division**

>> 1/4

ans =

0.2500

**#power**

>> 2^5

ans =

32

**Logical Operations**

**#equal**

>> 2 == 1

ans =

logical

0

**#not equal**

>> 2 ~= 1

ans =

logical

1

**#AND operator**

>> 0 && 1

ans =

logical

0

**#OR operator**

>> 0 || 1

ans =

logical

1

**#XOR operator**

>> xor(1,0)

ans =

logical

1

**Assignment Operator**

**#assign numeric value**

>> a = 3

a =

3

**#assign string value**

>> b = 'Hi'

b =

'Hi'

**#use semicolon to skip printing output onto the console**

>> a = 3;

**#print variable value**

>> a

a =

3

>> b

b =

'Hi'

**#assign logical value**

>> c = (3>=1)

c =

logical

1

**#assign mathematical term 'pi' value to variable a**

>> a = pi;

**#print variable a**

>> a

a =

3.1416

**#more ways to display variable values in a good format**

>> disp(a)

3.1416

**#print limited decimals for any given variable, in this case upto 2 decimal values**

>> disp(sprintf('2 decimals: %0.2f', a))

2 decimals: 3.14

**#upto 6 decimal values**

>> disp(sprintf('2 decimals: %0.6f', a))

2 decimals: 3.141593

**#original format**

>> a

a =

3.1416

**#change default format to long**

>> format long

**#example of long value format**

>> a

a =

3.141592653589793

**#change default format to short**

>> format short

**#example of short value format**

>> a

a =

3.1416

**MATLAB/Octave - Vectors and Matrices**

**#assign matrix to variable A, semicolon used here to split rows**

>> A = [1 2; 3 4; 5 6]

A =

1 2

3 4

5 6

**#another way of defining matrix**

>> A = [1 2;

>> 3 4;

>> 5 6]

A =

1 2

3 4

5 6

**#row vector or 1\*3 matrix**

>> V = [1 2 3]

V =

1 2 3

**#column vector or 3\*1 matrix**

>> v = [1; 2; 3]

v =

1

2

3

**#range of values with fixed interval with both sides inclusive**

>> 1:0.1:2

ans =

1.0000 1.1000 1.2000 1.3000 1.4000 1.5000 1.6000 1.7000 1.8000 1.9000 2.0000

**#sequence of value with default step size of 1**

>> 1:6

ans =

1 2 3 4 5 6

**#ones matrix of 2\*3 size, here 2 is row and 3 is column**

>> ones(2,3)

ans =

1 1 1

1 1 1

**#twos matrix by multiplying 2 with ones matrix**

>> c = 2\*ones(2,3)

c =

2 2 2

2 2 2

**#another way of defining the twos matrix**

>> c = [2 2 2; 2 2 2]

c =

2 2 2

2 2 2

**#zeros matrix of 1\*3 size**

>> w = zeros(1,3)

w =

0 0 0

**#random matrix of 1\*3 size with values between 0 and 1**

>> w = rand(1,3)

w =

0.8147 0.9058 0.1270

**#random matrix of 3\*3 size with values between 0 and 1**

>> rand(3,3)

ans =

0.9134 0.2785 0.9649

0.6324 0.5469 0.1576

0.0975 0.9575 0.9706

**#random matrix from Gaussian distribution with mean=0 and standard deviation=1**

>> w = randn(1,3)

w =

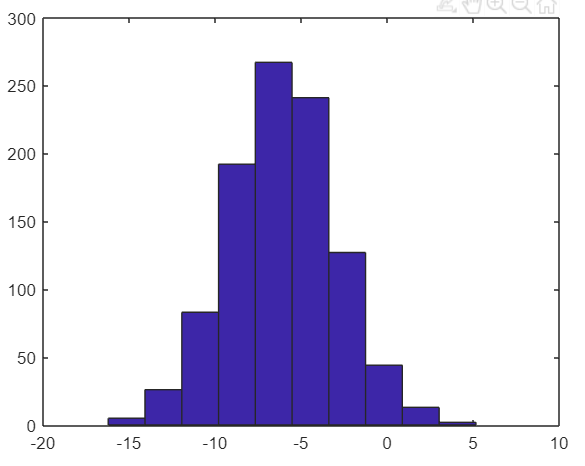
0.7254 -0.0631 0.7147

**#expression**

>> w = -6 + sqrt(10) \* (randn(1,1000));

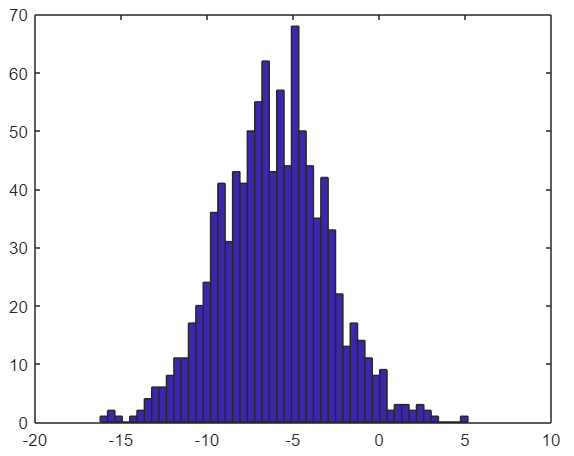
**#plot histogram for above expression**

>> hist(w)



**#plot histogram with new bin size=50**

>> hist(w, 50)



**#identity matrix of 5\*5 size**

>> eye(5)

ans =

1 0 0 0 0

0 1 0 0 0

0 0 1 0 0

0 0 0 1 0

0 0 0 0 1

**#identity matrix of 3\*3 size**

>> I = eye(3)

I =

1 0 0

0 1 0

0 0 1

**#seek help for any method/function/keyword**

>> help eye

eye Identity matrix.

eye(N) is the N-by-N identity matrix.

eye(M,N) or eye([M,N]) is an M-by-N matrix with 1's on

the diagonal and zeros elsewhere.

<.. more>

**MATLAB/Octave - Moving Data Around**

**# printing matrix A value for later use**

>> A

A =

1 2

3 4

5 6

**#shape of matrix A (former element shoes number of rows and latter one shows number of columns)**

>> size(A)

ans =

3 2

**#display number of rows in a given matrix A**

>> size(A, 1)

ans =

3

**#display number of columns in a given matrix A**

>> size(A, 2)

ans =

2

**#printing vector V for later use**

>> V

V =

1 2 3

**#length of vector V**

>> length(V)

ans =

3

**#length of matrix A, here length function will return either number of rows or columns whichever is maximum**

>> length(A)

ans =

3

**#to print the current working directory**

>> pwd

ans =

'/MATLAB Drive'

**#print the content of the current directory**

>> ls

machine-learning-ex machine-learning-ex.zip Published

**#change directory**

>> cd machine-learning-ex/

**#go back to the previous directory**

>> cd ..

**#print all variables present in the current session**

>> who

Your variables are:

A I V a ans b c v w

**#print all variables (detailed view) present in the current session**

>> whos

Name Size Bytes Class Attributes

A 3x2 48 double

I 3x3 72 double

V 1x3 24 double

a 1x1 8 double

ans 1x13 26 char

b 1x2 4 char

c 2x3 48 double

v 3x1 24 double

w 1x1000 8000 double

**#delete variable 'w' from the current session**

>> clear w

**#confirm that 'w' variable is not in the current session list**

>> whos

Name Size Bytes Class Attributes

A 3x2 48 double

I 3x3 72 double

V 1x3 24 double

a 1x1 8 double

ans 1x13 26 char

b 1x2 4 char

c 2x3 48 double

v 3x1 24 double

**#load data file present in the current directory**

>> load priceY.dat

**#print the content of the data file read in the above step, here variable name is same as filename without extension**

>> priceY

priceY =

2150

1250

4100

2563

6452

<.. more values>

**#create new variable from existing variable, assign top 10 elements of priceY vector to new variable**

>> w = priceY(1:10)

w =

2150

1250

4100

2563

6452

8452

6547

5124

7519

3520

**#save the vector/matrix in the new file**

>> save hello.mat w

**#to delete all the variables from the current session**

>> clear

**#check if everything is gone or not**

>> whos

**#load the saved file again**

>> load hello.mat

**#confirmed if the load is successful or not**

>> whos

Name Size Bytes Class Attributes

w 10x1 80 double

**#save the vector/matrix in the human readable format**

>> save hello.txt w -ascii

**#create matrix A for later use**

>> A = [1 2; 3 4; 5 6]

A =

1 2

3 4

5 6

**#fetch the specific element of A, in this case element from 3rd row 2nd column**

>> A(3,2)

ans =

6

**#fetch all element of specific row, in this case get all element from 2nd row**

>> A(2,:)

ans =

3 4

**#fetch all element of specific column, in this case get all element from 2nd column**

>> A(:,2)

ans =

2

4

6

**#fetch multiple rows element**

>> A([1,3],:)

ans =

1 2

5 6

**#assign new values on the specific location of the matrix, in this case assign [10;11;12] vector to the second row of the matrix**

>> A(:,2) = [10; 11; 12]

A =

1 10

3 11

5 12

**#concatenate 2 different matrix/vector one after another using comma or space**

>> A = [A, [100; 101; 102]]

A =

1 10 100

3 11 101

5 12 102

**#check the size of matrix A after above concatenation**

>> size(A)

ans =

3 3

**#put all elements in a single column vector using colon ':'**

>> A(:)

ans =

1

3

5

10

11

12

100

101

102

**#create matrix A for later use**

>> A = [1 2; 3 4; 5 6]

A =

1 2

3 4

5 6

**#create matrix B for later use**

>> B = [11 12; 13 14; 15 16]

B =

11 12

13 14

15 16

**#concatenate two matrix/vectors one after another using comma or space**

>> C = [A B]

C =

1 2 11 12

3 4 13 14

5 6 15 16

**#concatenate two matrix/vectors one below another using semicolon**

>> C = [A; B]

C =

1 2

3 4

5 6

11 12

13 14

15 16

**#check the size of matrix C after above concatenation**

>> size(C)

ans =

6 2

**MATLAB/Octave - Computing on data**

**#create matrix A for later use**

>> A = [1 2; 3 4; 5 6]

A =

1 2

3 4

5 6

**#create matrix B for later use**

>> B = [11 12; 13 14; 15 16]

B =

11 12

13 14

15 16

**#create matrix C for later use**

>> C = [1 1; 2 2]

C =

1 1

2 2

**#multiply 2 matrix A and C, number of columns of first matrix should match with number of rows of second matrix**

>> A \* C

ans =

5 5

11 11

17 17

**#unsuccessful multiplication of 2 matrix A and B as above condition didn't meet**

>> A \* B

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 '.\*'.

**#element wise multiplication of 2 matrix, here both the number of rows and columns should match between both the matrices**

>> A .\* B

ans =

11 24

39 56

75 96

**#element wise square**

>> A .^ 2

ans =

1 4

9 16

25 36

**#create vector V for later use**

>> V = [ 1; 2; 3]

V =

1

2

3

**#element wise division**

>> 1 ./ V

ans =

1.0000

0.5000

0.3333

**#log of the vector V**

>> log(V)

ans =

0

0.6931

1.0986

**#exp of the vector V**

>> exp(V)

ans =

2.7183

7.3891

20.0855

**#absolute values of random vector containing negative integers**

>> abs([-1; 2; -3])

ans =

1

2

3

**#multiply every element of vector V with -1**

>> -1 \* V

ans =

-1

-2

-3

**#same as above but with short cut approach**

>> -V

ans =

-1

-2

-3

**#add ones to every element of vector V**

>> V + ones(length(V),1)

ans =

2

3

4

**#same as above bur with short cut approach**

>> V + 1

ans =

2

3

4

**#print matrix A**

>> A

A =

1 2

3 4

5 6

**#transpose of A**

>> A'

ans =

1 3 5

2 4 6

**#double transpose of A gives back A only as per the transpose property**

>> (A')'

ans =

1 2

3 4

5 6

**#single row vector**

>> a = [1 15 2 0.5]

a =

1.0000 15.0000 2.0000 0.5000

**#find maximum value from any vector**

>> val = max(a)

val =

15

**#fetch maximum value from a vector with its index location**

>> [val, ind] = max(a)

val =

15

ind =

2

**#print matrix A**

>> A

A =

1 2

3 4

5 6

**#find maximum element from every column of matrix A**

>> max(A)

ans =

5 6

**#print row vector 'a'**

>> a

a =

1.0000 15.0000 2.0000 0.5000

**#element wise comparison, return 1 as true and 0 as false based on the given condition**

>> a < 3

ans =

1×4 logical array

1 0 1 1

**#return elements which satisfy the given condition**

>> find(a<3)

ans =

1 3 4

**#magic matrix, the sum of every row, column, diagonals are same**

>> A = magic(3)

A =

8 1 6

3 5 7

4 9 2

**#find row and column location of the element of a matrix which satisfy the given condition**

>> [r, c] = find(A >= 7)

r =

1

3

2

c =

1

2

3

**#fetch specific element of a matrix**

>> A(2,3)

ans =

7

**#print row vector 'a'**

>> a

a =

1.0000 15.0000 2.0000 0.5000

**#sum of all the elements of vector 'a'**

>> sum(a)

ans =

18.5000

**#prod of all the elements of vector 'a'**

>> prod(a)

ans =

15

**#get floor values of all the elements of vetor/matrix**

>> floor(a)

ans =

1 15 2 0

**#get ceil values of all the elements of vetor/matrix**

>> ceil(a)

ans =

1 15 2 1

**#fetch element wise maximum of 2 random matries**

>> max(rand(3), rand(3))

ans =

0.7458 0.6173 0.8021

0.8131 0.5755 0.9891

0.9861 0.5301 0.4516

**#print matrix A**

>> A

A =

8 1 6

3 5 7

4 9 2

**#column wise maximum values for a given matrix**

>> max(A, [], 1)

ans =

8 9 7

**#row wise maximum values for a given matrix**

>> max(A, [], 2)

ans =

8

7

9

**#column wise maximum values for a given matrix if no extra parameter given (default behaviour)**

>> max(A)

ans =

8 9 7

**#maximum element from the whole matrix**

>> max(max(A))

ans =

9

**#same as above with different approach**

>> max(A(:))

ans =

9

**#magic matrix**

>> A = magic(9)

A =

47 58 69 80 1 12 23 34 45

57 68 79 9 11 22 33 44 46

67 78 8 10 21 32 43 54 56

77 7 18 20 31 42 53 55 66

6 17 19 30 41 52 63 65 76

16 27 29 40 51 62 64 75 5

26 28 39 50 61 72 74 4 15

36 38 49 60 71 73 3 14 25

37 48 59 70 81 2 13 24 35

**#column wise sum**

>> sum(A)

ans =

369 369 369 369 369 369 369 369 369

**#column wise sum**

>> sum(A,1)

ans =

369 369 369 369 369 369 369 369 369

**#row wise sum**

>> sum(A,2)

ans =

369

369

369

369

369

369

369

369

369

**#print matrix A**

>> A

A =

47 58 69 80 1 12 23 34 45

57 68 79 9 11 22 33 44 46

67 78 8 10 21 32 43 54 56

77 7 18 20 31 42 53 55 66

6 17 19 30 41 52 63 65 76

16 27 29 40 51 62 64 75 5

26 28 39 50 61 72 74 4 15

36 38 49 60 71 73 3 14 25

37 48 59 70 81 2 13 24 35

**#fetch diagonal elements of any matrix using identity (eye) matrix**

>> A .\* eye(9)

ans =

47 0 0 0 0 0 0 0 0

0 68 0 0 0 0 0 0 0

0 0 8 0 0 0 0 0 0

0 0 0 20 0 0 0 0 0

0 0 0 0 41 0 0 0 0

0 0 0 0 0 62 0 0 0

0 0 0 0 0 0 74 0 0

0 0 0 0 0 0 0 14 0

0 0 0 0 0 0 0 0 35

**#sum of diagonal elements on a matrix**

>> sum(sum(A .\* eye(9)))

ans =

369

**#flip upside down any matrix, in this case eye matrix has been fliped upside down**

>> flipud(eye(9))

ans =

0 0 0 0 0 0 0 0 1

0 0 0 0 0 0 0 1 0

0 0 0 0 0 0 1 0 0

0 0 0 0 0 1 0 0 0

0 0 0 0 1 0 0 0 0

0 0 0 1 0 0 0 0 0

0 0 1 0 0 0 0 0 0

0 1 0 0 0 0 0 0 0

1 0 0 0 0 0 0 0 0

**#magic matrix**

>> A = magic(3)

A =

8 1 6

3 5 7

4 9 2

**#pseudo inverse of matrix A**

>> temp = pinv(A)

temp =

0.1472 -0.1444 0.0639

-0.0611 0.0222 0.1056

-0.0194 0.1889 -0.1028

**#multiply matrix A and pseudo inverse of A gives back identity matrix based on the matrix property**

>> temp \* A

ans =

1.0000 0.0000 -0.0000

-0.0000 1.0000 0.0000

0.0000 -0.0000 1.0000

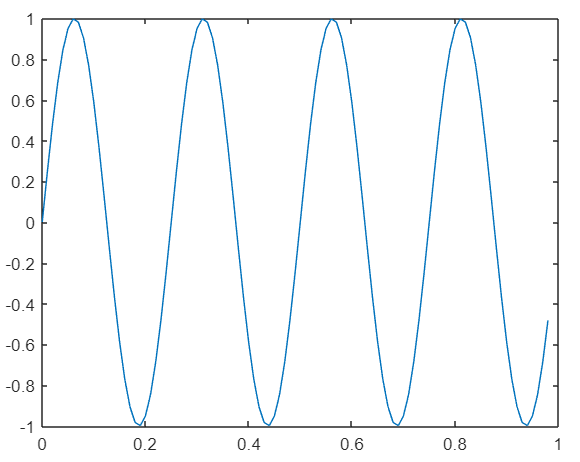
**MATLAB/Octave - Plotting Data**

**#plot sin graph**

>> t = [0:0.01:0.98];

>> y1 = sin(2\*pi\*4\*t);

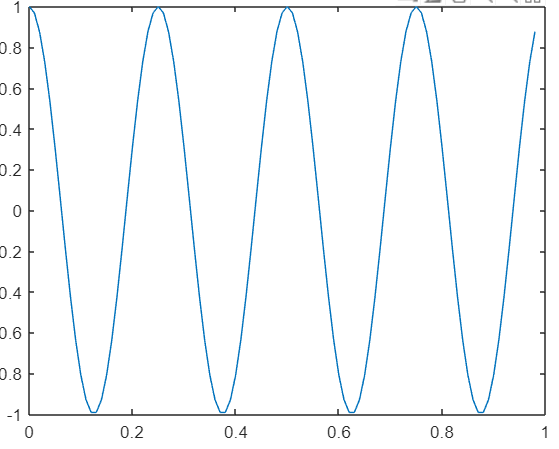
>> plot(t, y1)



**#plot cos graph**

>> y2 = cos(2\*pi\*4\*t);

>> plot(t, y2)



**#plot sin and cos graph one onto another with all labels**

>> plot(t, y1)

>> hold on;

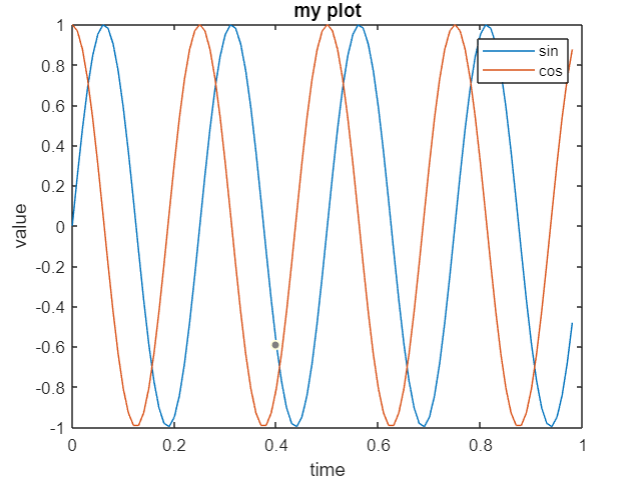
>> plot(t, y2)

>> xlabel('time')

>> ylabel('value')

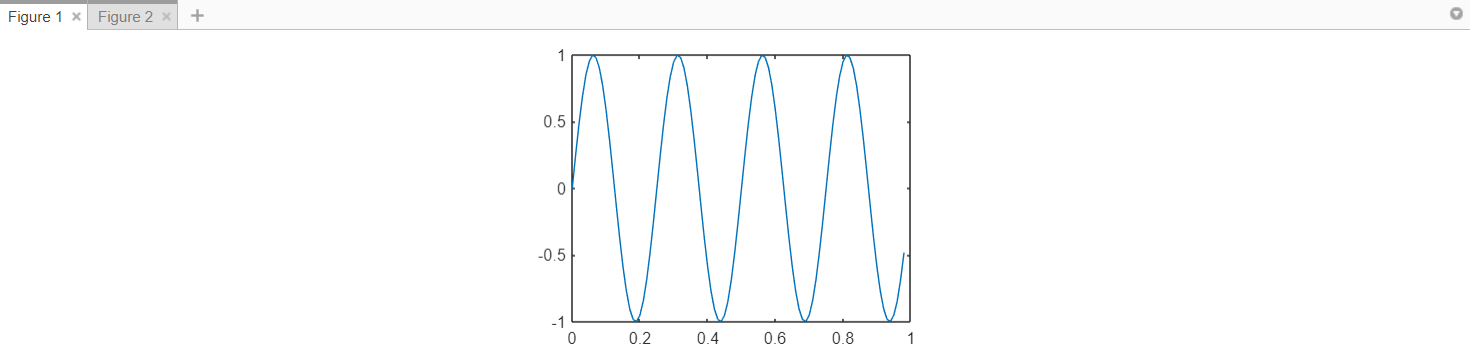
>> legend('sin', 'cos')

>> title('my plot')



**#plot sin graph with title on the tab**

>> figure(1); plot(t, y1);



**#plot sin and cos graph one after another in a same row with different axis**

>> subplot(1,2,1);

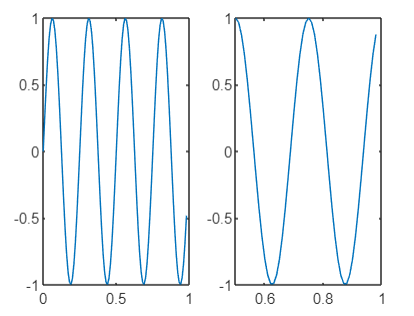
>> subplot(1,2,1)

>> plot(t, y1)

>> subplot(1,2,2)

>> plot(t, y2)

>> axis([0.5 1 -1 1])

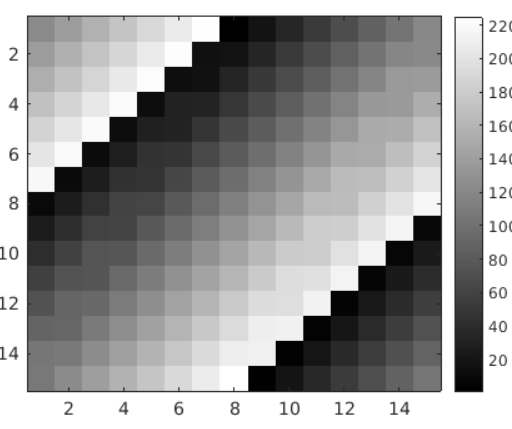


**#clear the frame**

>> clf;

**#visualize the matrix using colorbar scheme**

>> imagesc(magic(15)), colorbar, colormap gray;



**#run various commands in a single line with values printing onto the console**

>> a=1, b=2, c=3

a =

1

b =

2

c =

3

**#run various commands in a single line without values printing onto the console**

>> a=1; b=2; c=3;

**MATLAB/Octave - Control statements**

**#create zeros vector for later use**

>> v = zeros(10,1)

v =

0

0

0

0

0

0

0

0

0

0

**#for loop implementation**

>> for i=1:10,

v(i) = 2^i;

end;

>> v

v =

2

4

8

16

32

64

128

256

512

1024

**#same as above but with different approach**

>> indices = 1:10

indices =

1 2 3 4 5 6 7 8 9 10

>> for i=indices,

disp(i);

end;

1

2

3

4

5

6

7

8

9

10

**#print vector 'v'**

>> v

v =

2

4

8

16

32

64

128

256

512

1024

**#while loop implementation**

>> i=1;

>> while i <= 5,

v(i) = 100;

i = i+1;

end;

>> v

v =

100

100

100

100

100

64

128

256

512

1024

**#while loop implementation with different approach**

>> i=1;

>> while true,

v(i) = 999;

i = i+1;

if i==6,

break;

end;

end;

**#print modified vector**

>> v

v =

999

999

999

999

999

64

128

256

512

1024

**#change value of first element for later use**

>> v(1)

ans =

999

**#if then else implementation**

>> v(1) = 2;

>> if v(1)==1,

disp('The value is one');

elseif v(1)==2,

disp('The value is two');

else

disp('The value is not one or two');

end;

The value is two

**#user defined function - create local file containing function definition with 'filename.m' extension, call the function**

****

>> squareThisNumber(5)

ans =

25

**#same as above but with different function definition**

****

>> [a, b] = squareAndCubeThisNumber(2)

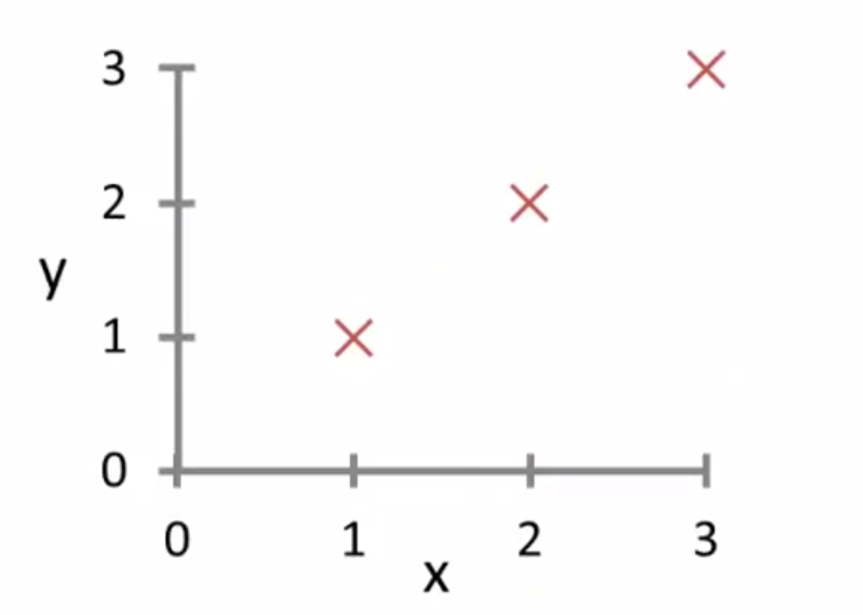
a =

4

b =

8

**#Define a function to compute the cost function J(theta)**



**#create feature matrix**

>> X = [1 1; 1 2; 1 3]

X =

1 1

1 2

1 3

**#create prediction vector**

>> Y = [1; 2; 3]

Y =

1

2

3

**#create parameters vector**

>> theta = [0; 1]

theta =

0

1

**#call user defined function to calculate cost function**

****

>> costFunctionJ(X, Y, theta)

ans =

0

**#check for different values of theta**

>> theta = [0; 0]

theta =

0

0

**#call user defined function to calculate cost function**

>> costFunctionJ(X, Y, theta)

ans =

2.3333