Ex.No:01	**Mathematical Operations**
Date:	

Aim:

To perform elementary mathematical operation: Addition, subtraction, Multiplication, Division and Exponential using octave.

Algorithm:

Step 1: Start the process

Step 2: Click start→All programs→GNU Octave(CLI)

Step 3: Declare the variable A and B to assign a value

Step 4: Using mathematical operators perform the operations as

c=a+b,d=a-b,e=a*b,f=a/b,g=a^b.

Step 5: Results are displayed.

Step 6: Stop the process.

Program:

octave:1> a=10;

octave:2> b=5;

Addition:

octave:3> c=a+b

Subtraction:

octave:4> d=a-b

Multiplication:

octave:5> e=a*b

Division:

Octave: 6> f=a/b

Exponential:

Octave: 7> g=a^b

Output:

c = 15

d = 5

e = 50

f = 2

g = 100000

Result:

The above program has been executed successfully .And output is verified.

Ex.No:02	**Logical Operators**
Date:	

Aim:

To perform elementary logical operators like AND, OR, NOT, XOR checking for equality.

Algorithm:

Step 1: Start the process.

Step 2: Click Start→All programs →GNU octave(CLI).

Step 3: Declare the binary values in the matrix format for the variable a and b.

Step 4: By using the logical operators perform, and(a , b) , or(a , b) , not(a) , xor(a , b) , isequal (a , b)

Step 5: The results are displayed.

Step 6: Stop the process.

Program:

octave:1> a=[0,0,1,1]

octave:2> b=[0,1,0,1]

AND:

octave:3> c=and(a,b)

OR:

octave:4> d=or(a,b)

NOT:

octave:5> e=not(a)

XOR:

octave:6> f=xor(a,b)

Equal:

octave:7> g=isequal(a,b)

Output:

a = 0 0 1 1

b = 0 1 0 1

c = 0 0 0 1

d = 0 1 1 1

e = 1 1 0 0

f = 0 1 1 0

g = 0

Result:

The above program has been executed successfully .And output is verified.

Ex.No:03	**Formatting Number and String**
Date:	

Aim:

To create, initialize, display simple variables and simple strings and use simple formatting for variables.

Algorithm:

Step 1: Start the process.

Step 2: Click start→All programs→GNU Octave(CLI).

Step 3: Declare the variable for a and perform the operations like exp(), ceil(), factorial(), round() .

Step 4: Declare the variable in string format and perform the operations like strcat , strcmp and strchr.

Step 5: The results are displayed.

Step 6: Stop the process.

Program:

```
octave:1> a=10

octave:2> b=exp(10)

octave:3> c=factorial(5)

octave:4> d=sqrt(5)

octave:5> e=ceil(10.67)

octave:7> f=round(10.88)

octave:8> f=round(10.44)

octave:9> str="raja"

octave:10> u=length(str)

octave:11>v=strcat("raja","rani")

octave:12> str1="rani"

octave:13>x=strcmp(str,str1)

octave:14> y=length(str1)

octave:15> res=strchr(str1,'r')
```

Output:

a=10

b = 22026.46579

c = 120

d = 2.2361

e = 11

f = 10

str = raja

u = 4

v = rajarani

str1 = rani

x = 0

y= 4

res = 1

Result:

The above program has been executed successfully .And output is verified.

Ex.No:04	**Program to perform Matrix Operations**
Date:	

Aim:

To perform basic operations on matrices (like addition, subtraction, multiplication) and specific rows or columns of the matrix.

Algorithm:

Step 1: Start the process

Step 2: Click start→All programs→GNU Octave(CLI).

Step 3: Declare the values for the variable A and B in the matrix format. .

Step 4: The matrix operations are like addition(+),subtraction(-), Multiplication(*), are performed.

Step 5: The results are displayed.

Step 6: Stop the process.

Program:

octave:1> A=[1,2,3;4,5,6;9,8,7]

octave:2> B=[2,3,4;6,8,9;5,6,7]

Addition:

octave:3> C=A+B

Subtraction:

octave:4> D=A-B

Multiplication:

octave:5> E=A*B

Output:

A= 1 2 3

4 5 6

9 8 7

B = 2 3 4

6 8 9

5 6 7

C = 3 5 7

10 13 15

14 14 14

D = -1 -1 -1

-2 -3 -3

4 2 0

E = 29 37 43

68 88 103

101 133 157

Result:

The above program has been executed successfully .And output is verified.

Ex.No:05	**Matrix Operations**
Date:	

Aim:

To perform other matrix operation like converting matrix data to absolute values taking the negative of matrix values, adding/removing rows/columns from a matrix , finding the maximum or minimum values in a matrix or in a row/column and finding the sum of some/all elements in a matrix.

Algorithm:

Step 1: Start the process

Step 2: Click start→All programs→GNU Octave(CLI)

Step 3: Declare the values for the variable a in matrix format.

Step 4: Find the absolute value by using abs() command.

Step 5: To calculate for loop and if condition to take that negative value of matrix.

Step 6: Add a one column and Remove a one row from a matrix.

Step 7: To find maximum or minimum values in a matrix use the functions max() and min().

Step 8: The results are displayed.

Step 9:Stop the process.

Program:

```
octave:1> a=[5,6,-7;1,-2,3;5,-4,9]
```

Absolute_value:

```
octave:2> b=abs(a)
```

Negative_value:

```
octave:3> [m,n]=size(2);
octave:4> [m,n]=size(a);
octave:5> for i=1:m
> for j=1:n
> if a(i,j)<0
> disp(a(i,j));
> end
> end
> end
```

Add:

```
octave:6> a(:,4)=[1,2,3]
```

Delete:

```
octave:7> a(1,:)=[]
```

Maximum value:

```
octave:8> max=max(max(a))
```

Minimum value:

```
octave:9> min=min(min(a))
```

Output:

a = 5 6 -7

1 -2 3

5 -4 9

Absolute vale:

b = 5 6 7

1 2 3

5 4 9

Negative value: -7

-2

-4

Add:

a= 5 6 -7 1

1 -2 3 2

5 -4 9 3

Delete:

a= 1 -2 3 2

5 6 -7 1

Maximum:

min= -4

Minimum:

max= 9

Result:

The above program has been executed successfully .And output is verified.

Ex.No:06	**Arrays**
Date:	

Aim:

To create single dimensional/multi-dimensional arrays and arrays with specific Values like array of all ones, all zeros, arrays with random values with in a range or a diagonal matrix.

Algorithm:

Step 1: Start the process

Step 2: Click start→All programs→GNU Octave(CLI)

Step 3: Initialize the values for the variables a, b, c in the matrix format.

Step 4: Use the functions ones(), rand(), randn() and diag() to perform the operations.

Step 5: Results are displayed.

Step 6: Stop the process.

Program:

```
octave:1> a=[1,2]

octave:2> b=[1,2;3,4]

octave:3> c=[1,2,3;4,5,6;7,8,9]

octave:4> d=1:4:10

octave:5> e=ones(2,2)

octave:6> f=rand(2,2)

octave:7> g=randn(2,2)

octave:8> h=diag(c)
```

Output:

a = 1 2

b = 1 2

3 4

c = 1 2 3

4 5 6

7 8 9

d = 1 5 9

e = 1 1

1 1

f = 0.049618 0.934417

0.256645 0.627032

g = 1.85213 -1.51817

-0.44230 -0.72496

h = Diagonal Matrix

1 0 0

0 5 0

0 0 9

Result:

The above program has been executed successfully .And output is verified.

Ex.No:07	**Conditional Statement And Different Type Of Loops**
Date:	

Aim:

To use conditional statement and different type of loops based on simple examples.

Algorithm:

Step 1: Start the process

Step 2: Click start→All programs→GNU Octave (CLI)

Step 3: Initialize and assign the values for the variables.

Step 4: Using **if** statement find whether the given number is odd or even.

Step 5: Using **for** loop statements, the matrix is displayed with zeros and ones.

Step 6: Using **do** statement, display the Fibonacci series.

Step 5: Similarly use the **while** statement is used to print the prime numbers.

Step 6: Display the result.

Step 7: Stop the process.

Program:

If-else

```
octave:1> x=10;
octave:2> if(rem(x,2)==0)
> printf("x is even\n");
> else
> printf("x is odd\n");
> endif
```

for

```
octave:3> n=5;
octave:4> a=zeros(n);
octave:5> for row=1:n
> for column=1:n
> if column>row
> continue;
> endif
> a(row,column)=1;
> endfor
> endfor
octave:6> disp(a);
```

do

```
octave:7> fib=ones(1,10);
octave:8> i=2;
octave:9> do
> i++;
> fib(i)=fib(i-1)+fib(i-2);
> until(i==0)
octave:10>disp(fib);
```

Switch

```
octave:11> a=7;
octave:12> switch(a)
> case{6,7}
> printf("variable is either 6 or 7\n");
> otherwise
> printf("variable is neither 6 or 7\n");
> endswitch
```

break

```
octave:13> num=103;
octave:14> div=2;
octave:15> while(1)
> if(rem(num,div)==0)
> printf("smallest divisor of %d is %d\n",num,div);
> break;
> endif
> div++;
> if(div*div>num)
> printf("%d is prime\n",num);
> break;
> endif
> endwhile
```

Output:

If-else

x is even

for

1 0 0 0 0

1 1 0 0 0

1 1 1 0 0

1 1 1 1 0

1 1 1 1 1

do

1 1 2 3 5 8 13 21 34 55

Switch

variable is either 6 or 7

break

103 is prime

Result:

The above program has been executed successfully .And output is verified.

Ex.No:08	**Type Of Charts**
Date:	

Aim:

To create various type of plots/charts like histogram, plot based on sine/cosine function based on data from a matrix , further lable different axes in a plot and data in a plot.

Algorithm:

Step 1: Start the process

Step 2: Click start→All programs→GNU Octave(CLI)

Step 3: By using the plot function , display the graph for sin and cos.

Step 4: Specify the titles and the axes values for the graph,

Step 7: The graph is displayed.

Step 8: Stop the process.

Program:

Histogram;

```
randn ("state", 1);
hist (randn (10000, 1), 30);
xlabel ("Value"); ylabel ("Count");
title ("Histogram of 10000 normally distributed random numbers");
```

sin:

```
 x = -10:0.1:10;
 plot (x, sin (x));
 xlabel ("x");  ylabel ("sin (x)");
 title ("Simple 2-D Plot");
```

cos:

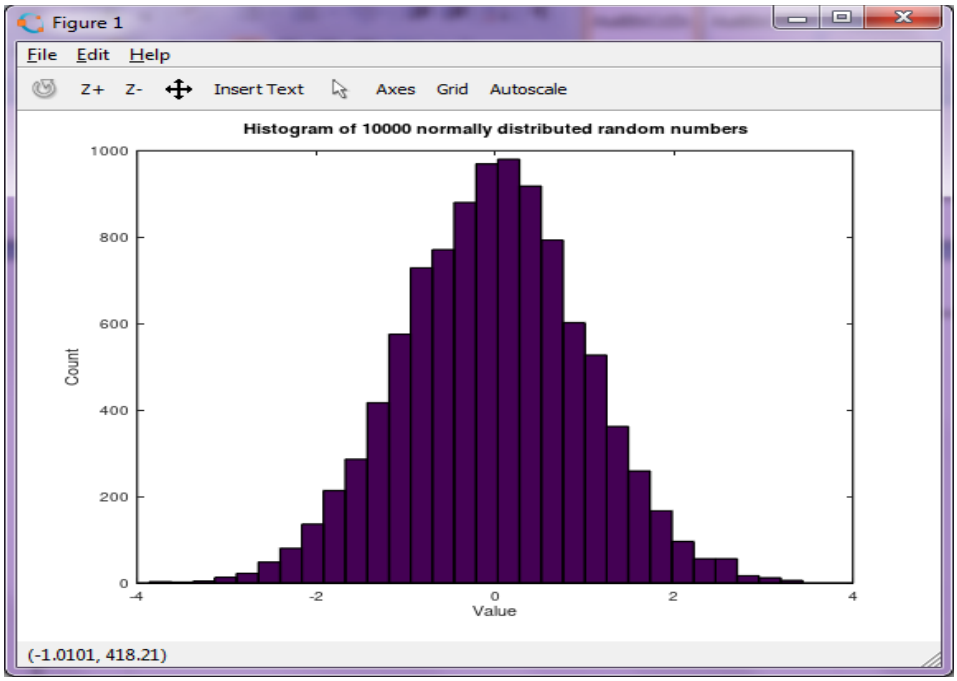
```
x = -10:0.1:10;
 plot (x, cos (x));
 xlabel ("x");  ylabel ("cos (x)");
 title ("Simple 2-D Plot");
```

Bar:

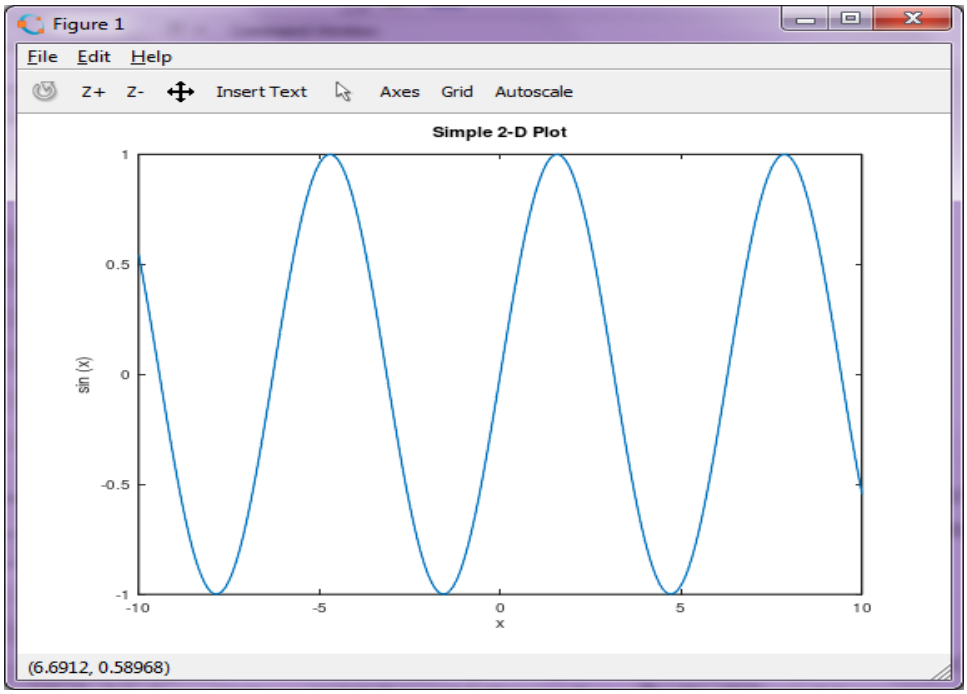
```
h = bar (rand (4, 3));
plot(axes(x),axes(y));
xlabel("axes(x)"); ylabel("axes(y)");
title("bars");
set (h(1), "facecolor", "r");
set (h(2), "facecolor", "g");
set (h(3), "facecolor", "b");
```

Output:

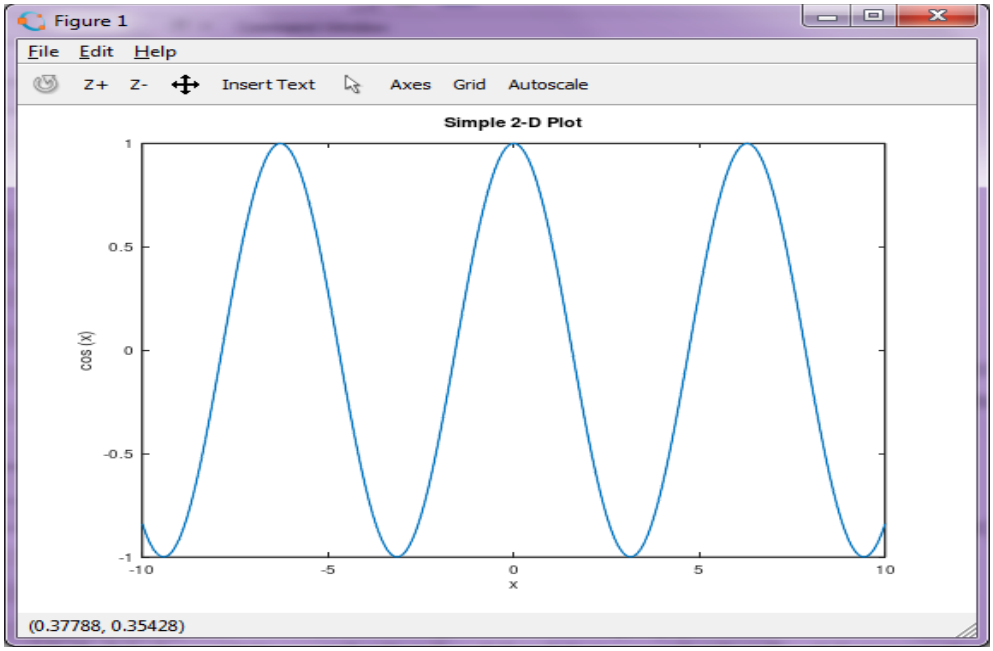
Histogram:



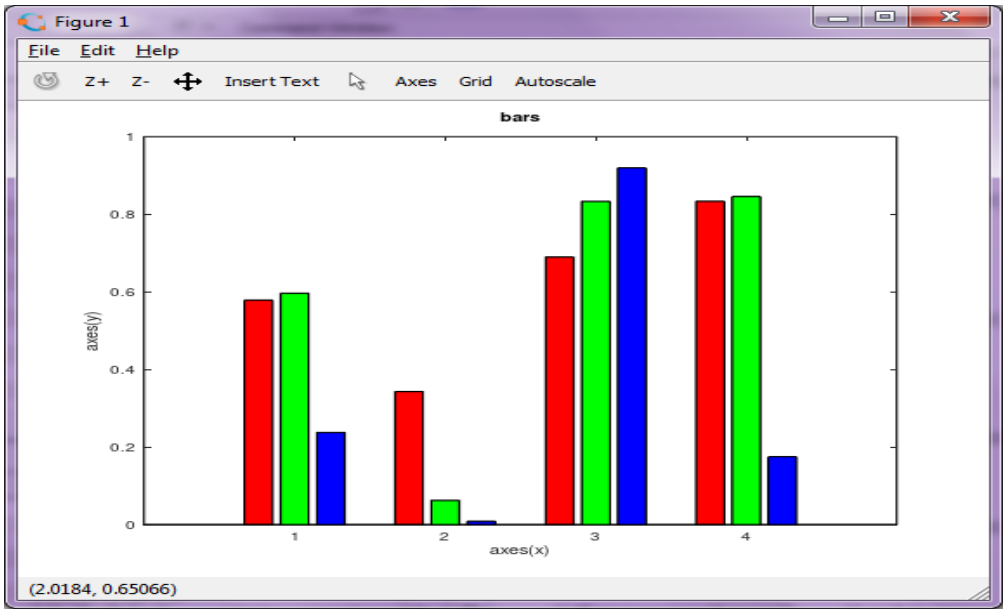
Sin:



Cos:



Bar:



Result:

The above program has been executed successfully .And output is verified.

Ex.No:09	**Subplots**
Date:	

Aim:

To generate different subplots from a given plot and color plot data.

Algorithm:

Step 1: Start the process

Step 2: Click start→ All programs→GNU Octave (GUI)

Step 3: Use the subplot function to display the subplot for sin, cos, tan and log functions.

Step 4: To like a color plot in data to set a color and basevalue.

Step 5: The graphs are displayed.

Step 6: Stop the process.

Program:

Subplots:

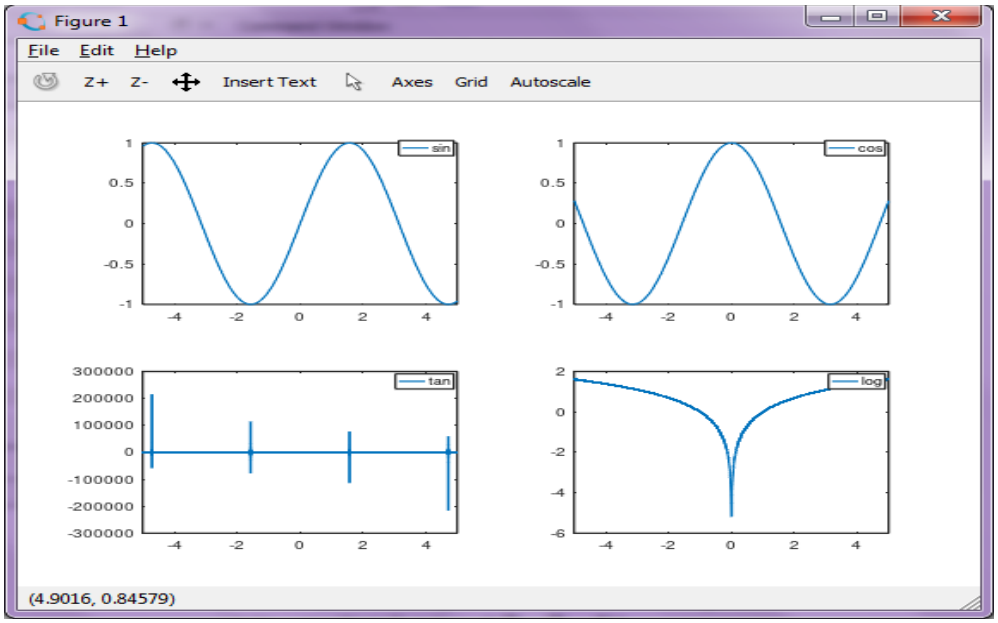
```
subplot (2, 2, 1)
 fplot (@sin, [-5, 5]);
subplot (2, 2, 2)
 fplot (@cos, [-5, 5]);
subplot (2,2,3)
 fplot (@tan,[-5,5]);
 subplot (2,2,4)
 fplot (@log,[-5,5]);
```

Color plot:

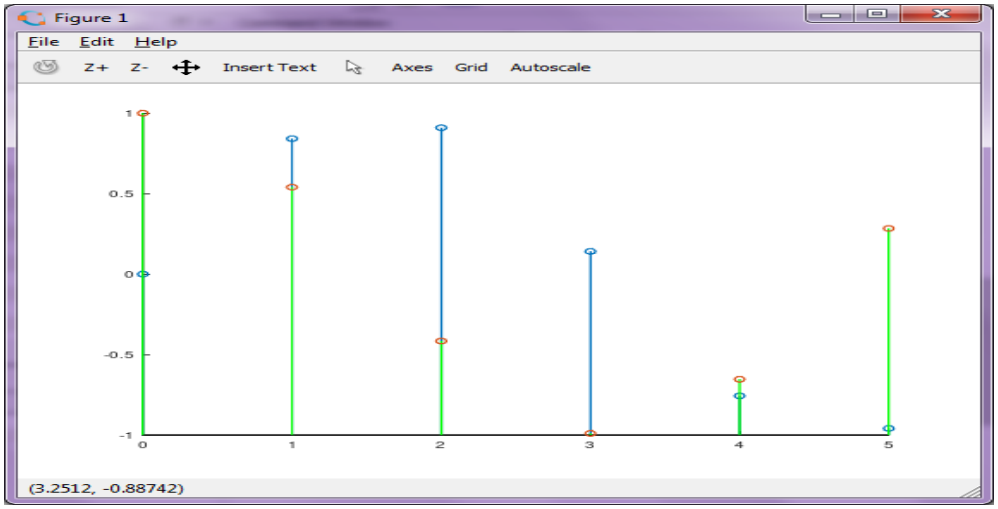
```
x = [0:5]';
 y = [sin(x), cos(x)];
h = stem (x, y);
set (h(2), "color", "g");
 set (h(1), "basevalue", -1);
```

Output:

Subplot:



Color plot:



Result:

The above program has been executed successfully .And output is verified.

Ex.No:10	**Vectorized Implementation of Simple Matrix Operations**
Date:	

Aim:

To perform vectorized implementation of simple matrix operation like finding The transpose of a matrix, adding, subtracting or multiplying two matrix.

Algorithm:

Step 1: Start the process

Step 2: Click start→ All programs→GNU Octave (GUI)

Step 3: Initialize the matrix values for the variables a and b.

Step 4: Use transpose(a), plus(a,b), minus(a,b) and mtimes(a,b) to perform the operations.

Step 5: The results are displayed.

Step 6: Stop the process.

Program:

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

Transpose:

```
transpose(a)
b=[3,4,6;2,7,8]
```

Add:

```
 i=plus(a,b)
```

Sub:

```
 j=minus(a,b)
c=3
```

Multiply:

```
 k=mtimes(a,c)
```

Output:

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

Transpose:

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

```
b =   3   4   6
      2   7   8
```

Add:

```
i =    4    6    9
       6   12   14
```

Subtraction:

```
j =  -2  -2  -3
      2  -2  -2
c =  3
```

Multiplication:

```
k =    3    6    9
      12   15   18
```

Result:

The above program has been executed successfully .And output is verified.

Ex.No:11	**Size of The Matrix**
Date:	

Aim:

To use command to compute the size of a matrix. size of a matrix, size/length of a particular row/column.

Algorithm:

Step 1: Start the process

Step 2: Click start→ All programs→GNU Octave (GLI)

Step 3: Initialize the matrix values for the variable a.

Step 4: Use size() , rows(), columns() to perform the operations in the matrix.

Step 5: Results are displayed.

Step 6: Stop the process.

Program:

octave:1> a=[1,2,3;4,5,6;2,3,4]

Size of matrix:

octave:2> size(a)

Size of rows:

octave:3> rows(a)

Size of columns:

octave:4> columns(a)

Output:

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

Size of matrix:

```
ans =   3   3
```

Size of rows:

```
ans =  3
```

Size of columns:

```
ans =  3
```

Result:

The above program has been executed successfully .And output is verified.

Ex.No:12	**Linear Regression**
Date:	

Aim:

To implement the linear regression problem.

Linear regression is a kind of statistical analysis that attempts to show a relationship between two variables. Linear regression looks at various data points and plots a trend line. Linear regression can create a predictive model on apparently random data, showing trends in data, such as in cancer diagnoses or in stock prices.

There are a number of ways to calculate linear regression. One of the most common is the ordinary least-squares method, which estimates unknown variables in the data, which visually turns into the sum of the vertical distances between the data points and the trend line

Algorithm:

Step 1: Start the process

Step 2: Click start→ All programs→GNU Octave (GUI)→Editor.

Step 3: Initialize the values for the variables.

Step 4: Load the data in the file `'ex1data1.txt'`

Step 5: Plot the data using the plot () function.

Step 6: To perform `Gradient descent add a column of ones to the data.`

Step 7: Change the theta and alpha values by using the equation

```
theta = theta - (X' * (X * theta - y)) * (alpha / m);
```

Step 8: Plot the training data in the graph.

Step 9: Lastly plot the graph for the predicted data.

Step 10: Stop the process.

Program:

```
%X refers to the population size in 10,000s
%y refers to the profit in $10,000s
%
%%initialization
clear;close all;clc
%%==========================Part1:
Loading and Plottting
%====================
%Load Data
fprintf('Loading Data...\n');
%
data=load('C:\Users\User.TERMINAL1\data.txt');
x=data(:,1);y=data(:,2);
m=length(y);
%number of training examples
fprintf('Loading complete.Press enter to continue.\n');
pause;
%Plot Data
fprintf('Plotting Data....\n');
%
figure;% open a new figure window
plot(X,y,'rx','MarkerSize',5);
ylable('Profit in $10,000');
xlable('Population of city in 10,000');
```

```
fprintf('Plotting complete.Press enter to continue.\n');
pause;%if you extra as a function,then call function as plotData(X,y);
%
%%=====================Part2;
Gradient descent ====================
fprintf('Running Gradient Descent...\n');
%
X=[ones(m,1),data(:,1)];%Add a column of ones to x
theta=zeros(2,1);%initialize fitting parameters
iteration=1500;%some gradient descent setting
alpha=0.01;%some gradient descent setting
% run gradient descent
for iter=1:iterations
  theta=theta-(X'*(X*theta-y))*(alpha/m);
 end
%otherwise run as a function,i.e theta=gradientDescent(X,y,theta,alpha,iteration);
%print theta to screen
 fprintf('Theta found by gradient descent:');
 fprintf('%f %f \n',theta(1),theta(2));
 %Plot the linear fit
 fprintf('\n Applying Gradient Descent least cost regression...\n');
 hold on:%keep previous plot visible
 plot(X(:,2),X*theta,'-');
 lengend('Training data','Linear regression');
 hold off %don't overlay any more plots on this figure
```

```
fprintf('Press enter to continue.\n');
pause;
%%==============================Part3:
Sample input tests
%=================
%Predict values for population size of35,000 and70,000
fprintf('Running example test of two population input.Press enter to continue.\n');
predict1=[1,3.5]*theta;
fprintf('For population = 35,000,we predict a profit of %f\n',....predict1*10000);
predict2=[1,7]*theta;
fprintf('For population = 70,000,we predict a profit of %f\n',....predict2*10000);
fprintf('\nComplete\n\n');
endfor
```

Data Below to cut n' paste and save as data1.txt:

```
6.1101,17.592
5.5277,9.1302
8.5186,13.662
7.0032,11.854
5.8598,6.8233
8.3829,11.886
7.4764,4.3483
8.5781,12
6.4862,6.5987
5.0546,3.8166
5.7107,3.2522
14.164,15.505
5.734,3.1551
8.4084,7.2258
```

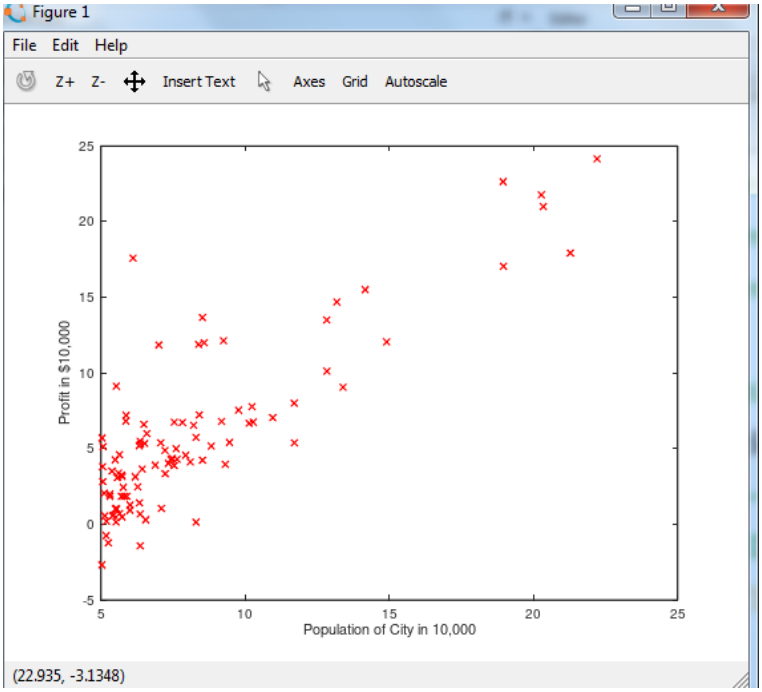
5.6407,0.71618
5.3794,3.5129
6.3654,5.3048
5.1301,0.56077
6.4296,3.6518
7.0708,5.3893
6.1891,3.1386
20.27,21.767
5.4901,4.263
6.3261,5.1875
5.5649,3.0825
18.945,22.638
12.828,13.501
10.957,7.0467
13.176,14.692
22.203,24.147
5.2524,-1.22
6.5894,5.9966
9.2482,12.134
5.8918,1.8495
8.2111,6.5426
7.9334,4.5623
8.0959,4.1164
5.6063,3.3928
12.836,10.117
6.3534,5.4974
5.4069,0.55657
6.8825,3.9115
11.708,5.3854
5.7737,2.4406
7.8247,6.7318
7.0931,1.0463
5.0702,5.1337
5.8014,1.844
11.7,8.0043
5.5416,1.0179
7.5402,6.7504
5.3077,1.8396
7.4239,4.2885
7.6031,4.9981
6.3328,1.4233
6.3589,-1.4211
6.2742,2.4756
5.6397,4.6042
9.3102,3.9624
9.4536,5.4141

8.8254,5.1694
5.1793,-0.74279
21.279,17.929
14.908,12.054
18.959,17.054
7.2182,4.8852
8.2951,5.7442
10.236,7.7754
5.4994,1.0173
20.341,20.992
10.136,6.6799
7.3345,4.0259
6.0062,1.2784
7.2259,3.3411
5.0269,-2.6807
6.5479,0.29678
7.5386,3.8845
5.0365,5.7014
10.274,6.7526
5.1077,2.0576
5.7292,0.47953
5.1884,0.20421
6.3557,0.67861
9.7687,7.5435
6.5159,5.3436
8.5172,4.2415
9.1802,6.7981
6.002,0.92695
5.5204,0.152
5.0594,2.8214
5.7077,1.8451
7.6366,4.2959
5.8707,7.2029
5.3054,1.9869
8.2934,0.14454
13.394,9.0551
5.4369,0.61705

Loading Data ...

Loading complete. Press enter to continue.

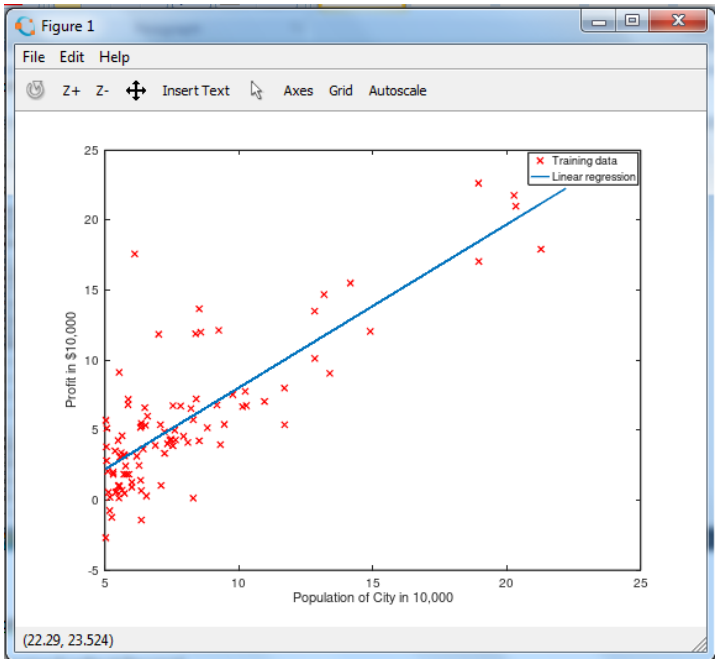
Plotting Data ...



Plotting complete. Press enter to continue.

Running Gradient Descent ...

Theta found by gradient descent: -3.630291 1.166362



Applying Gradient Descent least cost regression ...

Press enter to continue.

Running example test of two population inputs. Press enter to continue.

For population = 35,000, we predict a profit of 4519.767868

For population = 70,000, we predict a profit of 45342.450129

Complete

Result:

The above program has been executed successfully. And output is verified.

Ex. No:13 Date:	Multiple Linear Regression

Aim:

Source Code

```
clear all;
clc;
close all;
%% Load Data and Initialize Variables
fprintf('Loading data and initializing variables');
t = cputime;
data = load('dataHousePrices.txt');
X = data(:, 1:2); % Features
y = data(:, 3); % Housing Prices
m = length(y); % Number of training examples
d = size(X,2); % Number of features.
theta = zeros(d+1,1); % Initialize thetas to zero.
alpha = 0.03; % Learning rate
numIters = 1000; % How long gradient descent should run for
fprintf('...done\n');

%% Calculate Theta from Normal Equation
% This serves as a good reference to make sure that you are getting the
% right price in the end. Note for Normal equation we need to add
% the column of 1's to X. However we do not need to normalize the
features.
% I will be using XNormEqn to store X in order to avoid it replacing
% the X used in gradient descent.
fprintf('Calculating theta via normal equation');
XNormEqn = [ones(m,1) X];
thetaNormEqn = NormalEquation(XNormEqn,y);

fprintf('...done\n');


%% Feature Normalization
fprintf('Normalizing Features for gradient descent');
[X, mu, stddev] = featureNormalize(X);
fprintf('...done\n');

%% Compute the Cost Function
fprintf('Calculating theta via gradient descent');
X = [ones(m,1) X]; % Add a col of 1's for the x0 terms
```

```
[theta, CostHistory] = gradientDescent(X, theta, y, alpha, numIters);
fprintf('...done\n');

%% Prdeiction of house prices

fprintf('Predicting the price of the following house\n');
fprintf('Sq ft = , No of bedrooms = \n');
eg1 = [1 1320 2];
disp(eg1(:,2:end))
fprintf('Price via Normal Equation:');
price = (eg1)*thetaNormEqn;
disp(price);
clear price;
fprintf('\n');
fprintf('Price via Gradient Descent:');
price = predictPrice(theta,eg1,mu,stddev);
disp(price);
fprintf('Total time in seconds taken to run program\n');
disp(cputime-t);



function thetaNormEqn = NormalEquation(XNormEqn,y)
% This calculates the values of theta using the normal equation

thetaNormEqn = pinv((XNormEqn')*XNormEqn)*XNormEqn'*y;

end


function [XNorm, mu, stddev] = featureNormalize(X)
% This function provides feature normalization by taking in the input X
and
% calculating the normalized inputs along with the mean and standard
% deviation for each feature.
% X = (m x d) dimensions
% mean = (1 x d)
% stddev = (1 x d)

% Declare variables
XNorm = X;
mu = zeros(1, size(X, 2));
stddev = zeros(1, size(X, 2));

% Calculates mean and std dev for each feature
for i=1:size(mu,2)
    mu(1,i) = mean(X(:,i));
    stddev(1,i) = std(X(:,i));
    XNorm(:,i) = (X(:,i)-mu(1,i))/stddev(1,i);
end
end


function CostVal = cost(X,y,theta)
```

```
% This calculates the cost value. The cost value should always be
% decreasing with each iteration.

% Initialize values
m = length(y); % number of training examples

CostVal = (1/(2*m))*sum((X*theta - y).^2);

end


function pr = predictPrice(theta, eg1, mu, stddev)

% This function is used to predict the house price for a particular
example
% via the theta's obtained from gradient descent

% First need to normalize the values of the features
eg1 = eg1(:,2:end);
eg1 = ((eg1-mu)./stddev);

eg1 = [ones(size(eg1,1),1) eg1];
pr = eg1*theta;
end




function [theta, CostHistory] = gradientDescent(X, theta, y, alpha,
numIters)
% Gradient Descent is used to learn the parameters theta in order to fit a
% straight line to the points.

% Initialize values
m = length(y); % number of training examples
CostHistory = zeros(numIters, 1);
thetaLen = length(theta);
tempVal = theta; % Just a temporary variable to store theta values.

for iter=1:numIters
    temp = (X*theta - y);

    for i=1:thetaLen
        tempVal(i,1) = sum(temp.*X(:,i));
    end

    theta = theta - (alpha/m)*tempVal;

    CostHistory(iter,1) = cost(X,y,theta);

end
end
```

dataHousesprice.txt

1250 3 600000

2500 4 1000000

1000 2 500000