******************** Scilab Programs ********************

```
// Matrix Addition script file
clc
m=input("enter number of rows of the Matrix: ");
n=input("enter number of columns of the Matrix: ");
disp('enter the first Matrix')
for i=1:m
    for j=1:n
    A(i,j)=input('\');
     end
end
disp('enter the second Matrix')
for i=1:m
    for j=1:n
    B(i,j)=input('\');
     end
end
for i=1:m
    for j=1:n
    C(i,j)=A(i,j)+B(i,j);
     end
end
disp('The first matrix is')
disp(A)
disp('The Second matrix is')
disp(B)
disp('The sum of the two matrices is')
disp(C)
```

Scilab Console

```
enter number of rows of the Matrix: 2
enter number of columns of the Matrix: 2

 enter the first Matrix
\1
\0
\-2
\2

 enter the second Matrix
\1
\6
\-1
\2

 The first matrix is

    1.    0.
  - 2.    2.

 The Second matrix is

    1.    6.
  - 1.    2.

 The sum of the two matrices is

    2.    6.
  - 3.    4.

-->
```

```
//     Matrix Addition
clc
function []=addition(m, n, A, B)
C=zeros(m,n);
C=A+B;
disp('The first matrix is')
disp (A)
disp('The Second matrix is')
disp (B)
disp('The sum of two matrices is')
disp (C)
endfunction
```

```
Scilab Console
 Warning : redefining function: addition                . Use funcprot(0) to avoid this message


-->addition(2,2,[1 2; -1 0],[1 -1 ; 3 -2])

 The first matrix is

    1.    2.
  - 1.    0.

 The Second matrix is

    1.  - 1.
    3.  - 2.

 The sum of two matrices is

    2.    1.
    2.  - 2.

-->
```

```
// matrix multiplication script file
clc
m=input("Enter number of rows of the first Matrix: ");
n=input("Enter number of columns of the first Matrix: ");
p=input("Enter number of rows of the second Matrix: ");
q=input("Enter number of columns of the second  Matrix: ");
if n==p
 disp('Matrices are conformable for multiplication')
else
   disp('Matrices are not conformable for multiplication')
   break;
end
disp('enter the first Matrix')
for i=1:m
```

```
    for j=1:n
    A(i,j)=input('\');
     end
end
disp('enter the second Matrix')
for i=1:p
    for j=1:q
    B(i,j)=input('\');
     end
end
C=zeros(m,q);
for i=1:m
  for j=1:q
      for k=1:n
 C(i,j)=C(i,j)+A(i,k)*B(k,j);
     end
end
end
disp('The first matrix is')
disp(A)
disp('The Second matrix is')
disp(B)
disp('The product of the two matrices is')
disp(C)
```

Scilab Console

```
Enter number of rows of the first Matrix: 1
Enter number of columns of the first Matrix: 2
Enter number of rows of the second Matrix: 2
Enter number of columns of the second  Matrix: 1

 Matrices are conformable for multiplication

 enter the first Matrix
\1
\0

 enter the second Matrix
\-1
\2

 The first matrix is

    1.    0.    2.
    2.    0.    6.

 The Second matrix is

  - 1.    2.
    2.    0.
    0.    0.

 The product of the two matrices is

  - 1.

-->
```

```
// Matrix Multiplication
clc
function [ ] = multiplication(m, n, p, q, A, B)
C=zeros(m,n);
if n==p
disp('Matrices are conformable for multiplication')
else
disp('Matrices are not conformable for multiplication')
break;
end
C=A*B
disp('The first matrix is')
disp (A)
disp('The Second matrix is')
disp (B)
disp('The multiplication of two matrices is')
disp (C)
endfunction
```

```
Warning : redefining function: multiplication          . Use funcprot(0) to avoid this message


-->multiplication(2,1,1,2,[1;3],[3 2])

 Matrices are conformable for multiplication

 The first matrix is

    1.
    3.

 The Second matrix is

    3.    2.

 The multiplication of two matrices is

    3.    2.
    9.    6.

-->
```

```
// matrix transpose script file
clc
m=input("Enter number of rows of the Matrix: ");
n=input("Enter number of columns of the Matrix: ");
disp('Enter the Matrix')
for i=1:m
   for j=1:n
   A(i,j)=input('\');
    end
```

```
end
B=zeros(n,m);
for i=1:n
  for j=1:m
 B(i,j)=A(j,i)
  end
end
disp('Entered matrix is')
disp(A)
disp('Transposed matrix is')
disp(B)
```

```
Scilab Console

Enter number of rows of the Matrix: 2
Enter number of columns of the Matrix: 2

 Enter the Matrix
\1
\2
\8
\6

 Entered matrix is

    1.    2.    2.
    8.    6.    6.

 Transposed matrix is

    1.    8.
    2.    6.

-->
```

```
// Matrix Transpose function file
function []=transpose(m, n, A)
B=zeros(m,n);
B=A'
disp('The matrix is')
disp (A)
disp('Transposed matrix is')
disp (B)
endfunction
```

```
Scilab Console

-->transpose(2,2,[1 4; 6 3])

 The matrix is

    1.    4.
    6.    3.

 Transposed matrix is

    1.    6.
    4.    3.

-->
```

```
// Inverse of a 3 by 3 matrix using gauss jordan Method
 clc
disp('Enter a 3 by 3 matrix row-wise, make sure that diagonal elements are non -zeros')
for i=1:3
    for j=1:3
    A(i,j)=input('\');
     end
end
disp('Entered Matrix is')
disp(A)
if det(A)==0
disp('Matrix is singular, Inverse does not exist')
break;
end
//Taking the augmented matrix [A|I],
B=[A eye(3,3)]
disp('Augumented matrix is:')
disp(B)
//Making B(1,1)=1
B(1,:) = B(1,:)/B(1,1);
//Making B(2,1) and B(3,1)=0
B(2,:) = B(2,:) - B(2,1)*B(1,:);
B(3,:) = B(3,:) - B(3,1)*B(1,:);

//Making B(2,2)=1 and B(1,2), B(3,2)=0
```

```
B(2,:) = B(2,:)/B(2,2);
B(1,:) = B(1,:) - B(1,2)*B(2,:);
B(3,:) = B(3,:) - B(3,2)*B(2,:);
// Making B(3,3)=1 and B(1,3), B(2,3)=0
B(3,:) = B(3,:)/B(3,3);

B(1,:) = B(1,:) - B(1,3)*B(3,:);
B(2,:) = B(2,:) - B(2,3)*B(3,:);
disp('Augumented matrix after row operations is:')
disp(B)
B(:,1:3)=[]
disp('Inverse of the Matrix is')
disp(B)
```

```
Scilab Console
 Enter a 3 by 3 matrix row-wise, make sure that diagonal elements are non -zeros
\1
\2
\3
\-1
\2
\1
\4
\2
\1

 Augumented matrix is:

    1.    2.    3.    1.    0.    0.
  - 1.    2.    1.    0.    1.    0.
    4.    2.    1.    0.    0.    1.

 Augumented matrix after row operations is:

    1.    0.    0.    0.     - 0.2     0.2
    0.    1.    0.  - 0.25     0.55    0.2
    0.    0.    1.    0.5    - 0.3   - 0.2

 Entered Matrix is

    1.    2.    3.
  - 1.    2.    1.
    4.    2.    1.

 Inverse of the Matrix is

    0.    - 0.2     0.2
  - 0.25    0.55    0.2
    0.5   - 0.3   - 0.2
```

```
// Matrix Inverse using inbuilt functions
function []=inverse(m, A)
C=zeros(m,m);
B=det(A)
if B==0
disp('Matrix is singular, Inverse does not exist')
break;
end
C=inv(A)
disp('The matrix is')
disp (A)
disp('Inverse of given matrix is:')
disp (C)
endfunction
```

```
Scilab Console

-->inverse(2,[1,2;4,2])

 The matrix is

    1.    2.
    4.    2.

 Inverse of given matrix is:

  - 0.3333333    0.3333333
    0.6666667  - 0.1666667

-->
```

```
// Eigen Values
clc
disp('enter the Matrix')
for i=1:2
    for j=1:2
    A(i,j)=input('\');
     end
end
b=A(1,1)+A(2,2);
c=A(1,1)*A(2,2)-A(1,2)*A(2,1);
 // characteristic equation is e^2-trace(A)+ det(A)=0
 disp('The characteristic equation is:')
disp(['   e^2 + ' string(-b) '*e + ' string(c) ' = 0'])
e1=(b+sqrt(b^2-4*c))/2;
e2=(b-sqrt(b^2-4*c))/2;
if A(1,2) ~= 0
   v1 = [A(1,2); e1-A(1,1)];
   v2 = [A(1,2); e2-A(1,1)];
elseif A(2,1) ~= 0
   v1 = [e1-A(2,2); A(2,1)];
   v2 = [e2-A(2,2); A(2,1)];
else
   v1 = [1; 0];
   v2 = [0; 1];
end
              disp('First Eigen value is:');
              disp(e1)
              disp('First Eigen vector is:');
              disp (v1)
              disp('Second Eigen value is:');
              disp(e2)
              disp('Second Eigen vector is:');
```

```
        disp (v2)
```

```
Scilab Console

 enter the Matrix
\1
\2
\1
\-1

 The characteristic equation is:

!   e^2 +   0  *e +   -3   = 0  !

 First Eigen value is:

    1.7320508

 First Eigen vector is:

    2.
    0.7320508

 Second Eigen value is:

  - 1.7320508

 Second Eigen vector is:

    2.
  - 2.7320508

-->
```

```
//Program to find mean, S.D. and first r moments about mean of given grouped data
clc
n=input('Enter the no. of observations:');
disp('Enter the values of xi');
for i=1:n
    x(i)=input('\');
end;
disp('Enter the corresponding frequences fi:')
sum=0;
for i=1:n
    f(i)=input('\');
    sum=sum+f(i);
end;
r=input('How many moments to be calculated:');
sum1=0
for i=1:n
    sum1=sum1+f(i)*x(i);
end
A=sum1/sum;    //Calculate the average
printf('Average=%f\n',A);
for j=1:r
    sum2=0;
    for i=1:n
        y(i)=f(i)*(x(i)-A)^j;
```

```
        sum2=sum2+y(i);
    end
  M(j)=(sum2/sum);    //Calculate the moments
 printf('Moment about mean M(%d)=%f\n',j,M(j));
end
sd=sqrt(M(2));      //Calculate the standard deviation
printf('Standard deviation=%f\n',sd);
```

```
Scilab Console
Enter the no. of observations:9

 Enter the values of xi
\0
\1
\2
\3
\4
\5
\6
\7
\8

 Enter the corresponding frequences fi:
Warning : redefining function: sum                    . Use funcprot(0) to avoid this message

\1
\8
\28
\56
\70
\56
\28
\8
\1
How many moments to be calculated:4
Average=4.000000
Moment about mean M(1)=0.000000
Moment about mean M(2)=2.000000
Moment about mean M(3)=0.000000
Moment about mean M(4)=11.000000
Standard deviation=1.414214

-->
```

```
// program to find mean, mode, median, moments, skewness and kurtosis of linear data
clc
function []=moments(A)
B=gsort(A);
n = length(B);
meanA = sum(B)/n;
if pmodulo(n,2)==0
medianA =((B(n/2)+B(n/2 +1)))/2;
else medianA = B((n+1)/2);
end
C = diff(B)
//Y= diff(X) calculates differences between adjacent elements of X along the first array dimension whose size does not equal 1:
 //If X is a vector of length m, then Y = diff(X) returns a vector of length m-1. The elements of Y are the differences between adjacent elements of X.
            //Y = [X(2)-X(1) X(3)-X(2) ... X(m)-X(m-1)]
D = find(C) //D = find(C) finds the idices(positions), where value is non zero
E = diff(D)
[m k] = max(E) // maximum 'm' at kth position
modeA = B(D(k)+1)
printf('Mean of the given data is : %f  \n\n', meanA);
```

```
printf('Median of the given data is : %f  \n\n', medianA);
printf('Mode of the given data is : %f \n\n', modeA);
printf('First moment about the mean(M1)= %f \n\n', 0);
for i=1:n
X(i)=A(i)-meanA;
 end
M2 = sum(X.*X)/n;
M3 = sum(X.*X.*X)/n;
M4 = sum(X.*X.*X.*X)/n;
printf('Second moment about the mean(M2)= %f \n\n', M2);
printf('Third moment about the mean(M3)= %f \n\n', M3);
printf('Fourth moment about the mean(M4)= %f \n\n', M4);
sd= sqrt (M2);
printf('Standard deviation: %f  \n\n', sd);
Csk= (meanA - modeA)/sd;
printf('Coefficient of skewness: %f  \n\n', Csk);
Sk= (M3)^2/(M2)^3;
printf('Skewness: %f  \n\n', Sk);
Kur= M4/(M2)^2;
printf('Kurtosis: %f  \n\n', Kur);
endfunction
```

Execution:

```
Scilab Console
 Warning : redefining function: moments                 . Use funcprot(0) to avoid this message


-->moments([1 3 6 3 -2 6 8 ])
Mean of the given data is : 3.571429

Median of the given data is : 3.000000

Mode of the given data is : 6.000000

First moment about the mean(M1)= 0.000000

Second moment about the mean(M2)= 9.959184

Third moment about the mean(M3)= -10.688047

Fourth moment about the mean(M4)= 208.811329

Standard deviation: 3.155817

Coefficient of skewness: -0.769554

Skewness: 0.115645

Kurtosis: 2.105264


-->
```

Scilab plotting

The generic 2D multiple plot is

plot2di(x,y,<options>)

- index of plot2d : i= none,2,3,4.

  For the different values of i we have:

  i=none : piecewise linear/logarithmic plotting

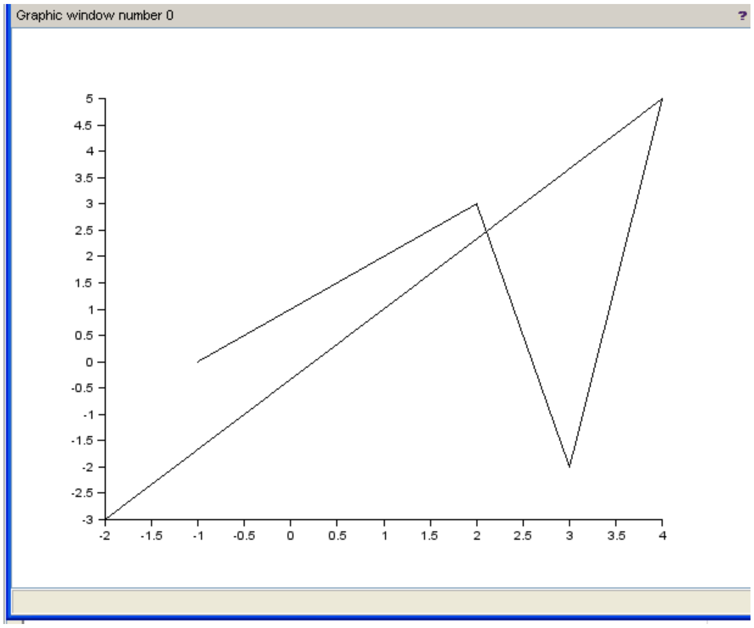
  i=2 : piecewise constant drawing style

  i=3 : vertical bars

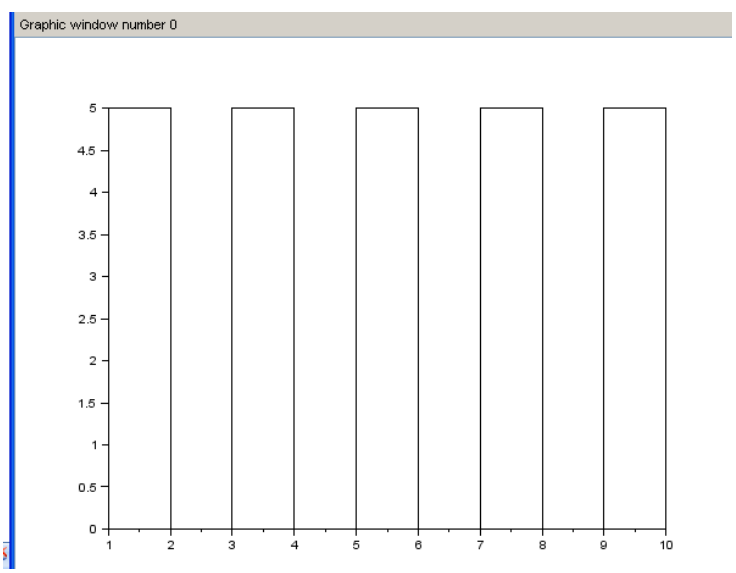
  i=4 : arrows style (e.g. ode in a phase space)

```
//Specifier Color
//r        Red
//g        Green
//b        Blue
//c        Cyan
//m        Magenta
//y        Yellow
//k        Black
//w        White
//Specifier Marker Type
//+        Plus sign
//o        Circle
//*        Asterisk
//.        Point
//x        Cross
//'square' or 's'    Square
//'diamond' or 'd' Diamond
//^        Upward-pointing triangle
//v        Downward-pointing triangle
//>        Right-pointing triangle
//<        Left-pointing triangle
//'pentagram' or 'p'          Five-pointed star (pentagram)

// a simple plot
clc
x = [1 -1 2 3 4 -2 ];
y = [2 0 3 -2 5 -3 ];
```

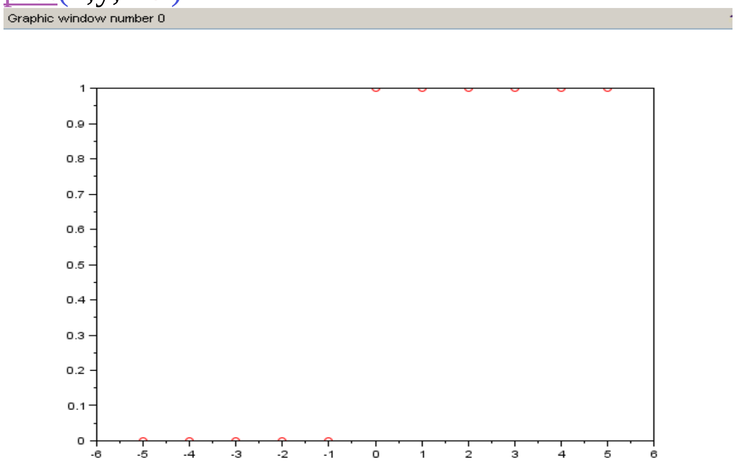
```
plot2d(x,y)
```



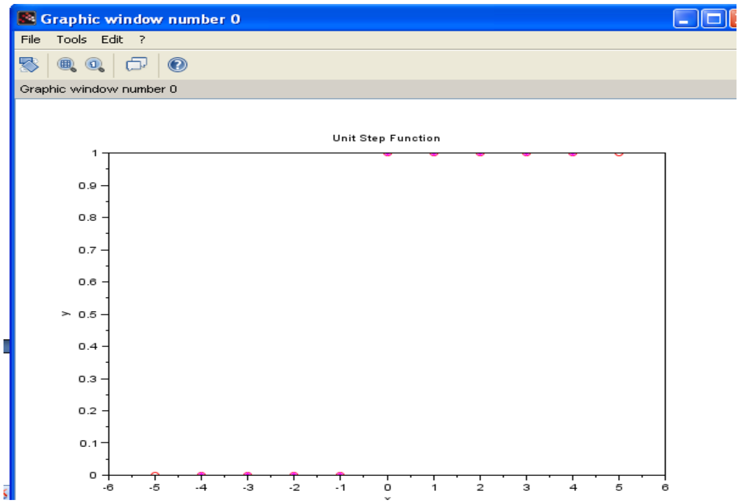

```
// generation of square wave
clc
x = [1 2 3 4 5 6 7 8 9 10 ];
y = [5 0 5 0 5 0 5 0 5 0 ];
plot2d2(x,y)
```



```
// unit step function
clc
x = [-1 -2 -3 -4 -5 0 1 2 3 4 5  ];
y = [0 0 0 0 0 1 1 1 1 1 1 ];
plot(x,y, 'ro')
```




```
// program to plot unit step function
function y=unitstep2(x)
y(find(x < 0)) = 0;
y(find(x >=0)) = 1;
endfunction
clc
// define your independent values in a column row
x = [-4 : 1 : 4]';
// call your previously defined function
y = unitstep2(x);
// plot
plot(x, y, 'm*')
xlabel('x');
ylabel('y');
title('Unit Step Function');
```




```
// Program to fit a straight line to given n pairs of values (x,y)
clc;clear;close;
n=input('Enter the no. of pairs of values (x,y):')
disp('Enter the values of x:')
for i=1:n
    x(i)=input('   ')
end
disp('Enter the corresponding values of y:')
for i=1:n
    y(i)=input('   ')
end
sumx=0;sumx2=0;sumy=0;sumxy=0
for i=1:n
    sumx=sumx+x(i);
    sumx2=sumx2+x(i)*x(i);
    sumy=sumy+y(i);
    sumxy=sumxy+x(i)*y(i);
end
A=[sumx n; sumx2 sumx];
B=[sumy;sumxy];
C=inv(A)*B
printf('The fitted line is y=(%g)x+(%g)',C(1,1),C(2,1))
```

Output

Enter the no. of pairs of values (x,y):5

Enter the values of x:

1

2

3

4

5

Enter the correrspronding valyes of y:

14

13

9

5

2

The fitted line is y=(-3.2)x+(18.2)

/

```
// Program of parabola fitting for given n pairs of values (x,y)

clc;
n=input('Enter the no. of pairs of values (x,y):')
disp('Enter the values of x:')
for i=1:n
    x(i)=input('   ')
end
disp('Enter the corresponding values of y:')
for i=1:n
```

```
   y(i)=input('   ')
end
sumx=0;sumx2=0;sumx3=0;sumx4=0;sumy=0;sumxy=0;sumx2y=0;
for i=1:n
   sumx=sumx+x(i);
   sumx2=sumx2+x(i)^2;
   sumx3=sumx3+x(i)^3;
   sumx4=sumx4+x(i)^4;
   sumy=sumy+y(i);
   sumxy=sumxy+x(i)*y(i);
   sumx2y=sumx2y+x(i)^2*y(i);
end
A=[sumx2  sumx  n; sumx3 sumx2  sumx; sumx4  sumx3  sumx2];
B=[sumy;sumxy;sumx2y];
C=inv(A)*B
printf('The fitted parabola is y=(%g)x^2+(%g)x+(%g)',C(1,1),C(2,1),C(3,1))
```

Output

```
Enter the no. of pairs of values (x,y):5


 Enter the values of x:

  1

  2

  3

  4

  5


 Enter the corresponding values of y:

  2

  6

  7
```

8

10

The fitted parabola is y=(-0.8)+(3.51429)x+(-0.285714)x^2

```
//Bisection method
clc
deff('y=f(x)','y=x^3+x^2-3*x-3')
a=input("enter initial interval value: ");
b=input("enter final interval value: ");
fa = f(a);   //compute initial values of f(a) and f(b)
fb = f(b);
if  sign(fa) == sign(fb)    // sanity check: f(a) and f(b) must have different signs
disp('f must have different signs at the endpoints a and b')
error
end
e=input(" answer correct upto : ");
iter=0;
printf('Iteration\ta\t\tb\t\troot\t\tf(root)\n')
while abs(a-b)>2*e
   root=(a+b)/2
   printf('   %i\t\t%f\t%f\t%f\t%f\n',iter,a,b,root,f(root))
   if f(root)*f(a)>0
      a=root
      else
    b=root
end
iter=iter+1
end
printf('\n\nThe solution of given equation is %f after %i Iterations',root,iter-1)
```

output

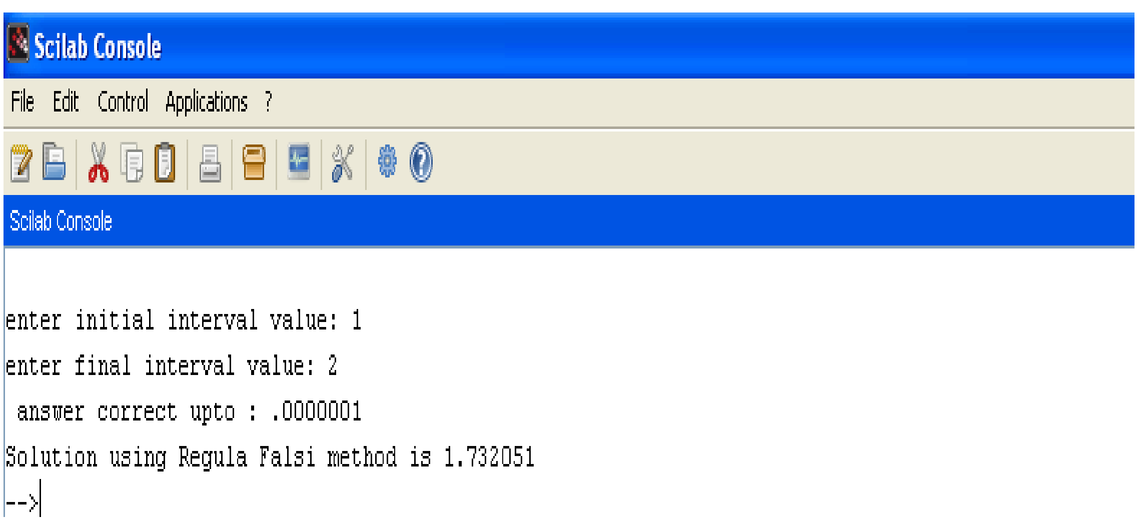
```
Scilab Console

enter initial interval value: 1
enter final interval value: 2
 answer correct upto : .0001
Iteration       a               b               root            f(root)
   0            1.000000        2.000000        1.500000        -1.875000
   1            1.500000        2.000000        1.750000        0.171875
   2            1.500000        1.750000        1.625000        -0.943359
   3            1.625000        1.750000        1.687500        -0.409424
   4            1.687500        1.750000        1.718750        -0.124786
   5            1.718750        1.750000        1.734375        0.022030
   6            1.718750        1.734375        1.726563        -0.051755
   7            1.726563        1.734375        1.730469        -0.014957
   8            1.730469        1.734375        1.732422        0.003513
   9            1.730469        1.732422        1.731445        -0.005728
   10           1.731445        1.732422        1.731934        -0.001109
   11           1.731934        1.732422        1.732178        0.001201
   12           1.731934        1.732178        1.732056        0.000046


The solution of given equation is 1.732056 after 12 Iterations
-->
```

```
// newton raphson method x(n+1)= x(n)-f(x(n))/df(x(n))
clc
deff('y=f(x)','y=x^3+x^2-3*x-3')
deff('y=df(x)','y=3*x^2+2*x-3')
x(1)=input('Enter Initial Guess:');
e=input(" answer correct upto : ");
for i=1:100
x(i+1)=x(i)-((f(x(i))/df(x(i))));
err(i)=abs((x(i+1)-x(i))/x(i));
if err(i)<e
break;
end
end
printf('the solution is %f',x(i))
```

```
Scilab Console

Enter Initial Guess:1
 answer correct upto : .00000001
the solution is 1.732051
-->
```

```
// regula falsi method
```

```
clc
deff('y=f(x)','y=x^3+x^2-3*x-3')
a=input("enter initial interval value: ");
b=input("enter final interval value: ");
e=input(" answer correct upto : ");
for i=2:100
if f(b)>f(a)
   xn=b-((f(b)*(b-a))/(f(b)-f(a)));
else
   xn=a-((f(a)*(a-b))/(f(a)-f(b)));
end
if f(b)*f(xn)<0
    a=xn;
else
    b=xn;
end
if f(a)*f(xn)<0
    b=xn;
else
    a=xn;
end
xnew(1)=0;
xnew(i)=xn;
if abs((xnew(i)-xnew(i-1))/xnew(i))<e;
    break;
    end
end
printf('Solution using Regula Falsi method is %f',xnew(i))
```

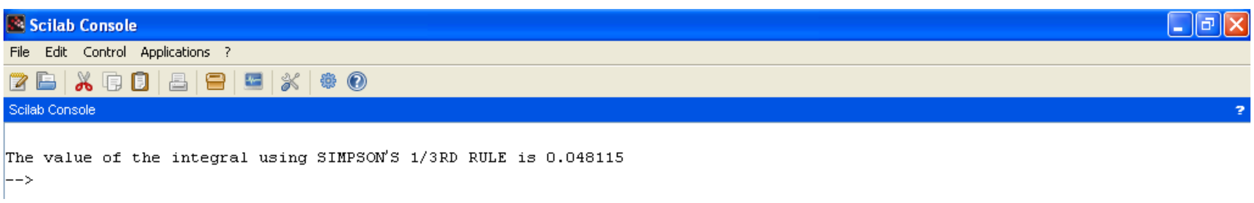


```
\\ Regressionm lines
clc
n=input('Enter the number of terms:')
  printf(' Enter the values of xi')
   for i=1:n
   x(i)=input('\');
   end
  printf(' Enter the values of yi')
   for i=1:n
   y(i)=input('\');
   end

   sumx=0;sumy=0;sumxy=0;sumx2=0;
   for i=1:n

      sumx=sumx +x(i);
      sumx2=sumx2 +x(i)*x(i);
      sumy=sumy +y(i);
      sumxy=sumxy +x(i)*y(i);

   end
   a=((sumx2*sumy -sumx*sumxy)*1.0/(n*sumx2-sumx*sumx)*1.0);
   b=((n*sumxy-sumx*sumy)*1.0/(n*sumx2-sumx*sumx)*1.0);
   printf('The line is Y=%3.3f +%3.3f X',a,b)
```

```
Scilab Console
File  Edit  Control  Applications  ?
Scilab Console

Enter the number of terms:5
 Enter the values of xi
\10
\12
\13
\9
\15
 Enter the values of yi
\11
\13
\14
\9
\12
The line is Y=5.175 +0.561 X
-->
```

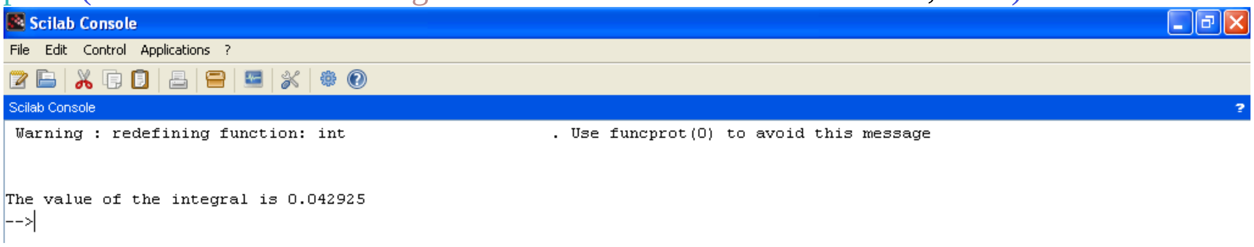
```
clc
// SIMPSON'S 1/3RD RULE
deff('y=f(x)','y=x/(x^3+10)');
```

```
x1=0;
x2=1;
n=4;
h=(x2-x1)/n;
x(1)=x1;
sum=f(x1);
for i=2:n
   x(i)=x(i-1)+h;
end
for j=2:2:n
   sum=sum+4*f(x(j));
end
for k=3:2:n
   sum=sum+2*f(x(k));
end
sum=sum+f(x2);
value=sum*h/3;
printf('\nThe value of the integral using SIMPSONS 1/3RD RULE is %f',value)
```

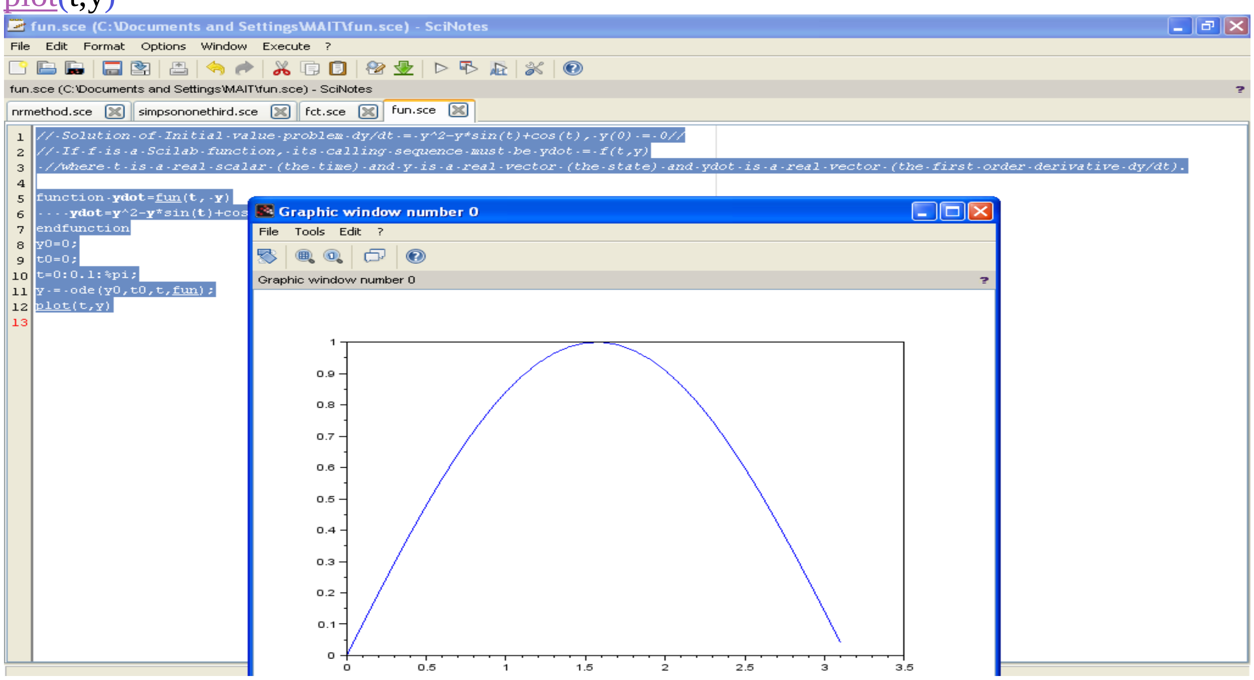



```
clc
// SIMPSON'S 3/8TH RULE
deff('y=f(x)','y=x/(x^3+10)');
x1=0;
x2=1;
n=4;
h=(x2-x1)/n;
x(1)=x1;
sum=f(x1);
for i=2:n
   x(i)=x(i-1)+h;
end
for j=2:3:n
   sum=sum+3*f(x(j));
end
for k=3:3:n
   sum=sum+3*f(x(k));
end
for l=4:3:n
```

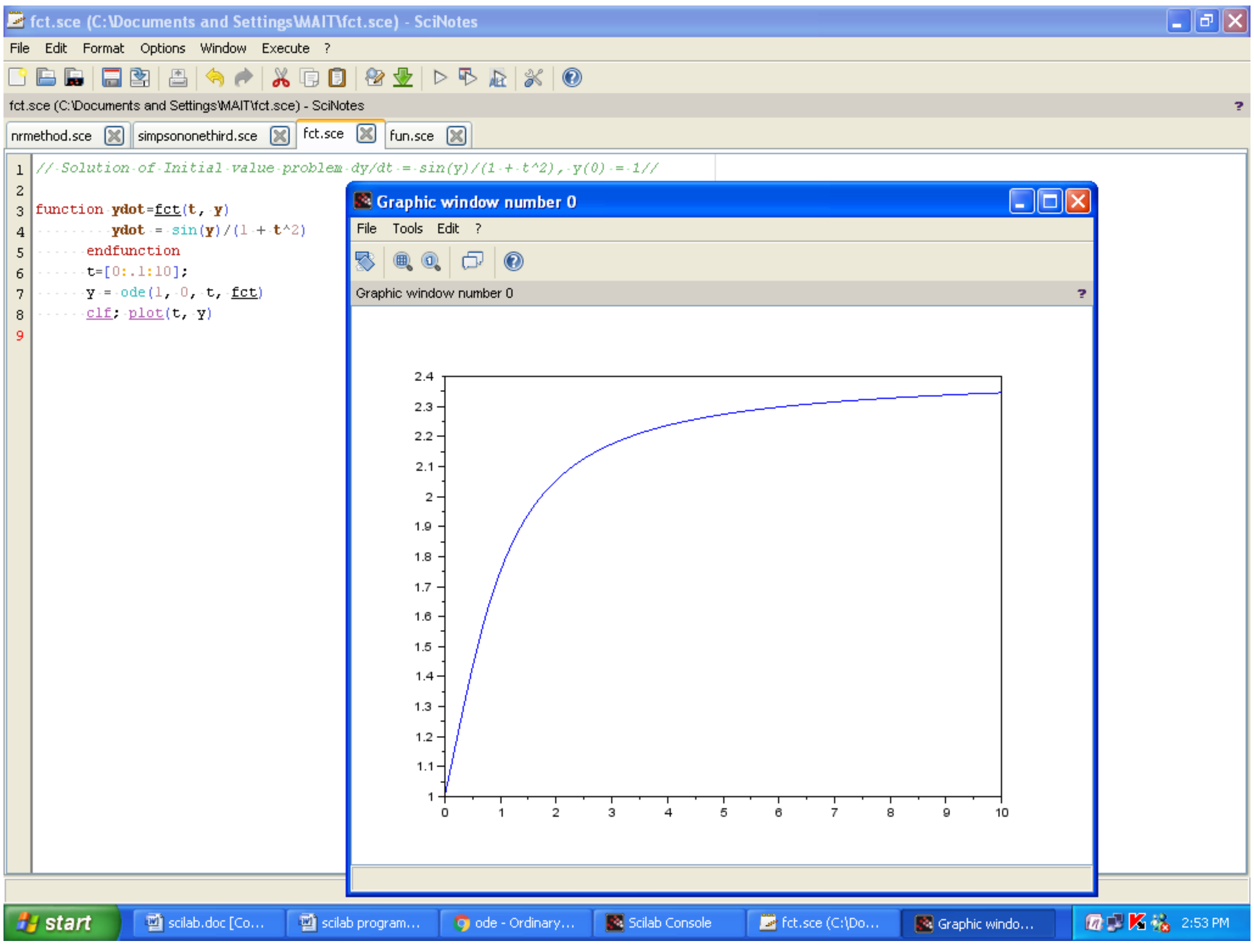
```
    sum=sum+2*f(x(l));
end

sum=sum+f(x2);
value=sum*3*h/8;
printf('\nThe value of the integral SIMPSONS 3/8th RULE  is %f ',value)
```




```
// Solution of Initial value problem dy/dt = y^2-y*sin(t)+cos(t), y(0) = 0//
// If f is a Scilab function, its calling sequence must be ydot = f(t,y)
 //where t is a real scalar (the time) and y is a real vector (the state) and ydot is a real vector (the first order derivative dy/dt).

function ydot=fun(t, y)
    ydot=y^2-y*sin(t)+cos(t)
endfunction
y0=0;
t0=0;
t=0:0.1:%pi;
y = ode(y0,t0,t,fun);
plot(t,y)
```



```
// Solution of Initial value problem dy/dt = sin(y)/(1 + t^2), y(0) = 1//

function ydot=fct(t, y)
        ydot = sin(y)/(1 + t^2)
    endfunction
    t=[0:.1:10];
    y = ode(1, 0, t, fct)
    clf; plot(t, y)
```



```
// RUNGE KUTTA METHOD
clc
function ydot=f(x, y)
        ydot =x+y^2
     endfunction

x1=0;
y1=1;

h=0.1;
x(1)=x1;
y(1) = y1;

for i=1:2
   k_1 = h*f(x(i),y(i));
   k_2 = h*f(x(i)+0.5*h,y(i)+0.5*k_1);
   k_3 = h*f((x(i)+0.5*h),(y(i)+0.5*k_2));
   k_4 = h*f((x(i)+h),(y(i)+k_3));
   k   = (1/6)*(k_1 +2*k_2 +2*k_3 +k_4);
   y(i+1)= y(i)+ k;
   printf('\n The value of y at x=%f is %f ', i*h,y(i+1))
   x(i+1)=x(1)+ i*h;
end
```