5. Write a SCILAB program to demonstrate Level Sets with Rosenbrock’s (Banana) function. (With Output)

Code: x1 = linspace(-2,2,100);

x2 = linspace(-2,2,100);

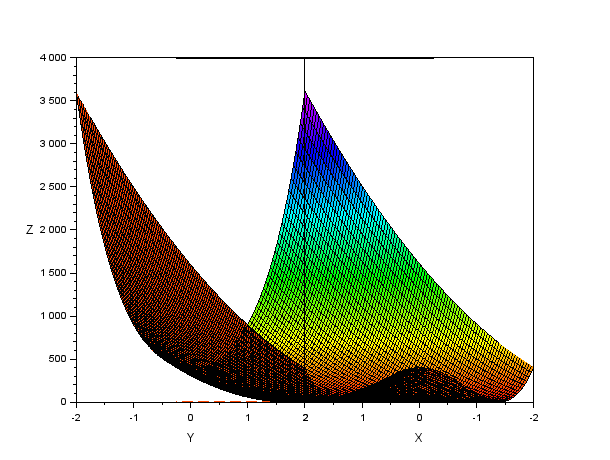
[X1,X2] = meshgrid(x1,x2);

Z = 100\*(X2-X1.^2).^2+(1-X1).^2;

f = scf();

f.color\_map = rainbowcolormap(64);

plot3d1(x1,x2,Z');



7. Write a function to find solution of Linear function AX=b in SCILAB.

Ex : x1 + 2x2 + 4x3 = 15

2x1 + x2 + 3x3 = 10

X1 + x2 + x3 = 5

**Code** : a = [1,2,4;

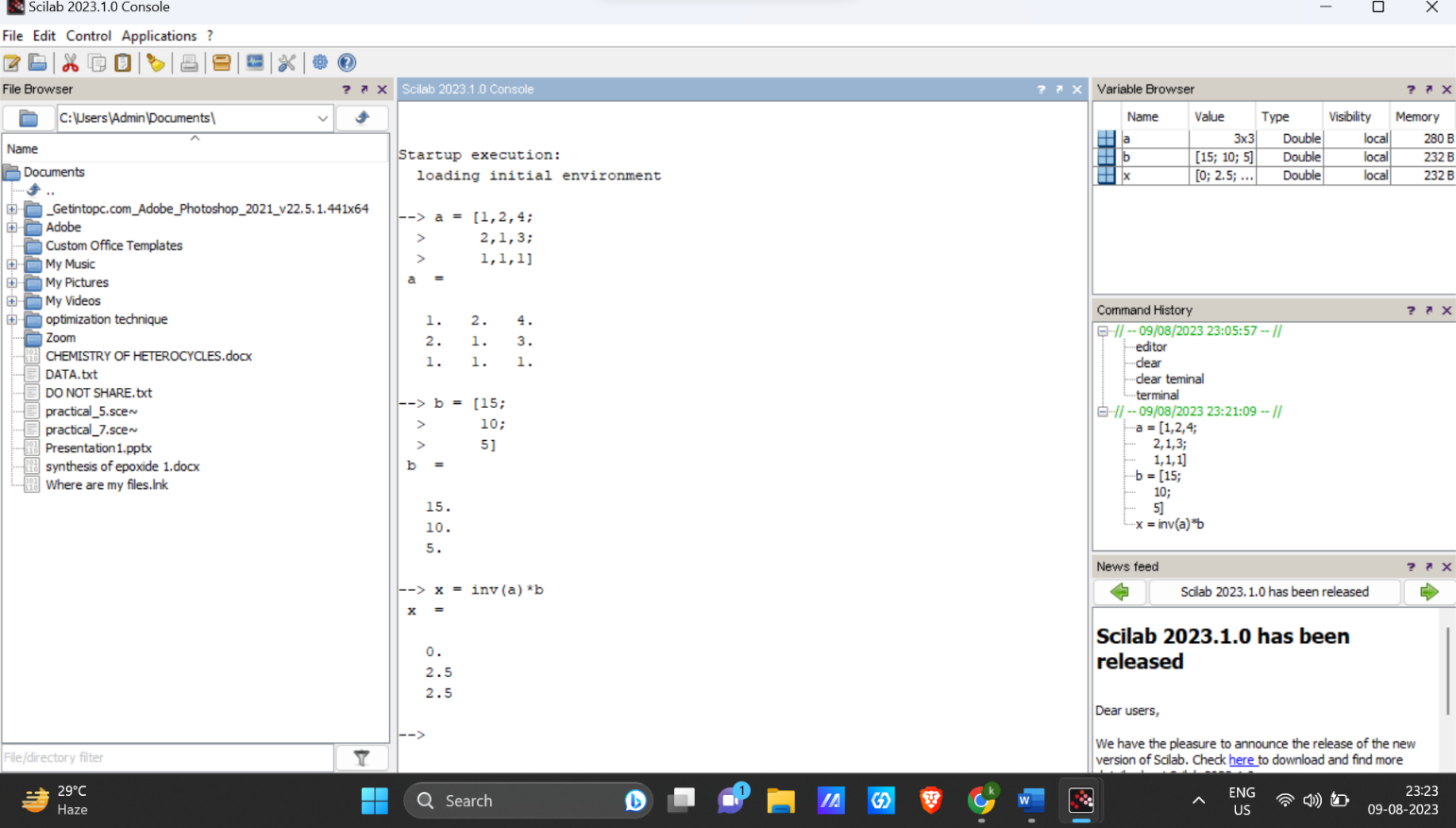
2,1,3;

1,1,1]

b = [15;

10;

5]

 x = inv(a)\*b

9.Plot LPP problems using Contour and Plot2d function in Scilab.

A contour plot is a graphical technique for representing a 3-dimensional surface by plotting constant z slices, called contours, on a 2-dimensional format

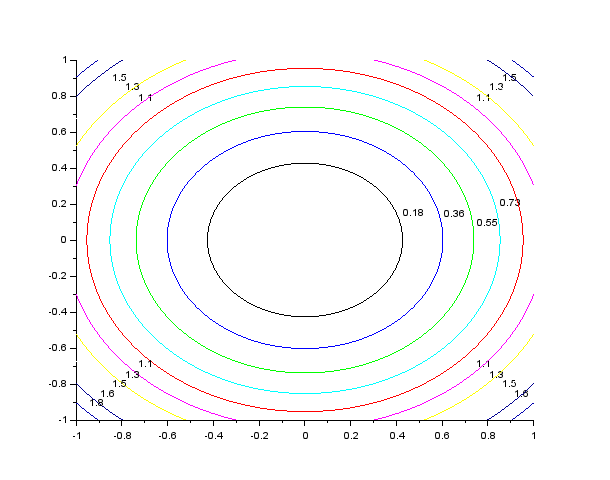
**Code :** function **f**=myquadratic2arg(**x1**, **x2**)

**f** = **x1**\*\*2 + **x2**\*\*2;

endfunction

xdata = linspace ( -1 , 1 , 100 );

ydata = linspace ( -1 , 1 , 100 );

 contour ( xdata , ydata , myquadratic2arg , 10)

13. Implement Secant Method in SCILAB.

the secant method is a root-finding algorithm that uses a succession of roots of secant lines to better approximate a root of a function f

**Code :** *//function --> y = x^3+2x^2-8*

disp("secant method")

deff('y=f(x)','y=x^3+2\*x^2-8')

a=1

b=2

for i = 1:5

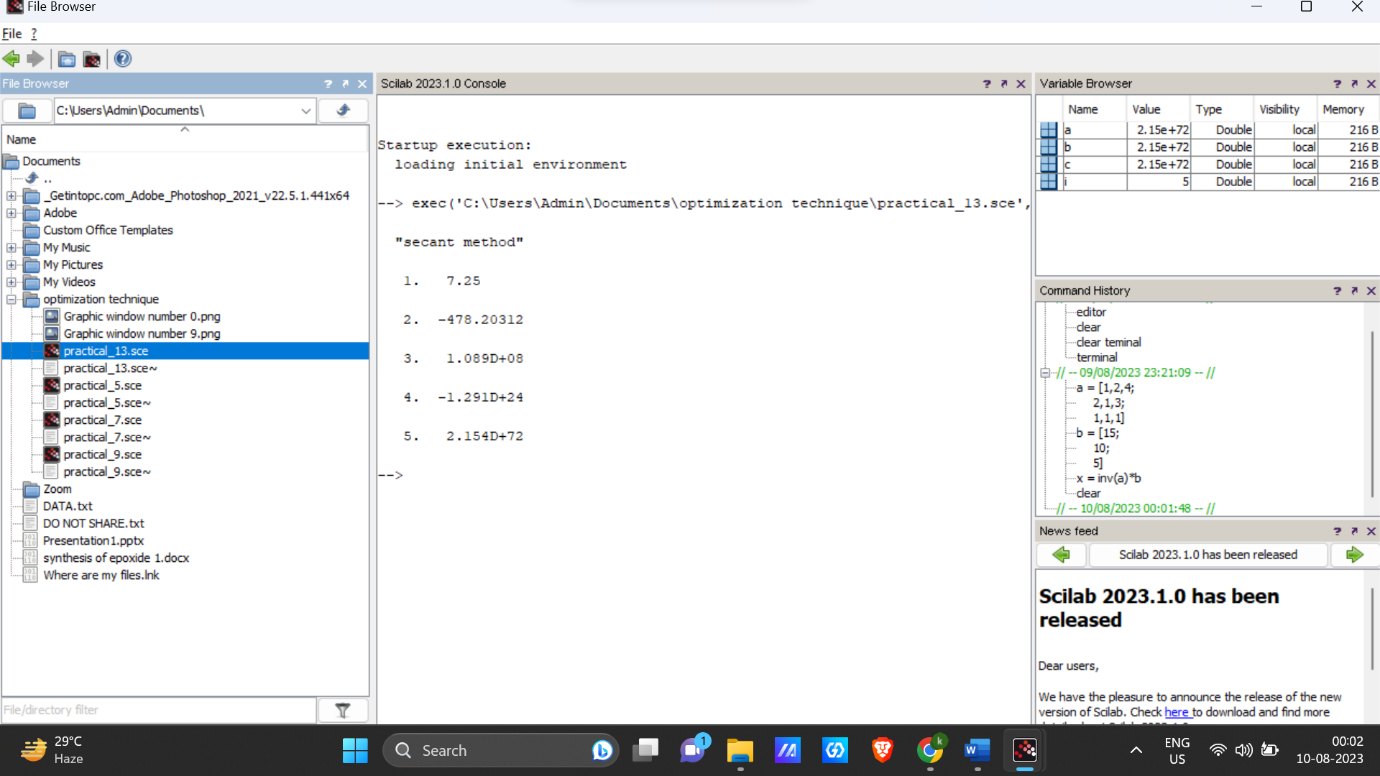
c = (a\*f(b)-b\*f(a))/f(b)-f(a)

b=c

a=c

disp([i,c])

end

\_

6. Implement a SCILAB program to find out various norms of vector and Matrix.

*// Define a vector*

vector = [1, 2, 3, 4, 5];

*// Calculate the L1 norm (sum of absolute values)*

l1\_norm = norm(vector, 1);

*// Calculate the L2 norm (Euclidean norm)*

l2\_norm = norm(vector, 2);

*// Calculate the infinity norm (maximum absolute value)*

inf\_norm = norm(vector, 'inf');

disp('Vector:');

disp(vector);

disp('L1 Norm:');

disp(l1\_norm);

disp('L2 Norm:');

disp(l2\_norm);

disp('Infinity Norm:');

disp(inf\_norm);

*// Define a matrix*

matrix = [1, 2, 3; 4, 5, 6; 7, 8, 9];

*// Calculate the Frobenius norm*

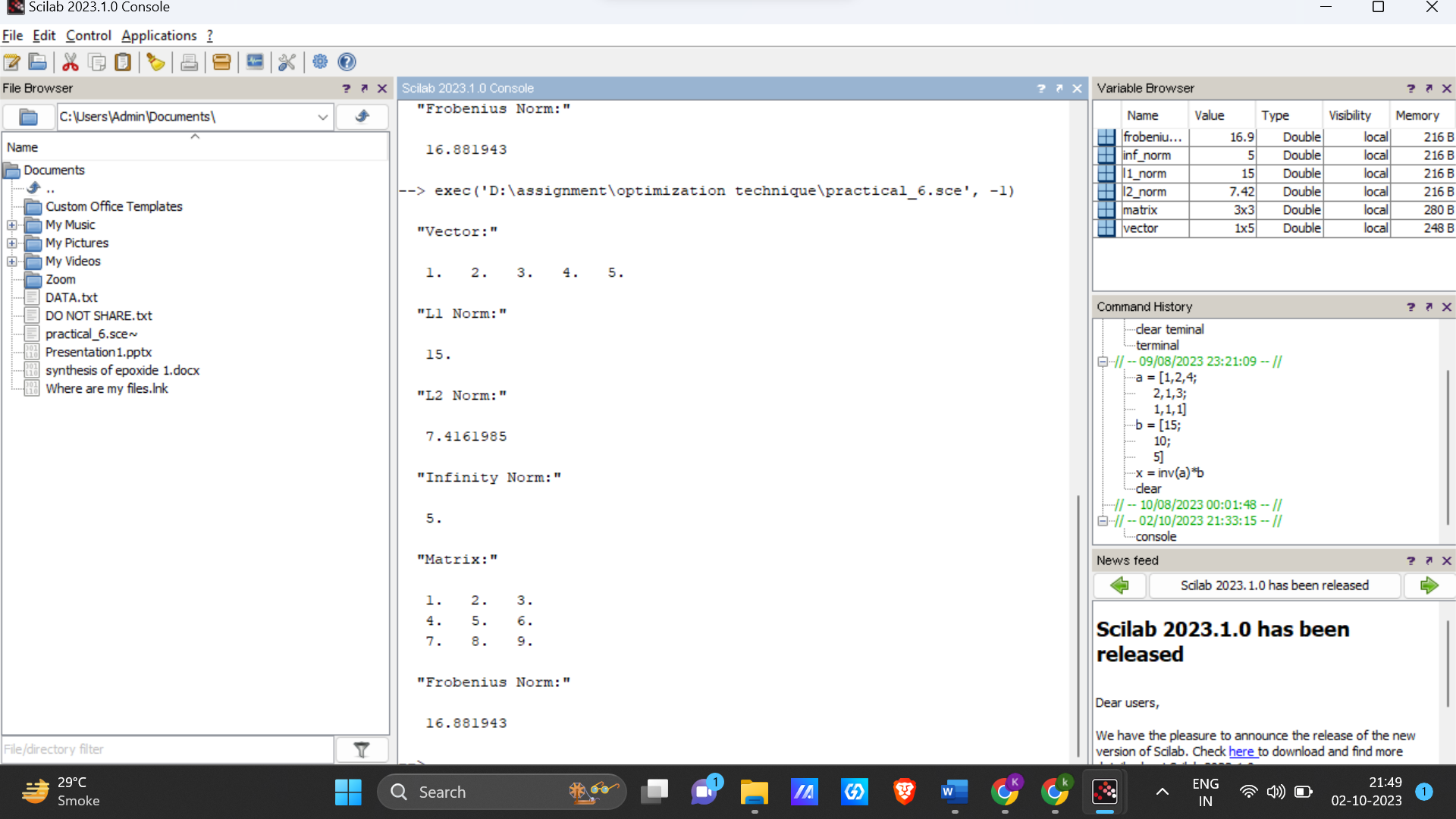
frobenius\_norm = norm(matrix, 'fro');

disp('Matrix:');

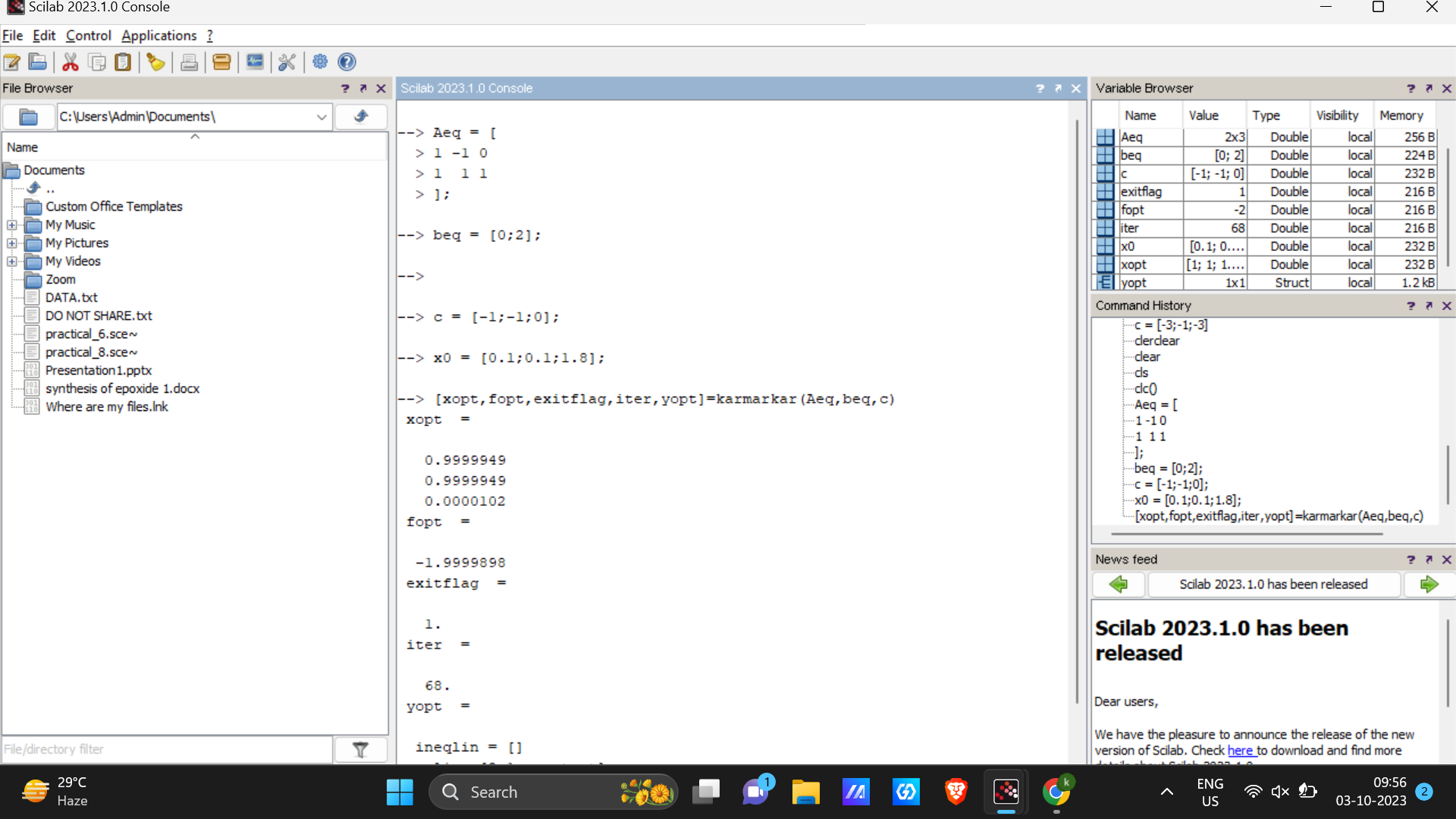
disp(matrix);

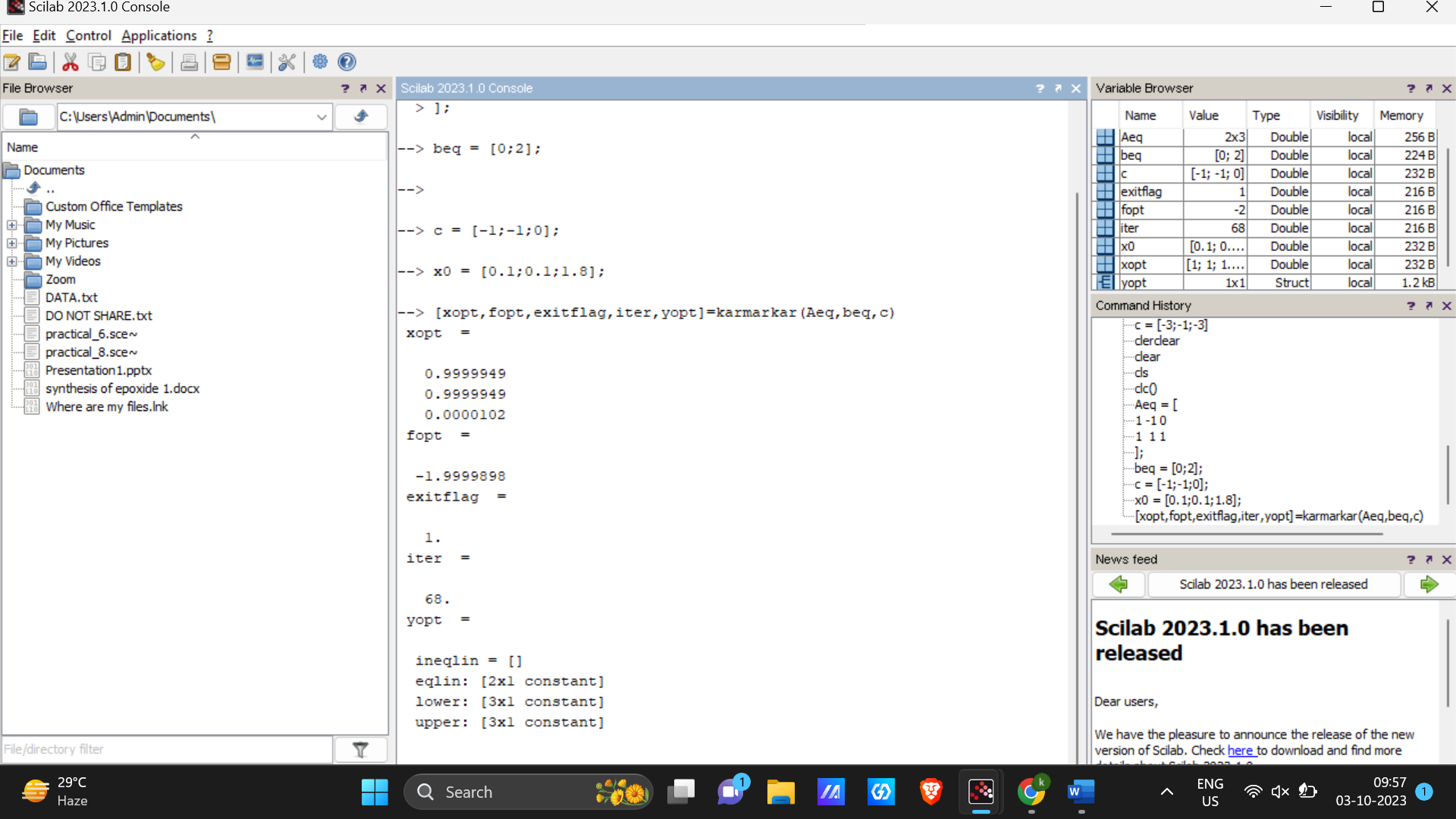
disp('Frobenius Norm:');

disp(frobenius\_norm);



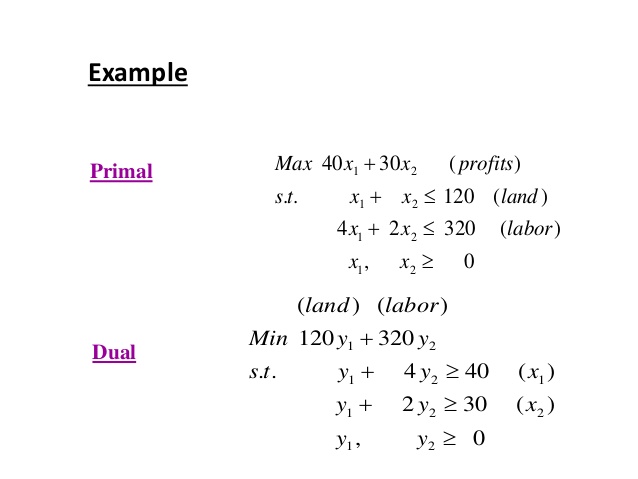
8. Solve LPP problems using Karmarkar Method in SCILAB.





10. Formulate LPP problem from given problem statement.

11. Solve and Plot Primal and Dual problem using Karmarkar Method in SCILAB.



c = [-40 -30]';

A = [

1 1

4 2

];

b = [120 320]';

[xopt,fopt,exitflag,iter,yopt]=karmarkar([],[],c,[],[],[],[],[],A,b)

c=-c;

deff('[w]=f(x,y)','w=c(1)\*x+c(2)\*y');

xx=[0:10:180];yy=[0:10:180];zz=feval(xx,yy,f);

contour(xx,yy,zz,[3000 4000 5000]);

deff('[y1]=f1(x)','y1=120-x')

deff('[y2]=f2(x)','y2=(160-(2\*x))')

xxx=[0:1:180];yy1=f1(xxx);yy2=f2(xxx);

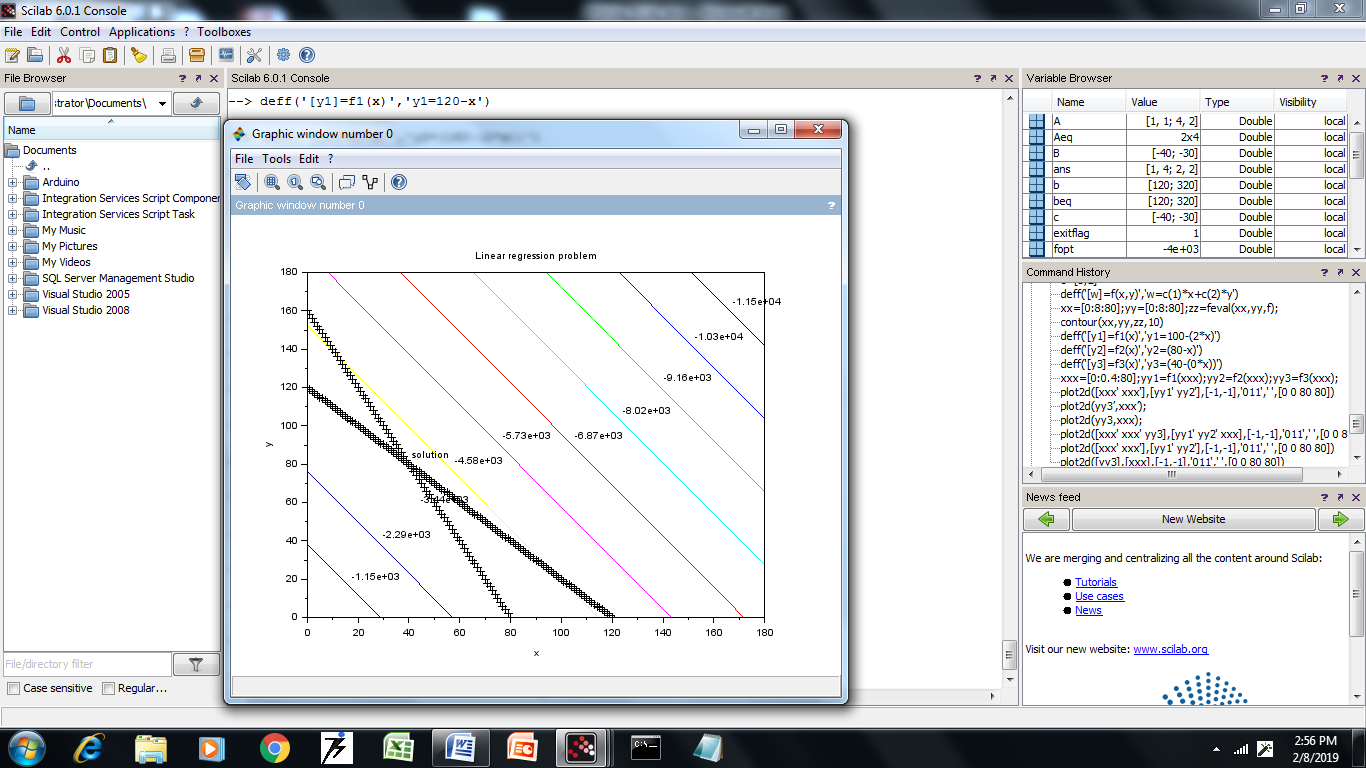
plot2d([xxx' xxx'],[yy1' yy2'],[-1,-1],'011',' ',[0 0 130 170])

xstring(xopt(1)+0.05, xopt(2)+0.05,'solution')

xtitle('Primal Dual problem','x','y')

c=-c

[xopt,fopt,exitflag,iter,yopt]=karmarkar([],[],c,[],[],[],[],[],A,b)



**Dual Problem**

c = [120 320]';

A = [

1 4

1 2

];

b = [40 30]';

A=-A

b=-b

lb=[0;0];

[xopt,fopt,exitflag,iter,yopt]=karmarkar([],[],c,[],[],[],[],[],A,b)

A=-A;

b=-b;

deff('[w]=f(x,y)','w=c(1)\*x+c(2)\*y');

xx=[0:1:40];yy=[0:1:40];zz=feval(xx,yy,f);

contour(xx,yy,zz,[3000 4000 5000]);

deff('[y1]=f1(x)','y1=10-(x/4)')

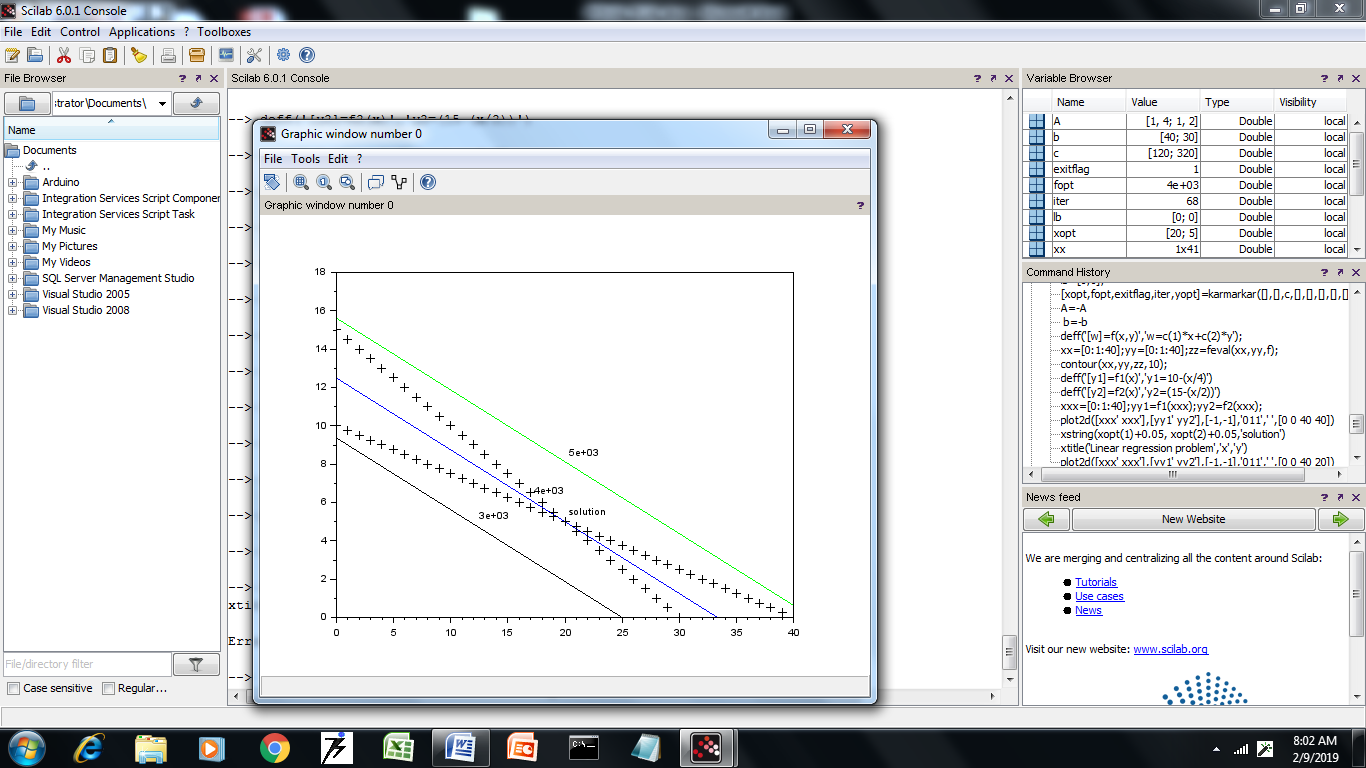
deff('[y2]=f2(x)','y2=(15-(x/2))')

xxx=[0:1:40];yy1=f1(xxx);yy2=f2(xxx);

plot2d([xxx' xxx'],[yy1' yy2'],[-1,-1],'011',' ',[0 0 40 18])

xstring(xopt(1)+0.05, xopt(2)+0.05,'solution')

xtitle('Primal Dual problem1,'x','y')



12. Plot and Prove FONC, SONC and SOSC condition of a function in SCILAB.

14. Implement Newton Raphson Method in SCILAB.

function **f**=myFunction(**x**)

**f** = **x**^3 - 2\***x**^2 + 4;

endfunction

function **df**=myDerivative(**x**)

**df** = 3\***x**^2 - 4\***x**;

endfunction

x0 = 2.0;

tolerance = 1e-6;

maxIterations = 100;

for i = 1:maxIterations

f\_x0 = myFunction(x0);

df\_x0 = myDerivative(x0);

x1 = x0 - f\_x0 / df\_x0;

if abs(x1 - x0) < tolerance

break;

end

x0 = x1;

end

if i >= maxIterations

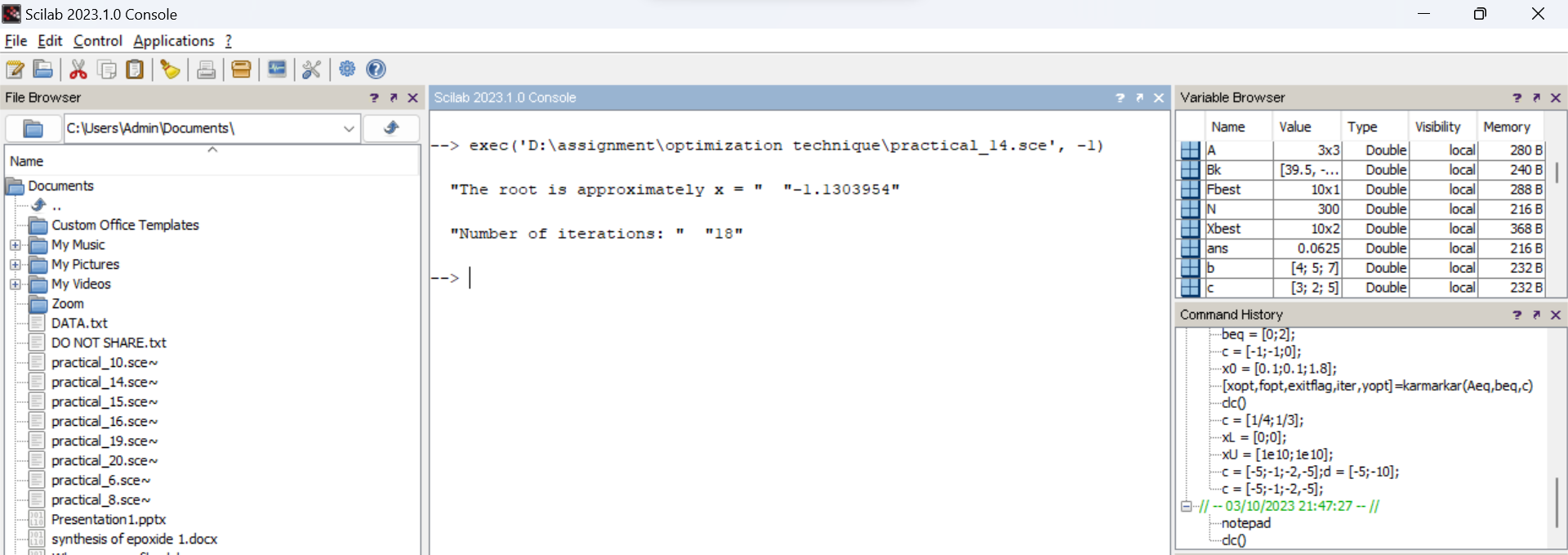
disp('Newton-Raphson method did not converge within the specified number of iterations.');

else

disp(['The root is approximately x = ', string(x0)]);

disp(['Number of iterations: ', string(i)]);

end



15. Implement Golden Section Method in SCILAB.

function **f**=objective(**x**)

**f** = (**x** - 2)^2 + 3;

endfunction

function [**xmin**, **fmin**]=goldenSectionMethod(**a**, **b**, **tolerance**)

phi = (1 + sqrt(5)) / 2; *// Golden ratio*

x1 = **b** - (**b** - **a**) / phi;

x2 = **a** + (**b** - **a**) / phi;

f1 = objective(x1);

f2 = objective(x2);

while abs(**b** - **a**) > **tolerance**

if f1 < f2

**b** = x2;

x2 = x1;

f2 = f1;

x1 = **b** - (**b** - **a**) / phi;

f1 = objective(x1);

else

**a** = x1;

x1 = x2;

f1 = f2;

x2 = **a** + (**b** - **a**) / phi;

f2 = objective(x2);

end

end

**xmin** = (**a** + **b**) / 2;

**fmin** = objective(**xmin**);

endfunction

a = 0;

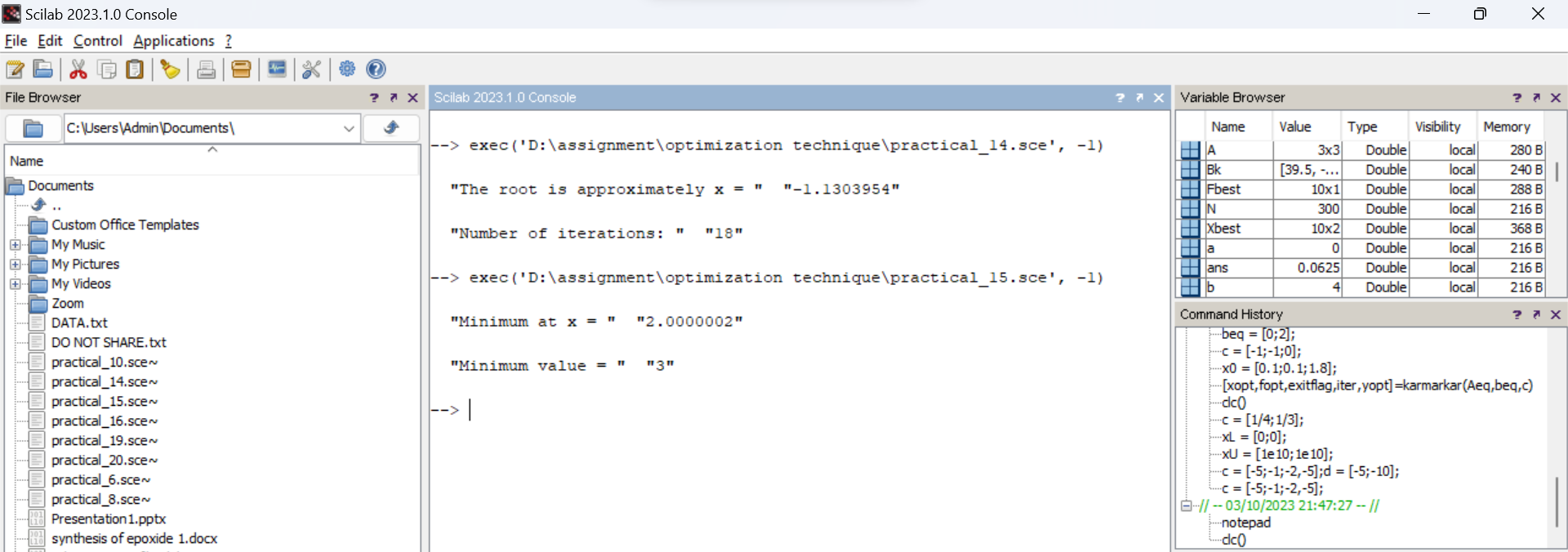
b = 4;

tolerance = 1e-6;

[xmin, fmin] = goldenSectionMethod(a, b, tolerance);

disp(['Minimum at x = ', string(xmin)]);

disp(['Minimum value = ', string(fmin)]);



16. Implement Fibonacci Series Method in SCILAB.

function **f**=objective(**x**)

**f** = (**x** - 2)^2 + 3;

endfunction

function [**xmin**, **fmin**]=fibonacciSearch(**a**, **b**, **n**)

fib = [1, 1];

for i = 3:**n**

fib(i) = fib(i-1) + fib(i-2);

end

x1 = **a** + (fib(**n**-2) / fib(**n**)) \* (**b** - **a**);

x2 = **a** + (fib(**n**-1) / fib(**n**)) \* (**b** - **a**);

f1 = objective(x1);

f2 = objective(x2);

for i = 1:**n**-2

if f1 < f2

**b** = x2;

x2 = x1;

f2 = f1;

x1 = **a** + (fib(**n**-i-2) / fib(**n**-i-1)) \* (**b** - **a**);

f1 = objective(x1);

else

**a** = x1;

x1 = x2;

f1 = f2;

x2 = **a** + (fib(**n**-i-1) / fib(**n**-i)) \* (**b** - **a**);

f2 = objective(x2);

end

end

**xmin** = (**a** + **b**) / 2;

**fmin** = objective(**xmin**);

endfunction

a = 0;

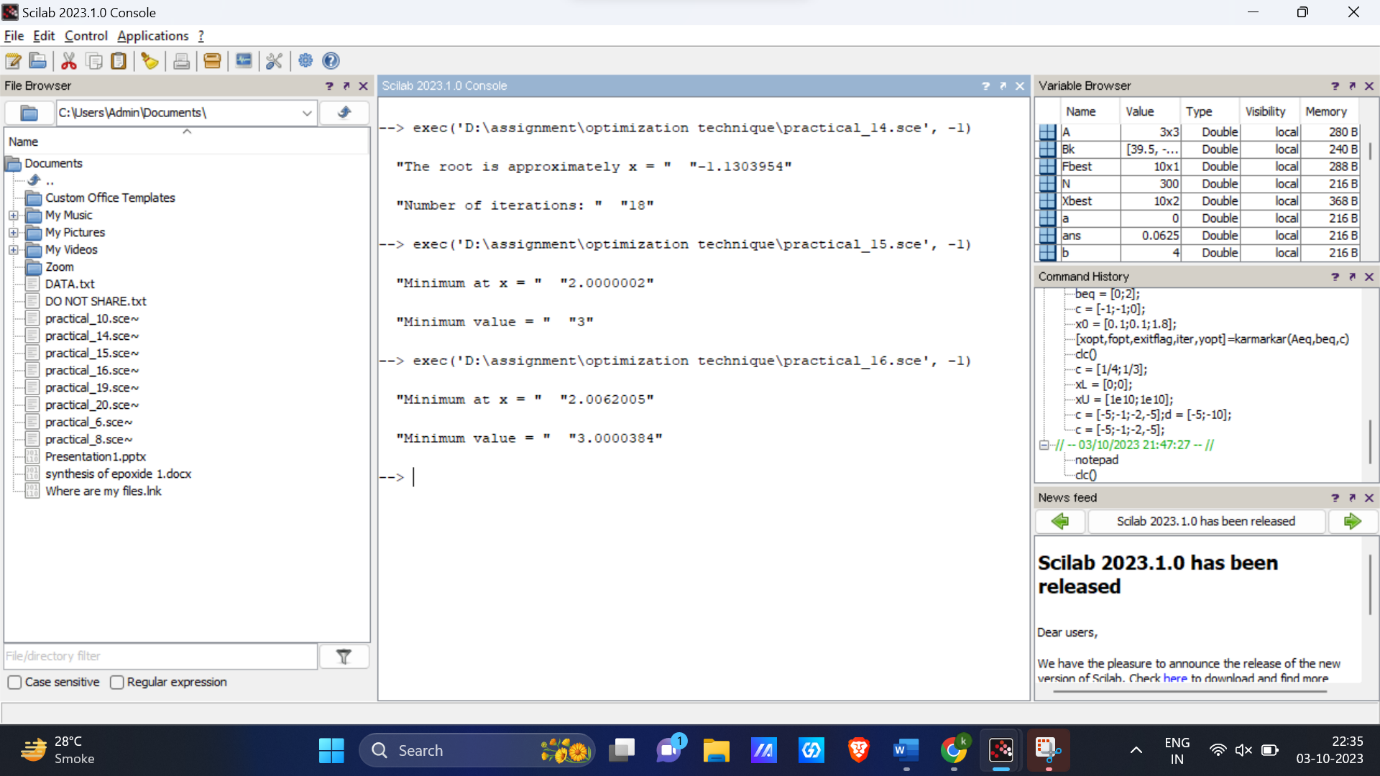
b = 4;

n = 10;

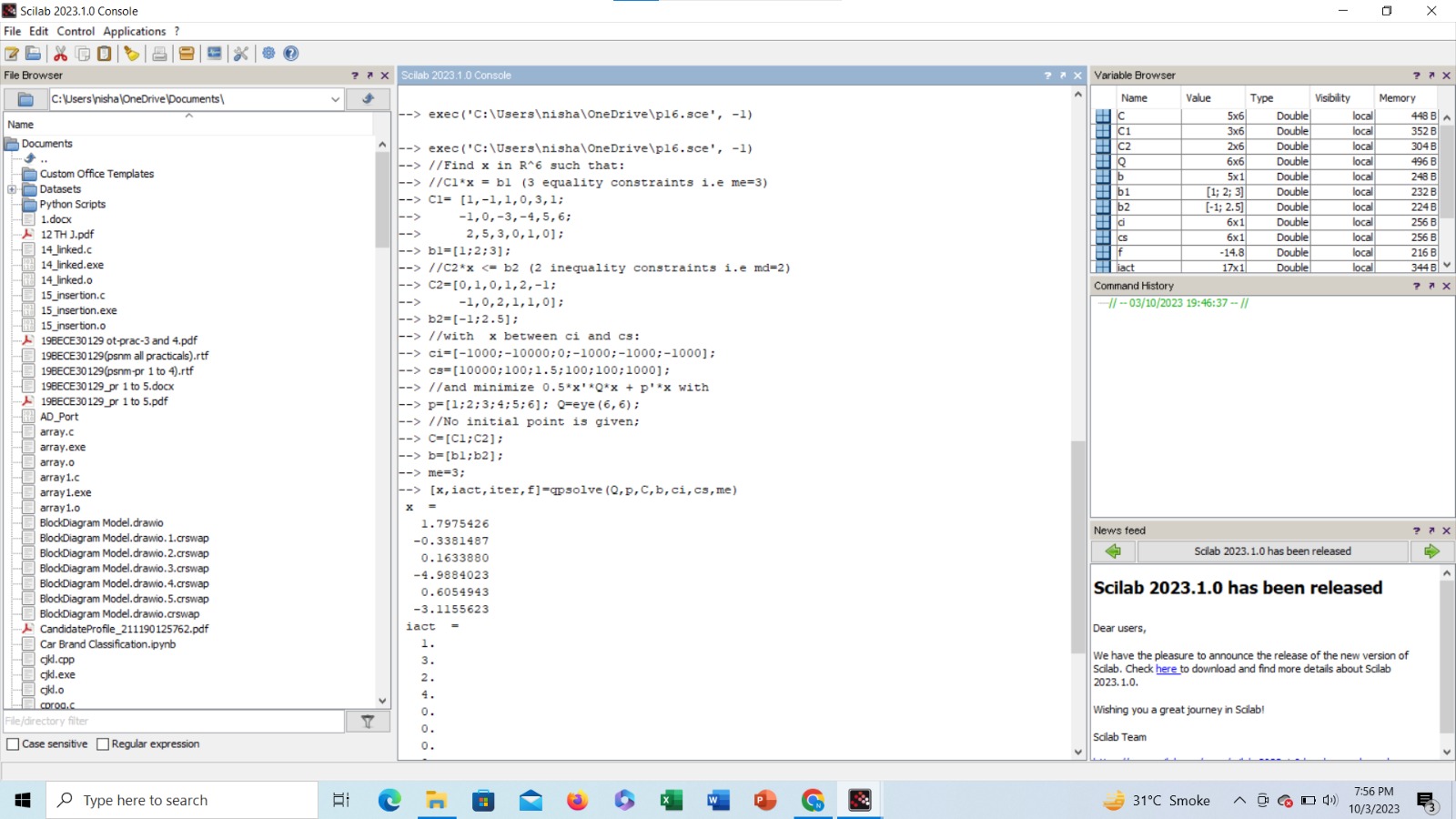
[xmin, fmin] = fibonacciSearch(a, b, n);

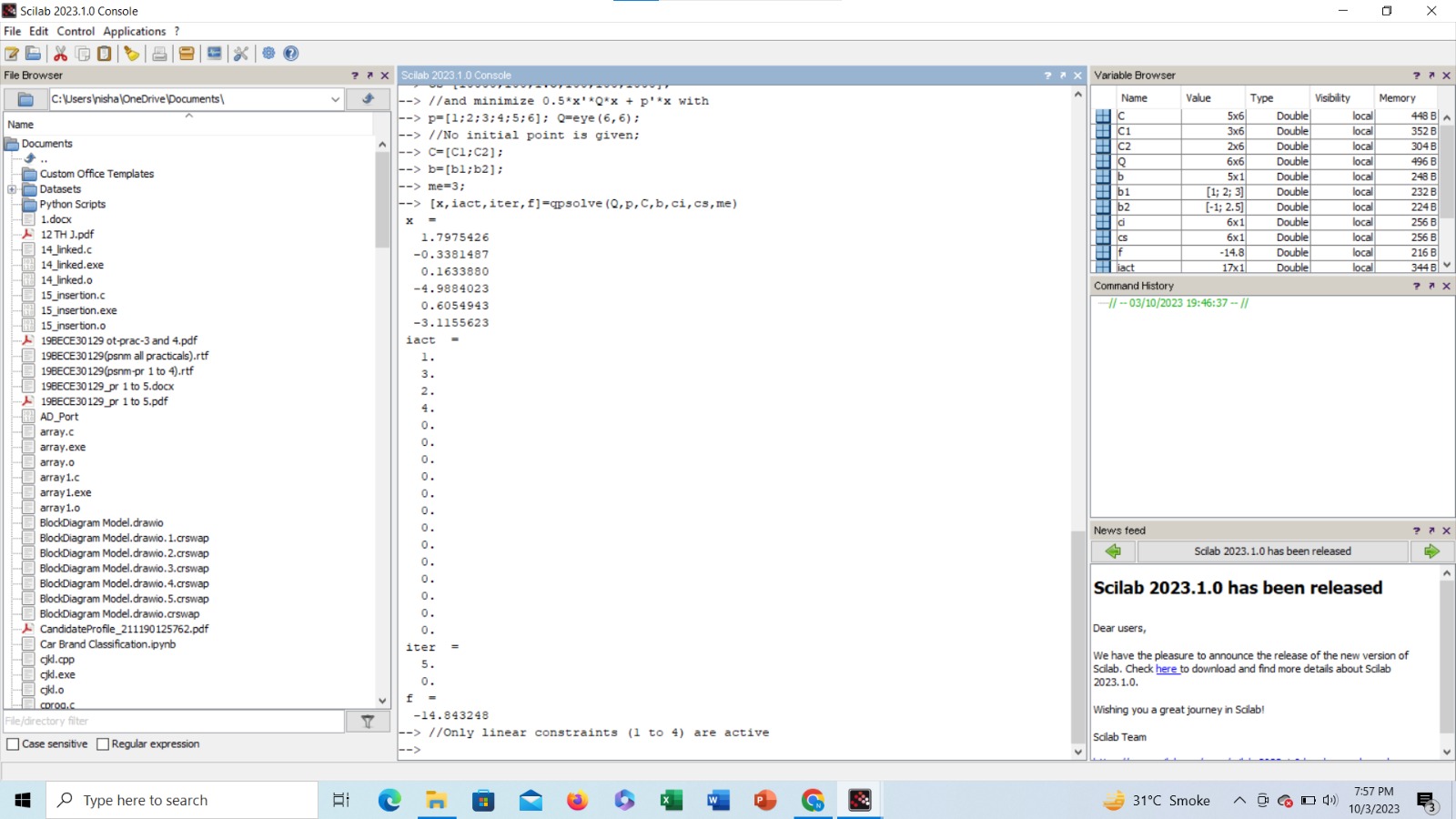
disp(['Minimum at x = ', string(xmin)]);

disp(['Minimum value = ', string(fmin)]);



17. Solve Quadratic Programming Problem using qp\_solve function in Scilab.





18. Solve Non Linear Programming Problem using optim in SCILAB.

19. Implement function for Steepest Descent Method in SCILAB.

function **y**=rastrigin(**x**)

n=max(size(**x**));

**y**=n+sum(**x**.^2-cos(2\*%pi\***x**));

endfunction

function **y**=rastrigingrad(**x**)

**y**=2\***x**+2\*%pi\*sin(2\*%pi\***x**);

endfunction

function **z**=dotprod(**x**, **y**)

**z**=sum(**x**.\***y**);

endfunction

function **d**=descentdirection(**f**, **x**, **fx**, **gx**)

**d**=-**gx**;

endfunction

function [**xnew**, **fnew**, **itback**]=backtracking(**f**, **x**, **fx**, **gx**, **d**)

tau=0.3;

bet=0.0001;

alphainit=1;

alpha=alphainit;**xnew**=**x**+alpha\***d**;

**fnew**=**f**(**xnew**);

**itback**=1;

while(**fnew**>**fx**+bet\*alpha\*dotprod(**gx**,**d**))

alpha=tau\*alpha;

**xnew**=**x**+alpha\***d**;

**fnew**=**f**(**xnew**);

**itback**=**itback**+1;

end

endfunction

disp('steepest descent method for the rastrigin function:');

timer();

n=evstr(x\_dialog('number of variables of the rastrigin function to minimize','2'));

epsilon=1E-5;

xmin=-5\*ones(1,n);

xmax=5\*ones(1,n);

u=rand(1,n);

x0=xmin+(xmax-xmin).\*u;

x=x0;fx=rastrigin(x);gx=rastrigingrad(x);

itgrad=1;

itfct=1;

Xbest=x;Fbest=fx;

while (norm(gx)>epsilon)

d=descentdirection(rastrigin,x,fx,gx);

[x,fx,itback]=backtracking(rastrigin,x,fx,gx,d);

Xbest=[Xbest;x];

Fbest=[Fbest;fx];

gx=rastrigingrad(x);

itgrad=itgrad+1;

itfct=itfct+itback;

end

disp('function evaluation number:');disp(itfct);

disp('gradient evaluation number:');disp(itgrad);

disp('minimum obtained:');disp(x);

disp('corresponding value by f:');disp(fx);

if (n==2)

xmin=-5.12;xmax=5.12;N=300;

xplot=xmin:((xmax-xmin)/(N-1)):xmax;

yplot=xplot;

zplot=zeros(N,N);

for i=1:N

for j=1:N

zplot(i,j)=rastrigin([xplot(i),yplot(j)]);

end

end

clf()

plot2d(Xbest(:,1),Xbest(:,2),rect=[-5.12,-5.12,5.12,5.12]);

contour2d(xplot,yplot,zplot,[0:0.01:0.1,0.2:1,1:10]);

xtitle('trajectory display');

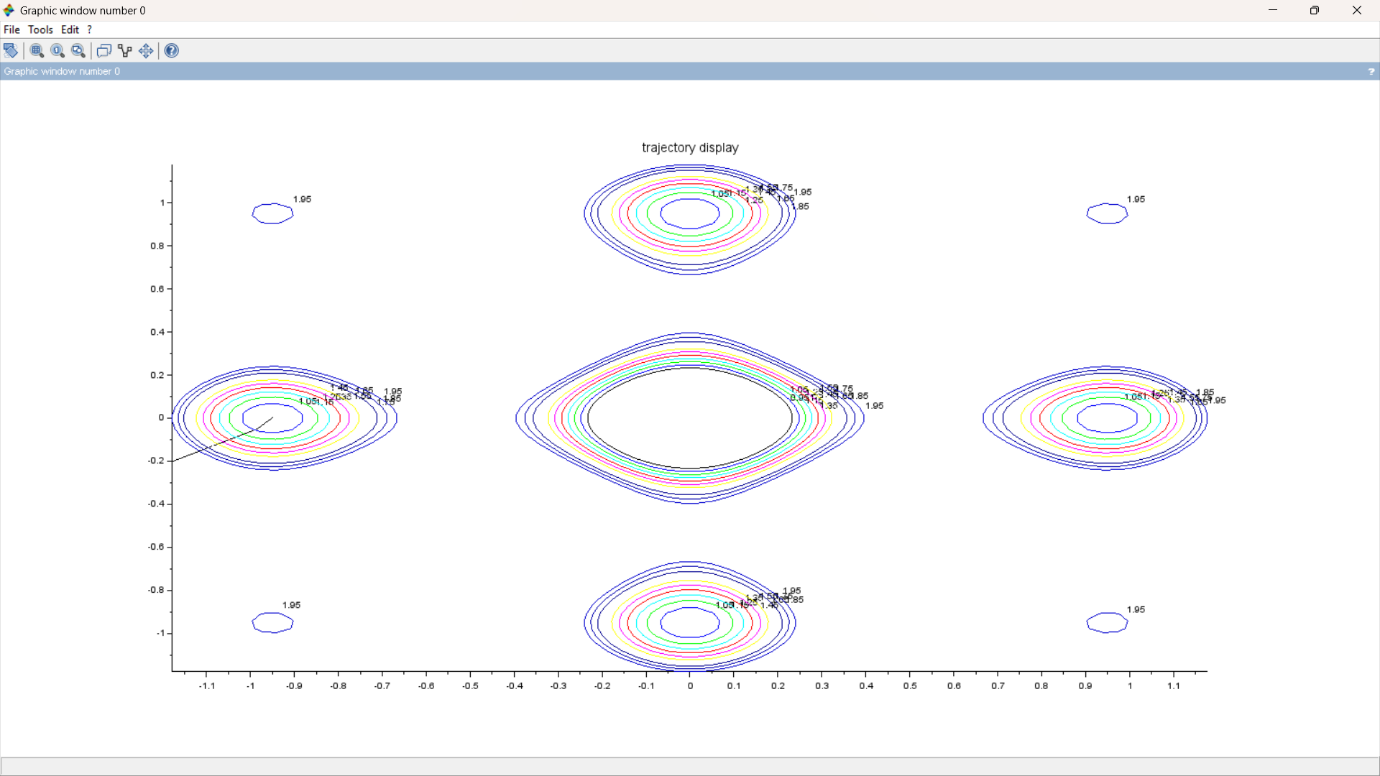
clf()

plot2d(Xbest(:,1),Xbest(:,2),rect=[x(1)-0.1,x(2)-0.1,x(1)+0.1,x(2)+0.1]);

contour2d(xplot,yplot,zplot,[fx:0.1:(fx+1)]);

xtitle('trajectory display');

end



20. Implement function for BFGS method in SCILAB.

disp('BFGS for Rastrigin');

function **y**=rastrigin(**x**)

n=max(size(**x**));

**y**=n+sum(**x**.^2-cos(2\*%pi\***x**));

endfunction

function **y**=rastrigingrad(**x**)

**y**=2\***x**+2\*%pi\*sin(2\*%pi\***x**);

endfunction

function **z**=dotprod(**x**, **y**);

**z**=sum(**x**.\***y**);

endfunction

function [**d**, **gx**]=descentdirection(**Bk**, **gx**);

**d**=linsolve(**Bk**,**gx**);

endfunction

function [**xnew**, **fnew**, **itback**]=backtracking(**f**, **x**, **fx**, **gx**, **d**);

tau=0.3;

bet=0.0001;

alphainit=1;

alpha=alphainit;**xnew**=**x**+alpha\***d**;

**fnew**=**f**(**xnew**);

**itback**=1;

while(**fnew**>**fx**+bet\*alpha\*dotprod(**gx**,**d**))

alpha=tau\*alpha;

**xnew**=**x**+alpha\***d**;

**fnew**=**f**(**xnew**);

**itback**=**itback**+1;

end

endfunction

timer();

n=2;

epsilon=1E-5;

xmin=-5\*ones(1,n);

xmax=5\*ones(1,n);

u=rand(1,n);

x0=xmin+(xmax-xmin).\*u;

x=x0;fx=rastrigin(x);gx=rastrigingrad(x);

itgrad=1;

itfct=1;

Xbest=x;Fbest=fx;

Bk=eye(n,n);

while (norm(gx)>epsilon)

x0=x;gx0=gx;

d=descentdirection(Bk,gx');

[x,fx,itback]=backtracking(rastrigin,x,fx,gx,d');

Xbest=[Xbest;x];

Fbest=[Fbest;fx];

gx=rastrigingrad(x);

itgrad=itgrad+1;

itfct=itfct+itback;

yk=gx'-gx0';sk=x'-x0';

Bk=Bk+(yk\*yk')/dotprod(sk,yk)-(Bk\*sk\*sk'\*Bk)/dotprod(sk,Bk\*sk);

end

disp('function evaluation number:');disp(itfct);

disp('gradient evaluation number:');disp(itgrad);

disp('minimum obtained:');disp(x);

disp('corresponding value by f:');disp(fx);

if (n==2)

xmin=-5.12;xmax=5.12;N=300;

xplot=xmin:((xmax-xmin)/(N-1)):xmax;

yplot=xplot;

zplot=zeros(N,N);

for i=1:N

for j=1:N

zplot(i,j)=rastrigin([xplot(i),yplot(j)]);

end

end

xset('window',0)

clf()

plot2d(Xbest(:,1),Xbest(:,2),rect=[-5.12,-5.12,5.12,5.12]);

contour2d(xplot,yplot,zplot,[0:0.01:0.1,0.2:1,1:10]);

xset('window',1)

clf()

plot2d(Xbest(:,1),Xbest(:,2),rect=[x(1)-0.1,x(2)-0.1,x(1)+0.1,x(2)+0.1]);

contour2d(xplot,yplot,zplot,[fx:0.1:(fx+1)]);

end

output:

"function evaluation number:"

19.

"gradient evaluation number:"

12.

"minimum obtained:"

2.8223695 -1.170D-08

"corresponding value by f:"

8.5265669