#Code 1 to enter 1*1 matrices and see their mathematical operations

```
clc % 'clc' stands for 'clean the screen (command window)'
clear % 'clear' command will clean all the previous variables from the 'workspace'
a = 5;
b = 7;
c = a+b;
display(c)
```

#Output

```
c =

    12
```

#Code 2 to enter matrices and do the basic operations

```
clc
clear
A = [6, 10];
B = A';
C = [7 ; 3];
D = [5,8,3; 2, 4, 9];
A1 = [1,2;3, 4];
B1 = [5,6; 7,8];
C1 = A1/B1;
```

#Output - Displaying all matrices in the Command Window

```
>> A

A =

     6    10

>> B

B =

     6
    10

>> C

C =

     7
     3

>> D

D =

     5     8     3
     2     4     9
```

```
>> A1

A1 =

     1     2
     3     4

>> B1

B1 =

     5     6
     7     8

>> C1

C1 =

    3.0000   -2.0000
    2.0000   -1.0000
```

#Code 3 to solve the system of linear equations

```
clc
clear
A = [6, 10];
B = A';
C = [7 ; 3];
D = [5,8,3; 2, 4, 9];
A1 = [1,2;3, 4];
B1 = [5,6; 7,8];
C1 = A1/B1;
X = inv(A1)*C;
Y = A1\C;
```

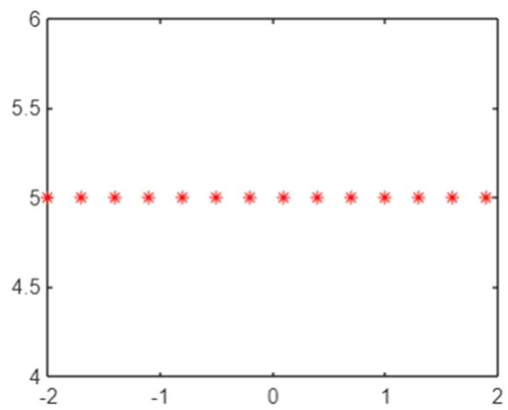
#Output - Displaying the matrices X and Y in the Command Window (matrices A1 & C are displayed in the output of code 2)

```
>> X

X =

  -11.0000
    9.0000

>> Y

Y =

  -11.0000
    9.0000
```

#Code 4 To draw graphs

```
clc
clear
x = -2:0.3:2;
y = 5*ones(size(x));
plot(x,y, 'r*')
```

#Output

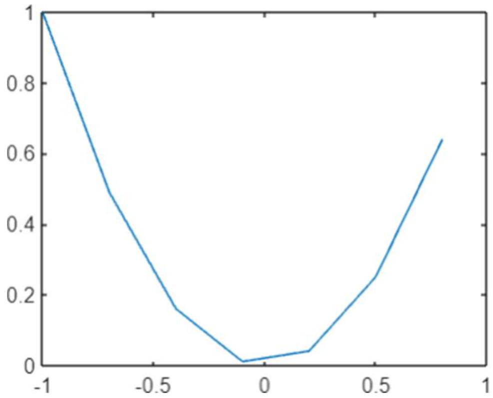


#Code 5 Drawing graphs using one function

#CASE 1(Drawing graph using normal line using plot function)

```
clc
clear
x = -1:0.3:1;
y = x.^2;
plot(x,y)
```

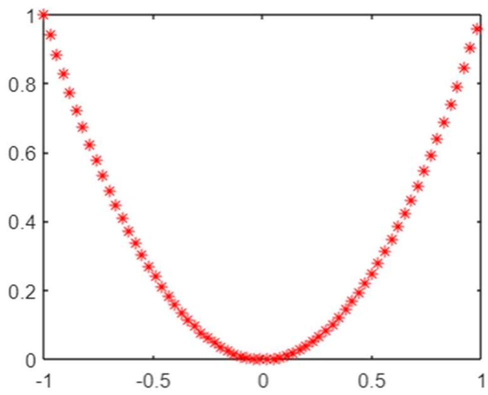
#Output



#CASE 2(Drawing graph using dotted line)

```
clc
clear
x = -1:0.03:1;
y = x.*x;
plot(x,y,'r*')
```

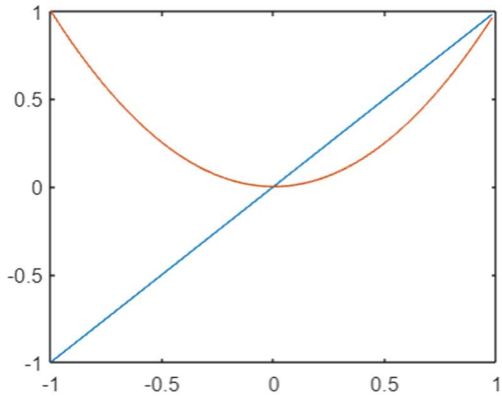
#Output



#Code 6 Drawing graphs using two functions

```
#CASE 1(Plotting the functions together using a single plot
function)
clc
clear
x = -1:0.03:1;
y1=x;y2=x.^2;
plot(x,y1,x,y2)
```

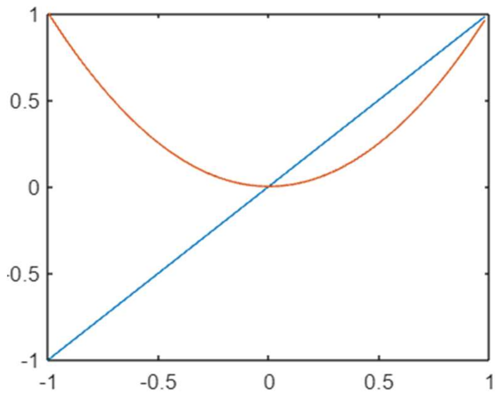
#Output



#CASE 2(using hold on function)

```
clc
clear
x = -1:0.03:1;
y1=x;y2=x.^2;
plot(x,y1)
hold on
plot(x,y2)
```

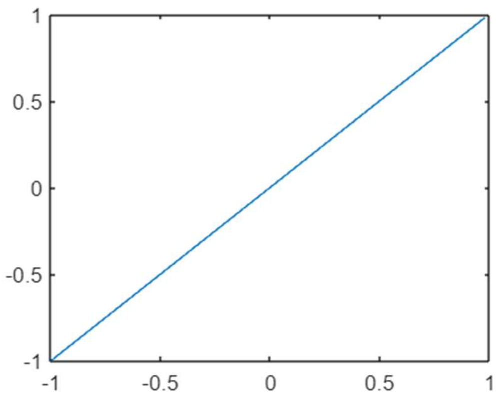
#Output



#CASE 3(Switching the positions of plot(x,y1) and plot(x,y2)and not using the hold on function)

```
clc
clear
x = -1:0.03:1;
y1=x;y2=x.^2;
plot(x,y2)
%hold on
plot(x,y1)
```

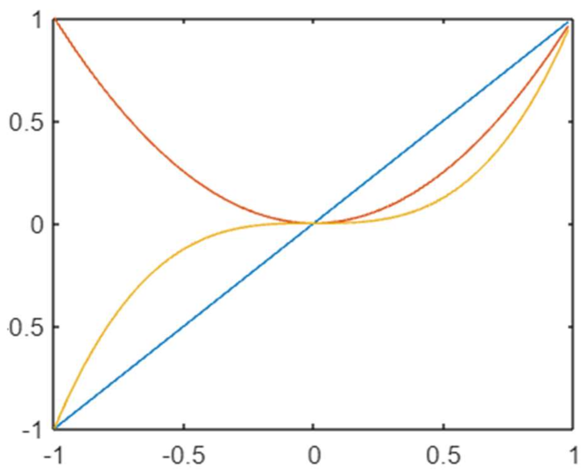
#Output (Note: It displays the graph of the last function i.e plot(x,y1))



#Code 7 Drawing graphs using three functions

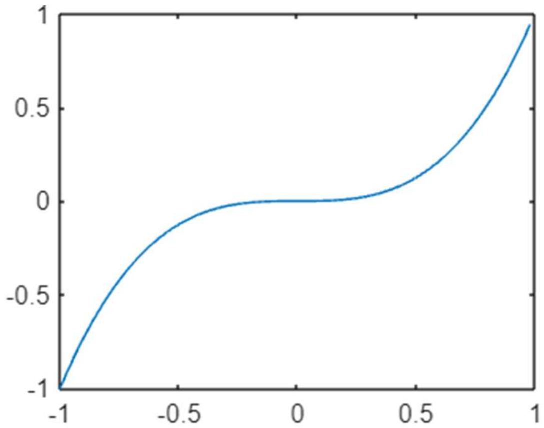
```
#CASE 1(Using hold on)
clc
clear
x = -1:0.03:1;
y1 = x;y2 = x.^2;y3 = x.^3;
plot(x,y1)
hold on
plot(x,y2)
hold on
plot(x,y3)
```

#Output



```
#CASE 2(Including hold off function)
clc
clear
x = -1:0.03:1;
y1 = x;y2 = x.^2;y3 = x.^3;
plot(x,y1)
hold on
plot(x,y2)
hold off
plot(x,y3)
```

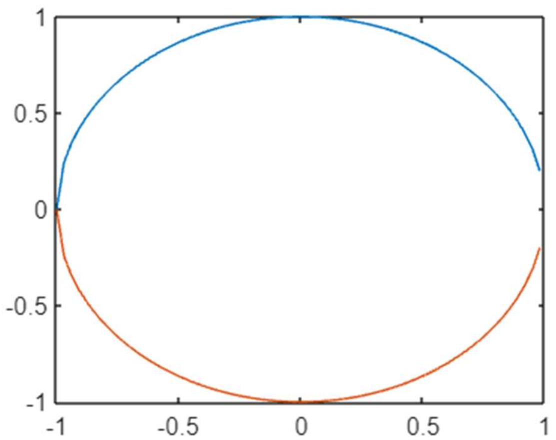
#Output (It will display the graph of the last function i.e plot(x,y3))



#Code 8 Drawing a circle using the equation x^2+y^2=1

```
clc
clear
x = -1:0.03:1;
y1 = sqrt(1-x.^2);
y2 = -sqrt(1-x.^2);
plot(x,y1)
hold on
plot(x,y2)
```

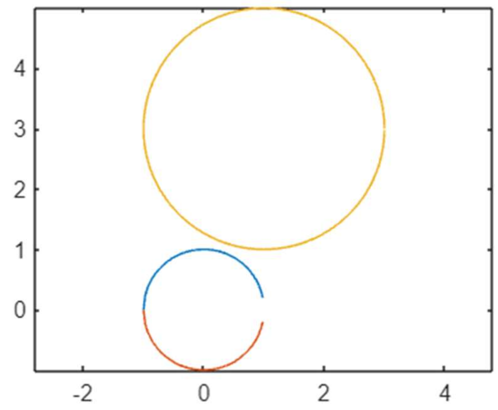
#Output



#Code 9 Drawing a circle using parametric equation

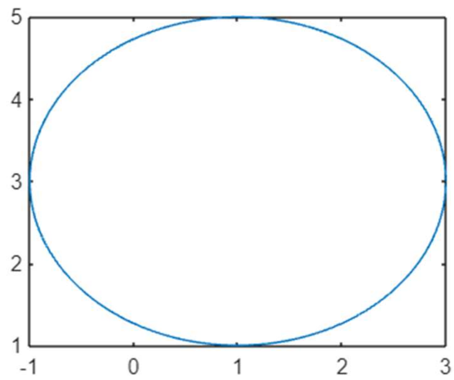
```
#CASE 1(Using axis equal function)
clc
clear
t = 0:0.03:2*pi;
x = 1 + 2*cos(t);
y = 3 + 2*sin(t);
plot(x,y)
axis equal
```

#Output



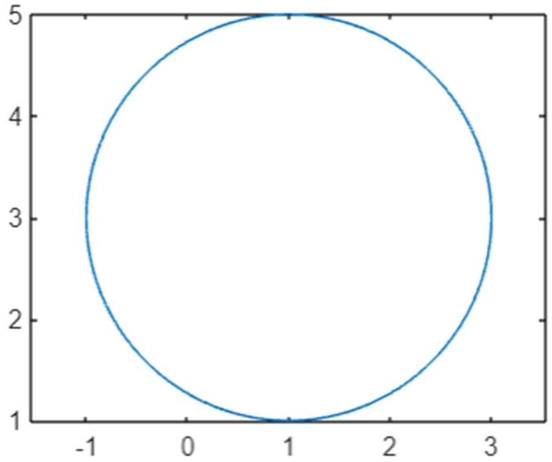
```
#CASE 2(Not using axis equal function)
clc
clear
t = 0:0.03:2*pi;
x = 1 + 2*cos(t);
y = 3 + 2*sin(t);
plot(x,y)
%axis equal
```

#Output



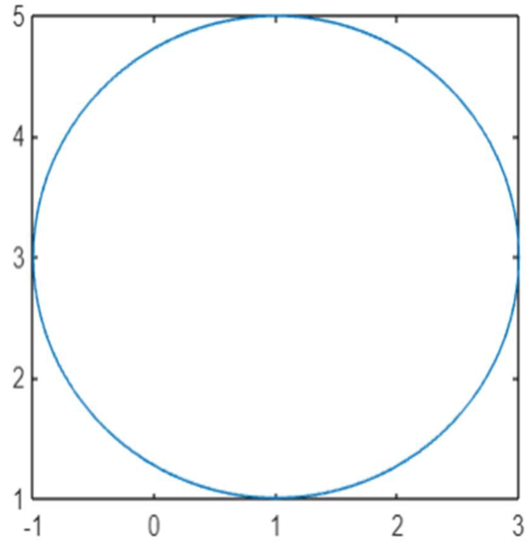
```
#CASE 3(Using linspace function and axis equal function)
clc
clear
t = linspace(0,2*pi,101);
x = 1 + 2*cos(t);
y = 3 + 2*sin(t);
plot(x,y)
axis equal
```

#Output



```
#CASE 4(Using linspace function and not using axis equal
function)
clc
clear
t = linspace(0,2*pi,101);
x = 1 + 2*cos(t);
y = 3 + 2*sin(t);
plot(x,y)
%axis equal
```

#Output



#Code 10 Examples using linspace function

```
#CASE 1
clc
clear
t = 0:10
t1 = linspace(0,10,11)
```

#Output - Displaying t and t1 in the Command Window

```
t =

     0     1     2     3     4     5     6     7     8     9    10


t1 =

     0     1     2     3     4     5     6     7     8     9    10
```

```
#CASE 2
clc
clear
t = 0:10
t1 = linspace(0,10)
```

#Output - Displaying t and t1 in the Command Window

```
t =

     0     1     2     3     4     5     6     7     8     9    10

t1 =

  Columns 1 through 13

         0    0.1010    0.2020    0.3030    0.4040    0.5051    0.6061    0.7071    0.8081    0.9091    1.0101    1.1111    1.2121

  Columns 14 through 26

    1.3131    1.4141    1.5152    1.6162    1.7172    1.8182    1.9192    2.0202    2.1212    2.2222    2.3232    2.4242    2.5253

  Columns 27 through 39

    2.6263    2.7273    2.8283    2.9293    3.0303    3.1313    3.2323    3.3333    3.4343    3.5354    3.6364    3.7374    3.8384

  Columns 40 through 52

    3.9394    4.0404    4.1414    4.2424    4.3434    4.4444    4.5455    4.6465    4.7475    4.8485    4.9495    5.0505    5.1515

  Columns 53 through 65

    5.2525    5.3535    5.4545    5.5556    5.6566    5.7576    5.8586    5.9596    6.0606    6.1616    6.2626    6.3636    6.4646

  Columns 66 through 78

    6.5657    6.6667    6.7677    6.8687    6.9697    7.0707    7.1717    7.2727    7.3737    7.4747    7.5758    7.6768    7.7778
  Columns 79 through 91

    7.8788    7.9798    8.0808    8.1818    8.2828    8.3838    8.4848    8.5859    8.6869    8.7879    8.8889    8.9899    9.0909

  Columns 92 through 100

    9.1919    9.2929    9.3939    9.4949    9.5960    9.6970    9.7980    9.8990   10.0000
```

```
#CASE 3
clc
clear
t = 0:10
t1 = linspace(0,10,10)
```

#Output
Displaying t and t1

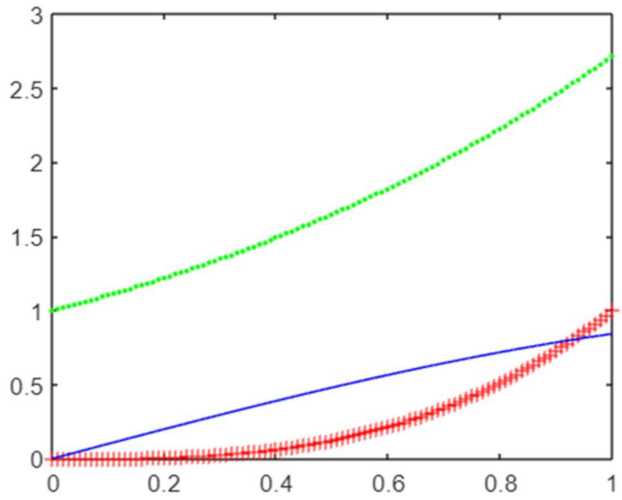
```
t =

  Column 1

     0

  Column 2

     1

  Column 3

     2

  Column 4

     3

  Column 5

     4

  Column 6

     5

  Column 7

     6

  Column 8

     7

  Column 9

     8

  Column 10

     9

  Column 11

    10
```

```
t1 =

  Column 1

         0

  Column 2

    1.1111

  Column 3

    2.2222

  Column 4

    3.3333

  Column 5

    4.4444

  Column 6

    5.5556

  Column 7

    6.6667

  Column 8

    7.7778

  Column 9

    8.8889

  Column 10

   10.0000
```

#Code 11 Curve Plot

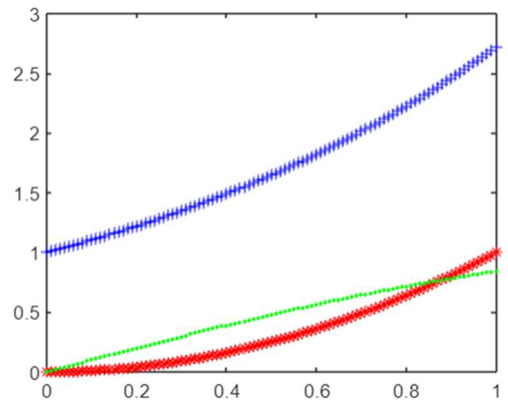
```
#CASE 1(plotting three functions without hold on)
clc
clear
x = linspace(0,1,101)
plot(x,x.^3,'r+',x,sin(x),'b-',x,exp(x),'g.')
```

#Output



```
#CASE 2(plotting three functions with hold on)
clear all
x = linspace(0,1,101)
plot(x,x.^2,'r*')
hold on
plot(x,sin(x),'g.')
hold on
plot(x,exp(x),'b+')
```

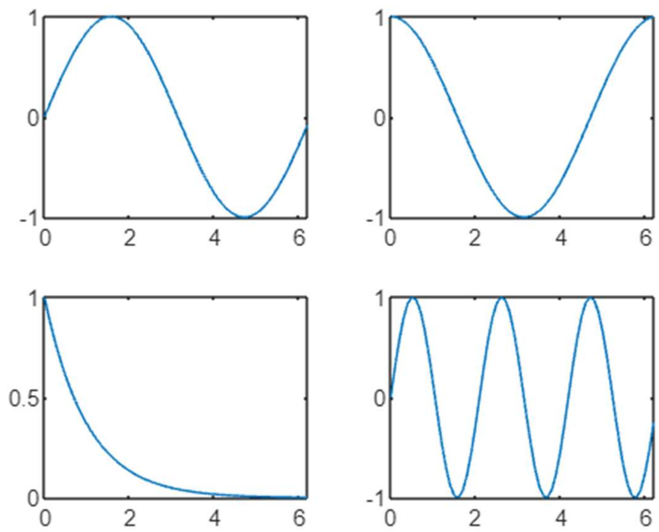
#Output



#Code 12 Drawing various subgraphs

```
x = 0:.1:2*pi;
subplot(2,2,1);
plot(x,sin(x));
subplot(2,2,2);
plot(x,cos(x));
subplot(2,2,3);
plot(x,exp(-x));
subplot(2,2,4);
plot(x,sin(3*x));
```

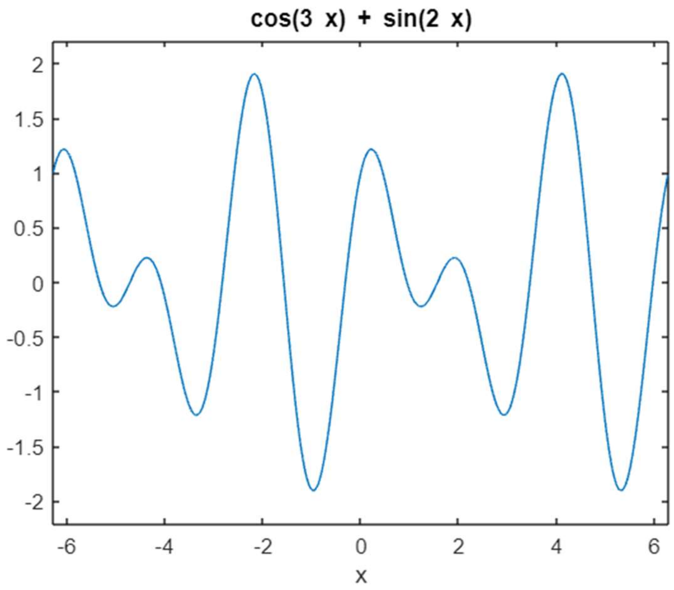
#Output



#Code 13 Ezplotting

```
syms x
f = sin(2*x)+cos(3*x)
ezplot(f)
```

#Output



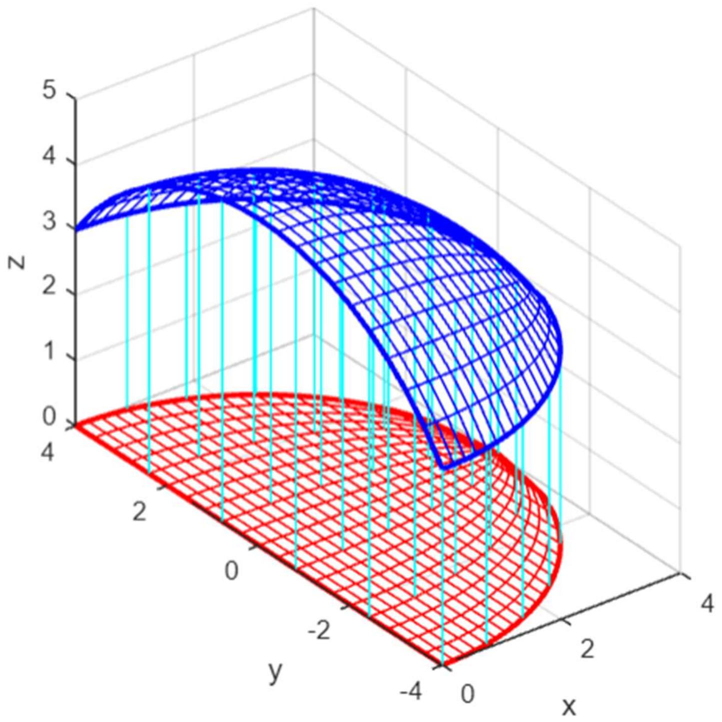
#Code 14 To create a solid of a particular function and displaying the solid

#CASE 1

```
function viewSolid(zvar, F, G, yvar, f, g, xvar, a, b)
%VIEWSOLID is a version for MATLAB of the routine on page 161
%  of "Multivariable Calculus and Mathematica" for viewing the
region
%  bounded by two surfaces for the purpose of setting up triple
integrals.
%  The arguments are entered from the inside out.
%  There are two forms of the command --- either f, g,
%  F, and G can be vectorized functions, or else they can
%  be symbolic expressions. xvar, yvar, and zvar can be
%  either symbolic variables or strings.
%  The variable xvar (x, for example) is on the
%  OUTSIDE of the triple integral, and goes between CONSTANT
limits a and b.
%  The variable yvar goes in the MIDDLE of the triple integral,
and goes
%  between limits which must be expressions in one variable
[xvar].
%  The variable zvar goes in the INSIDE of the triple integral,
and goes
%  between limits which must be expressions in two
%  variables [xvar and yvar].  The lower surface is plotted in
red, the
%  upper one in blue, and the "hatching" in cyan.
%
% Examples: viewSolid(z, 0, (x+y)/4, y, x/2, x, x, 1, 2)
% gives the picture on page 163 of "Multivariable Calculus and
Mathematica"
% and the picture on page 164 of "Multivariable Calculus and
Mathematica"
% can be produced by
%     viewSolid(z, x^2+3*y^2, 4-y^2, y, -sqrt(4-x^2)/2, sqrt(4-
x^2)/2, ...
%               x, -2, 2,)
% One can also type viewSolid('z', @(x,y) 0, ...
% @(x,y)(x+y)/4, 'y', @(x) x/2, @(x) x, 'x', 1, 2)
%

if isa(f, 'sym') % case of symbolic input
    ffun=inline(vectorize(f+0*xvar),char(xvar));
    gfun=inline(vectorize(g+0*xvar),char(xvar));
    Ffun=inline(vectorize(F+0*xvar),char(xvar),char(yvar));
    Gfun=inline(vectorize(G+0*xvar),char(xvar),char(yvar));
```

```
    oldviewSolid(char(xvar), double(a), double(b), ...
       char(yvar), ffun, gfun, char(zvar), Ffun, Gfun)
else
   oldviewSolid(char(xvar), double(a), double(b), ...
       char(yvar), f, g, char(zvar), F, G)
end
%%%%%%% subfunction goes here %%%%%%
function oldviewSolid(xvar, a, b, yvar, f, g, zvar, F, G)
for counter=0:20
  xx = a + (counter/20)*(b-a);
  YY = f(xx)*ones(1, 21)+((g(xx)-f(xx))/20)*(0:20);
  XX = xx*ones(1, 21);
%% The next lines inserted to make bounding curves thicker.
  widthpar=0.5;
  if counter==0, widthpar=2; end
  if counter==20, widthpar=2; end
%% Plot curves of constant x on surface patches.
 plot3(XX, YY, F(XX, YY).*ones(1,21), 'r', 'LineWidth',
widthpar);
 hold on
 plot3(XX, YY, G(XX, YY).*ones(1,21), 'b', 'LineWidth',
widthpar);
end;
%% Now do the same thing in the other direction.
XX = a*ones(1, 21)+((b-a)/20)*(0:20);
%% Normalize sizes of vectors.
YY=0:2; ZZ1=0:20; ZZ2=0:20;
for counter=0:20,
%% The next lines inserted to make bounding curves thicker.
  widthpar=0.5;
  if counter==0, widthpar=2; end
  if counter==20, widthpar=2; end
    for i=1:21,
       YY(i)=f(XX(i))+(counter/20)*(g(XX(i))-f(XX(i)));
       ZZ1(i)=F(XX(i),YY(i));
       ZZ2(i)=G(XX(i),YY(i));
    end;
  plot3(XX, YY, ZZ1, 'r', 'LineWidth',widthpar);
  plot3(XX, YY, ZZ2, 'b', 'LineWidth',widthpar);
end;
%% Now plot vertical lines.
for u = 0:0.2:1,
  for v = 0:0.2:1,
   x=a + (b-a)*u; y = f(a + (b-a)*u) +(g(a + (b-a)*u)-f(a + (b-
a)*u))*v;
   plot3([x, x], [y, y], [F(x,y), G(x, y)], 'c');
  end;
```

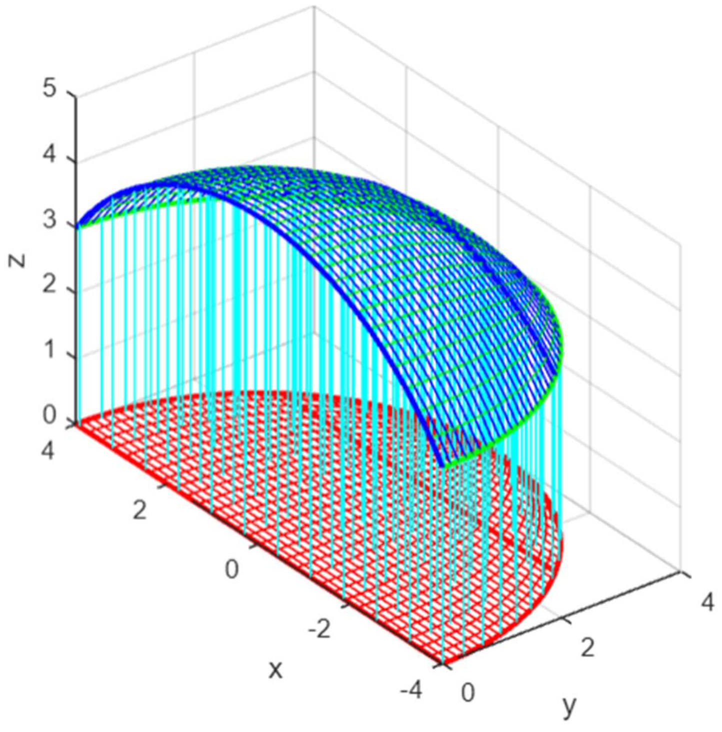
```
end;
xlabel(xvar)
ylabel(yvar)
zlabel(zvar)
hold off

xlabel(xvar)
ylabel(yvar)
zlabel(zvar)
hold off

# Code to display the solid
clc
clear
syms x y z
vol = int(int(sqrt(25-x^2-y^2),y,-sqrt(16-x^2),sqrt(16-
x^2)),x,0,4)
ViewSolid(z,0+0*x*y,sqrt(25-x^2-y^2),y,-sqrt(16-x^2),sqrt(16-
x^2),x,0,4);
axis equal;
grid on;

#Output
```



```
#CASE 2
function viewSolidOne(zvar, F, G, xvar, f, g, yvar, a, b)
%VIEWSOLID is a version for MATLAB of the routine on page 161
%  of "Multivariable Calculus and Mathematica" for viewing the
region
%  bounded by two surfaces for the purpose of setting up triple
integrals.
%  The arguments are entered from the inside out.
%  There are two forms of the command --- either f, g,
%  F, and G can be vectorized functions, or else they can
%  be symbolic expressions. xvar, yvar, and zvar can be
%  either symbolic variables or strings.
%  The variable xvar (x, for example) is on the
%  OUTSIDE of the triple integral, and goes between CONSTANT
limits a and b.
%  The variable yvar goes in the MIDDLE of the triple integral,
and goes
%  between limits which must be expressions in one variable
[xvar].
%  The variable zvar goes in the INSIDE of the triple integral,
and goes
%  between limits which must be expressions in two
%  variables [xvar and yvar].  The lower surface is plotted in
red, the
%  upper one in blue, and the "hatching" in cyan.
%
% Examples: viewSolid(z, 0, (x+y)/4, y, x/2, x, x, 1, 2)
% gives the picture on page 163 of "Multivariable Calculus and
Mathematica"
% and the picture on page 164 of "Multivariable Calculus and
Mathematica"
% can be produced by
%     viewSolid(z, x^2+3*y^2, 4-y^2, y, -sqrt(4-x^2)/2, sqrt(4-
x^2)/2, ...
%                 x, -2, 2,)
% One can also type viewSolid('z', @(x,y) 0, ...
% @(x,y)(x+y)/4, 'y', @(x) x/2, @(x) x, 'x', 1, 2)
%
if isa(f, 'sym') % case of symbolic input
    ffun=inline(vectorize(f+0*yvar),char(yvar));
    gfun=inline(vectorize(g+0*yvar),char(yvar));
    Ffun=inline(vectorize(F+0*xvar),char(xvar),char(yvar));
    Gfun=inline(vectorize(G+0*xvar),char(xvar),char(yvar));
    oldviewSolid(char(yvar),double(a), double(b), ...
       char(xvar), ffun, gfun, char(zvar), Ffun, Gfun)
```

```
else
   oldviewSolid(char(yvar),double(a),double(b),char(xvar), f, g,
char(zvar), F, G)
end
%%%%%%% subfunction goes here %%%%%%
function oldviewSolid(yvar,a , b, xvar, f, g, zvar, F, G)
for counter=0:30
  yy= a + (counter/30)*(b-a);
  XX = f(yy)*ones(1, 31)+((g(yy)-f(yy))/30)*(0:30);
  YY = yy*ones(1, 31);
%% The next lines inserted to make bounding curves thicker.
  widthpar=0.5;
  if counter==0, widthpar=2; end
  if counter==20, widthpar=2; end
%% Plot curves of constant x on surface patches.
 plot3(YY,XX, F(XX, YY).*ones(1,31), 'r', 'LineWidth',
widthpar);
 hold on
 plot3(YY,XX, G(XX, YY).*ones(1,31), 'b', 'LineWidth',
widthpar);
end;
%% Now do the same thing in the other direction.
YY = a*ones(1, 31)+((b-a)/30)*(0:30);
%% Normalize sizes of vectors.
XX=0:2; ZZ1=0:30; ZZ2=0:30;
for counter=0:30,
%% The next lines inserted to make bounding curves thicker.
  widthpar=0.5;
  if counter==0, widthpar=2; end
  if counter==30, widthpar=2; end
    for i=1:31,
       XX(i)=f(YY(i))+(counter/30)*(g(YY(i))-f(YY(i)));
       ZZ1(i)=F(YY(i),XX(i));
       ZZ2(i)=G(YY(i),XX(i));
    end;
  plot3(YY,XX, ZZ1, 'r', 'LineWidth',widthpar);
  plot3(YY,XX, ZZ2, 'g', 'LineWidth',widthpar);
end;
%% Now plot vertical lines.
for u = 0:0.09:1,
  for v = 0:0.09:1,
   y=a + (b-a)*u; x = f(a + (b-a)*u) +(g(a + (b-a)*u)-f(a + (b-
a)*u))*v;
   plot3([y, y], [x, x], [F(x,y), G(x, y)], 'c');
  end;
end;
xlabel(xvar)
```

```
ylabel(yvar)
zlabel(zvar)
hold off

# Code to display the solid
clc
clear
syms x y z
vol_1 = int(int(sqrt(25-x^2-y^2),y,-sqrt(16-x^2),sqrt(16-
x^2)),x,0,4);
ViewSolidOne(z,0+0*x*y,sqrt(25-x^2-y^2),y,-sqrt(16-x^2),sqrt(16-
x^2),x,0,4);
axis equal;
grid on;

#Output
```

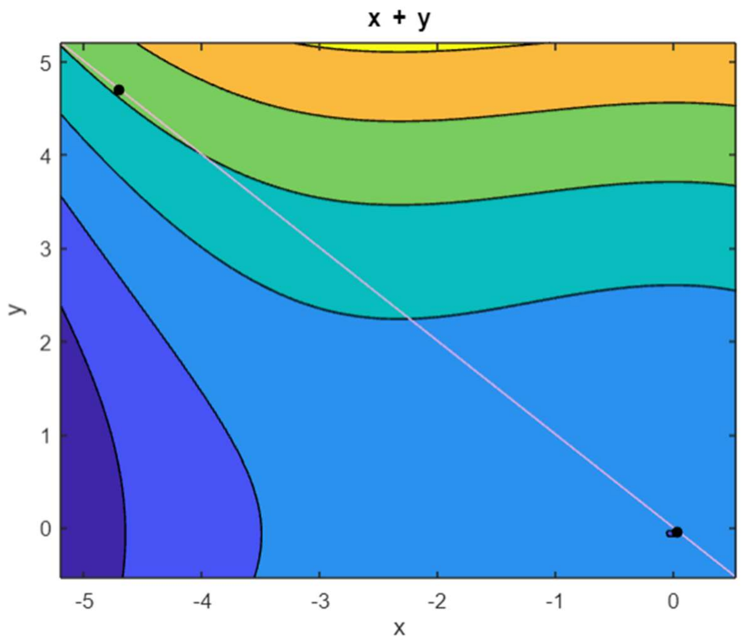


#Code 15 To find the extremities of a function of two variables

```
clc
clear
syms x y lam real
f = input('Enter f(x,y) to be extremized: ');
g = input('Enter the constraint function g(x,y): ');
F = f-lam*g;
Fd = jacobian(F,[x y lam]);
[ax,ay,alam] = solve(Fd,x,y,lam);
ax = double(ax); ay=double(ay);
T = subs(f,{x,y},{ax,ay}); T=double(T);
epxl = min(ax);
epxr = max(ax);
epyl = min(ay);
epyu = max(ay);
D = [epxl-0.5 epxr+0.5 epyl-0.5 epyu+0.5];
ezcontourf(f,D)
hold on
h = ezplot(g,D);
set(h,'Color',[1,0.7,0.9])
for i = 1:length(T)
fprintf('The critical point (x,y) is
(%1.3f,%1.3f).',ax(i),ay(i))
fprintf('The value of the function is %1.3f\n',T(i));
plot(ax(i),ay(i),'k.','markersize',15)
end
TT=sort(T);
f_min=TT(1)
f_max=TT(end)
```

#Output

```
Enter f(x,y) to be extremized:
2 * (x^3) + 7 * (x^2) + 7 * (y^2) + y
Enter the constraint function g(x,y):
x+y
The critical point (x,y) is (0.035,-0.035).The value of the function is -0.018
The critical point (x,y) is (-4.702,4.702).The value of the function is 106.314

f_min =

   -0.0178


f_max =

  106.3141
```



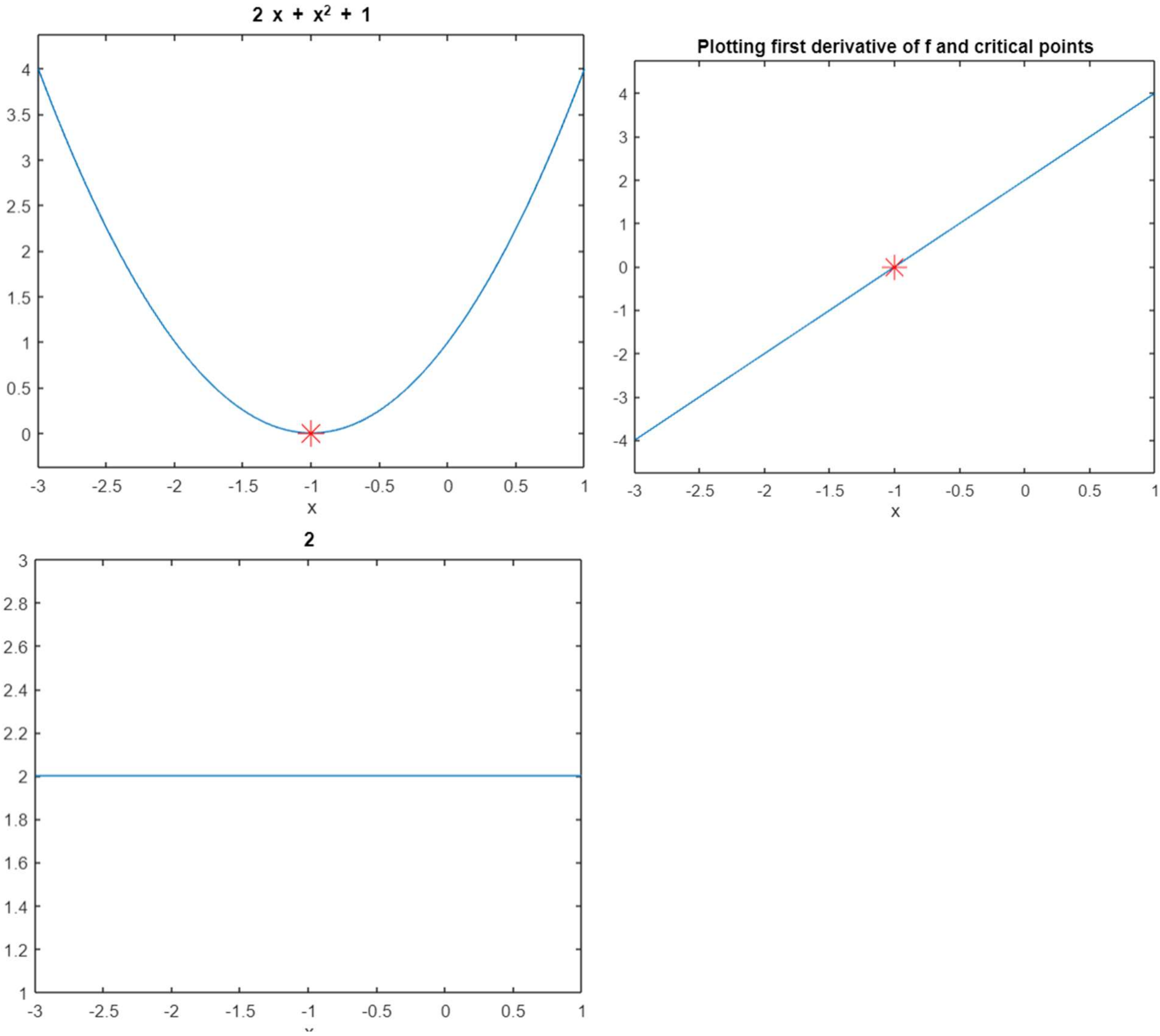
#Code 16 To find the local maxima and minima (using first and second derivative) and visualizing concavity for polynomial of degree 2 or more.

```
clc
clear
syms x real
f=input('Enter the function f(x):');
fx= diff(f,x);
c= solve(fx);
cmin= min(double(c));
cmax=  max(double(c));
figure(1)
ezplot(f,[cmin-2,cmax+2])
hold on
fxx= diff(fx,x)
for i=1:length(c)
    T1 = subs(fxx,x,c(i));
    T3 = subs(f,x,c(i));
if (double(T1)==0)
   sprintf('The test fails at x=%d',double(c(i)))
   else
      if (double(T1)<0)
   sprintf('The maximum point x is %d',double(c(i)))
   sprintf('The value of the function is %d',double(T3))
      else
   sprintf('The minimum point x is %d',double(c(i)))
         sprintf('The value of the function is %d',double(T3))
        end
end
plot(double(c(i)),double(T3),'r*','markersize',15);
end
%plotting inflection points for testing concavity
de=polynomialDegree(fxx);
if(de==0)
    sprintf('the given polynomial is second degree or less')
else
    d = solve(fxx) % finding inflection points
    for i = 1:1:size(d)
    T2=subs(f,x,d(i));
        R1=sign(subs(fxx,x,d(i)+0.0001));
        L1=sign(subs(fxx,x,d(i)-0.0001));
        check=abs(L1-R1)
     if (check==2)
         sprintf('The point x=%d is a point of
inflection',double(d(i)))
```

```
     else
         sprintf('The point x=%d is not a point of
inflection',double(d(i)))
     end
    plot(double(d(i)),double(T2),'g*','markersize',15);
    end
end
%Identifying maxima and minima through first derivative test
figure(2)
ezplot(fx,[cmin-2,cmax+2])
title('Plotting first derivative of f and critical points')
hold on
for i=1:1:size(c)
T4 = subs(fx,x,c(i));
plot(double(c(i)),double(T4),'r*','markersize',15);
end
figure(3)
ezplot(fxx,[cmin-2,cmax+2])
hold on
if(de==0)
    sprintf('the given polynomial is second degree or less,
second derivative plot is not possible')
else
for i = 1:1size(d)
T4 = subs(fxx,x,d(i));
plot(double(d(i)),double(T4),'r*','markersize',15);
end
title('Plotting second derivative of f and inflection points')
end
```

#Output

```
Enter the function f(x):
(x^2)+2*x+1

fxx =

2


ans =

    'The minimum point x is -1'


ans =

    'The value of the function is 0'


ans =

    'the given polynomial is second degree or less'


ans =

    'the given polynomial is second degree or less, second derivative plot is not possible'
```


2 x + x2 + 1
Plotting first derivative of f and critical points
2
x
x

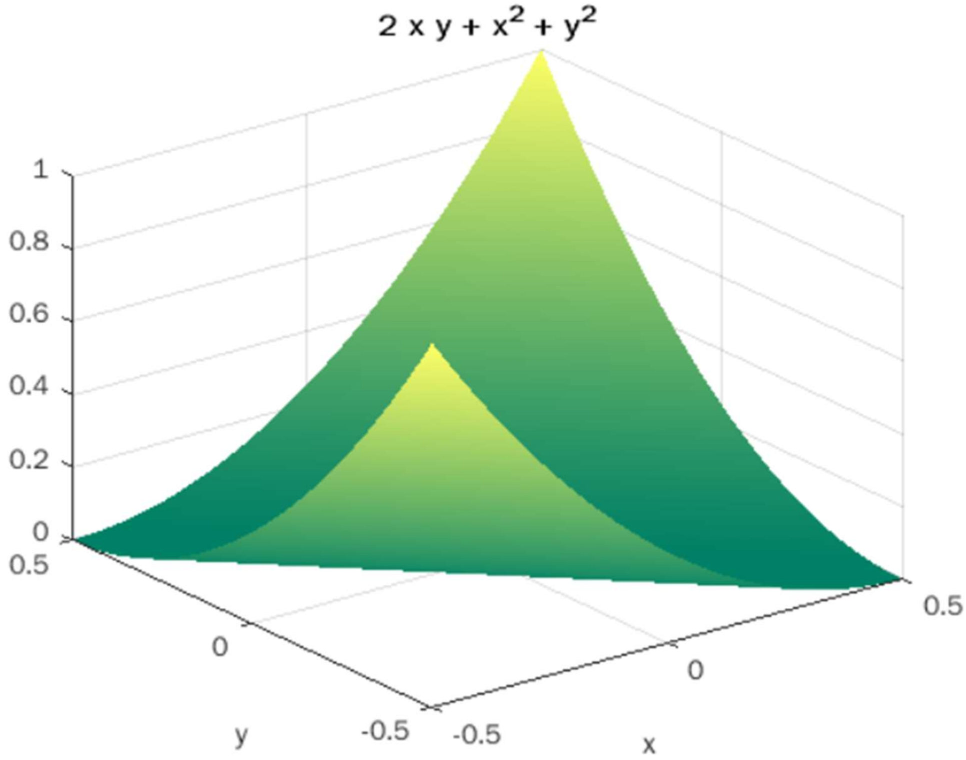
#Code 17 To find the Local Maxima and Minima for 2 variables

```
% Local maxima and minima for two variables
clc
clear
syms x y real
f = input('Enter the function f(x,y):');
fx = diff(f,x);
fy = diff(f,y);
[ax,ay] = solve(fx,fy);
fxx = diff(fx,x);
fyy = diff(fy,y);
fxy = diff(fx,y);
D = fxx*fyy-fxy^2;
r=1;
for k=1:size(ax)
if((imag(ax(k))==0))&&((imag(ay(k))==0))
ptx(r)=ax(k);
pty(r)=ay(k);
r=r+1;
end
end
a1=max(double(ax))
a2=min(double(ax))
b1=max(double(ay))
b2=min(double(ay))
ezsurf(f,[a2-0.5,a1+0.5,b2-0.5,b1+0.5])
colormap('summer');
shading interp
hold on
for r1=1:(r-1)
T1=subs(subs(D,x,ptx(r1)),y,pty(r1));
T2=subs(subs(fxx,x,ptx(r1)),y,pty(r1));
if (double(T1)==0)
sprintf('The point f(x,y) is (%d,%d) and need further
investigation',double(ptx(r1)),double(pty(r1)))
elseif (double(T1)<0)
T3=subs(subs(f,x,ptx(r1)),y,pty(r1))
sprintf('The point (x,y) is (%d,%d) a saddle
point',double(ptx(r1)),double(pty(r1)))
plot3(double(ptx(r1)),double(pty(r1)),double(T3),'b.','markersiz
e',30);
else
if (double(T2)<0)
sprintf('The maximum point(x,y) is
(%d,%d)',double(ptx(r1)),double(pty(r1)))
T3=subs(subs(f,x,ptx(r1)),y,pty(r1))
```

```
sprintf('The value of the function is %d',double(T3))
plot3(double(ptx(r1)),double(pty(r1)),double(T3),'r+','markersiz
e',30);
else
sprintf('The minimum point(x,y) is
(%d,%d)',double(ptx(r1)),double(pty(r1)))
T3=subs(subs(f,x,ptx(r1)),y,pty(r1))
sprintf('The value of the function is %d',double(T3))
plot3(double(ptx(r1)),double(pty(r1)),double(T3),'m*','markersiz
e',30);
end
end
end
```

#Output

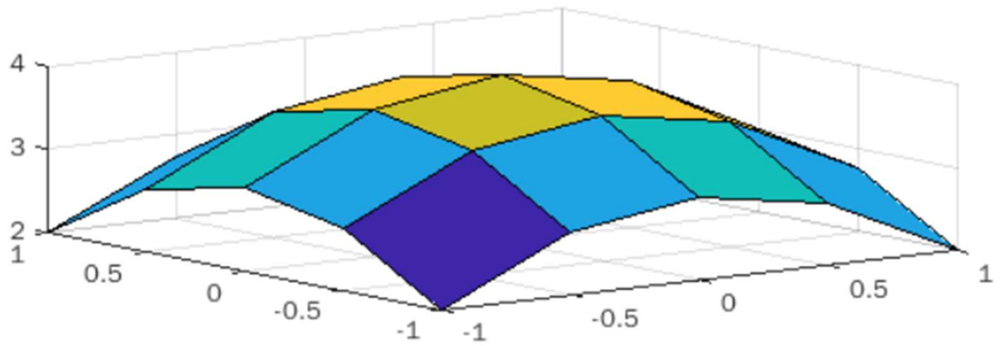
```
Enter the function f(x,y):
x^2+2*x*y+y^2

a1 =

     0


a2 =

     0


b1 =

     0


b2 =

     0


ans =

    'The point f(x,y) is (0,0) and need further investigation'
```

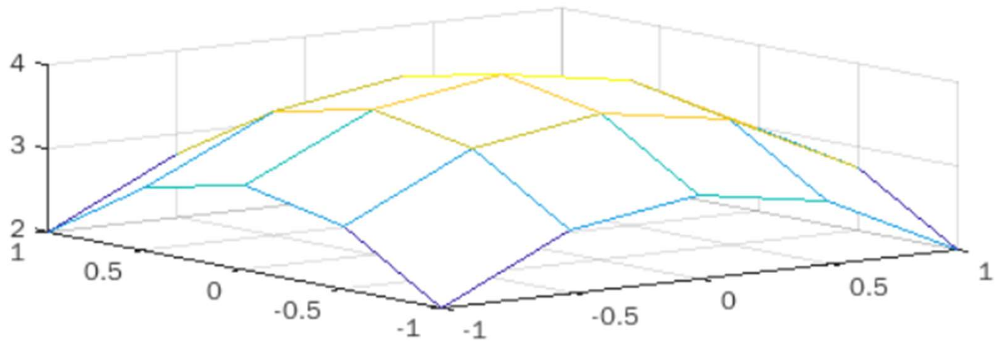

2 x y + x^2 + y^2
1
0.8
0.6
0.4
0.2
0
0.5
0
-0.5
y
-0.5
0
0.5
x

#Code 18 Subplot function

```
clc
clear
x=-1:.5:1;
y=x;
[x,y]=meshgrid(x,y);
z=4-x.^2-y.^2;
% z=0*x.^0.*y.^0;
% z=0
subplot(2,1,1)
surf(x,y,z)
subplot(2,1,2)
mesh(x,y,z)
```

#Output



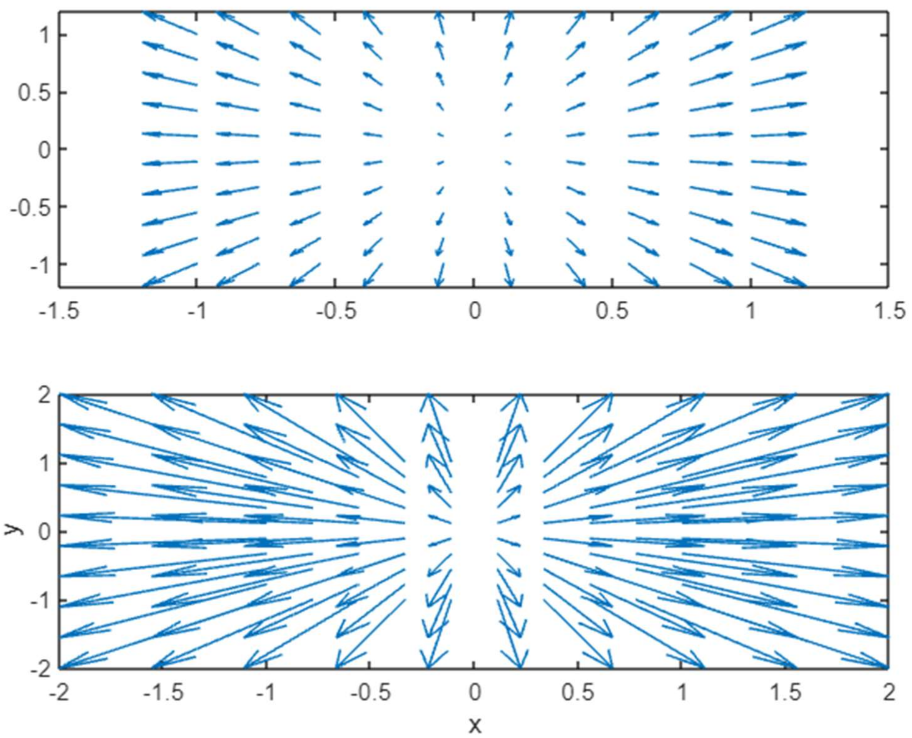


#Code 19 To draw the two dimensional vector field for the vector $x\vec{i} + y\vec{j}$

```
% To draw the two dimensional vector field for the vector xi →
+yj →
clc
clear
syms x y
F=input( 'enter the vector as i and j order in vector form:');
P = inline(vectorize(F(1)), 'x', 'y');
Q = inline(vectorize(F(2)), 'x', 'y');
x = linspace(-1, 1, 10);
y = x;
[X,Y] = meshgrid(x,y);
U = P(X,Y);
V = Q(X,Y);
subplot(2,1,1)
quiver(X,Y,U,V,1)
subplot(2,1,2)
quiver(X,Y,U,V,5)
axis on
xlabel('x')
ylabel('y')
```

#Output

```
enter the vector as i and j order in vector form:
[x,y]
```

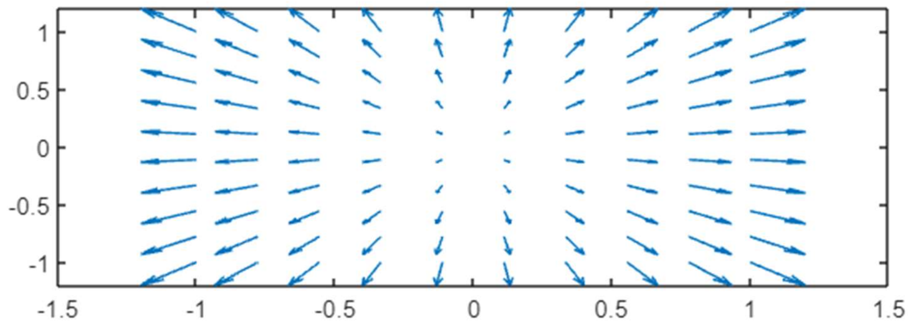


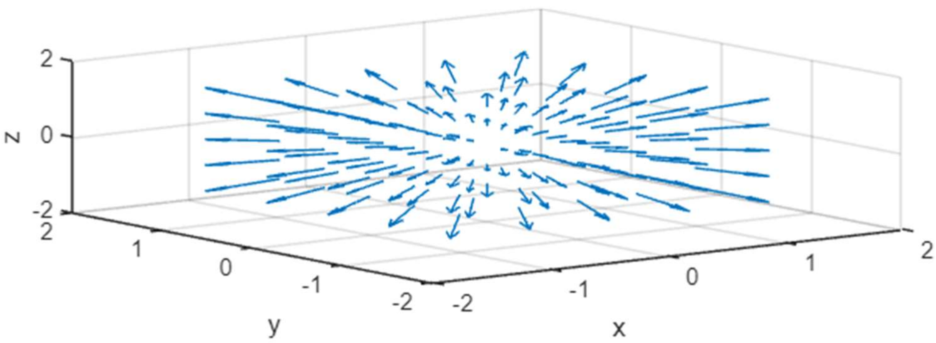
#Code 20 To draw the three dimensional vector field for the vector $x\vec{i} + y\vec{j} + z\vec{k}$

```
clc
clear
syms x y z
F=input( 'enter the vector as i,j and k order in vector form:')
P = inline(vectorize(F(1)), 'x', 'y','z');
Q = inline(vectorize(F(2)), 'x', 'y','z');
R = inline(vectorize(F(3)), 'x', 'y','z');
x = linspace(-1, 1, 5); y = x;
z=x;
[X,Y,Z] = meshgrid(x,y,z);
U = P(X,Y,Z);
V = Q(X,Y,Z);
W = R(X,Y,Z);
quiver3(X,Y,Z,U,V,W,1.5)
axis on
xlabel('x')
ylabel('y')
zlabel('z')
```

#Output

```
enter the vector as i,j and k order in vector form:
[x,y,z]


F =


[x, y, z]
```



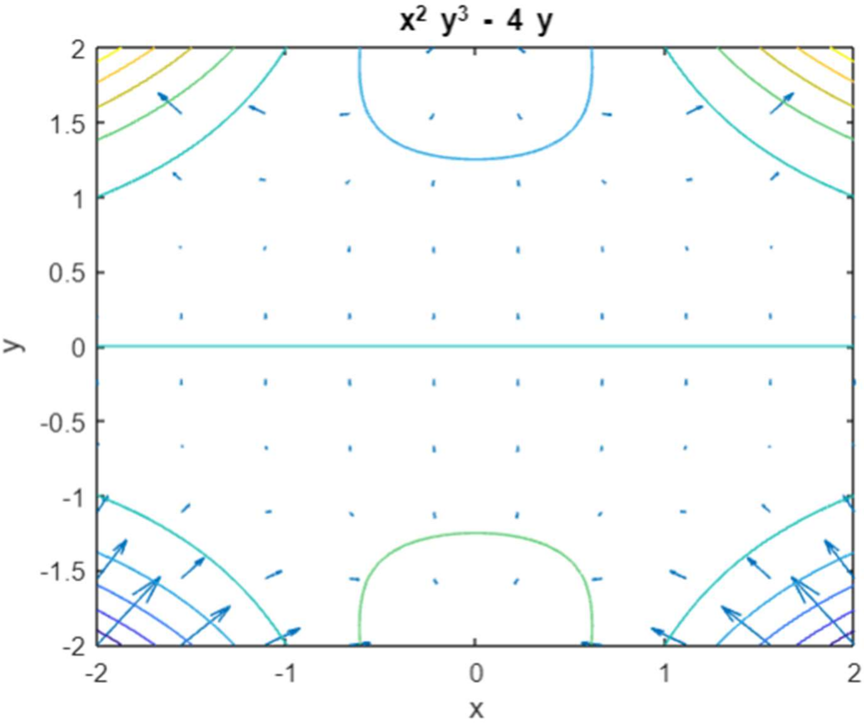


#Code 21 To find the gradient of a particular function

```
% To find the gradient of the function f=x^2*y^3-4*y
clc
clear
syms x y
f=input( 'enter the function f(x,y):');
fx=diff(f,x);
fy=diff(f,y);
P = inline(vectorize(fx), 'x', 'y');
Q = inline(vectorize(fy), 'x','y');
x = linspace(-2,2,10);
y = x;
[X,Y] = meshgrid(x,y);
U = P(X,Y);
V = Q(X,Y);
quiver(X,Y,U,V,1)
axis on
xlabel('x')
ylabel('y')
hold on
ezcontour(f,[-2, 2])%level curves
```

#Output

```
enter the function f(x,y):
x^2*y^3-4*y
```



#Code 22 Line dimension for work done in 2d

```
% Line integration to find the workdone 2-d
clc
clear
syms t x y
F=input('enter the F vector as i and j order in vector form:');
rbar = input('enter the r vector as i and j order in vector
form:');
lim=input('enter the limit of integration:');
vecfi=input('enter the vector field range');
drbar=diff(rbar,t);
sub = subs(F,[x,y],rbar);
F1=dot(sub,drbar)
int(F1,t,lim(1),lim(2))
P = inline(vectorize(F(1)), 'x', 'y');
Q = inline(vectorize(F(2)), 'x', 'y');
x = linspace(vecfi(1),vecfi(2), 10); y = x;
[X,Y] = meshgrid(x,y);
U = P(X,Y);
V = Q(X,Y);
quiver(X,Y,U,V)
hold on
fplot(rbar(1),rbar(2),[lim(1),lim(2)])% drawing the curve C
axis on
xlabel('x')
ylabel('y')
```

#Output

```
enter the F vector as i and j order in vector form:
[x^2,-x*y]
enter the r vector as i and j order in vector form:
[cos(t),sin(t)]
enter the limit of integration:
[0,pi/2]
enter the vector field range
[0,2]

F1 =

- sin(t)*cos(conj(t))^2 - sin(conj(t))*cos(t)*cos(conj(t))


ans =

-2/3
```