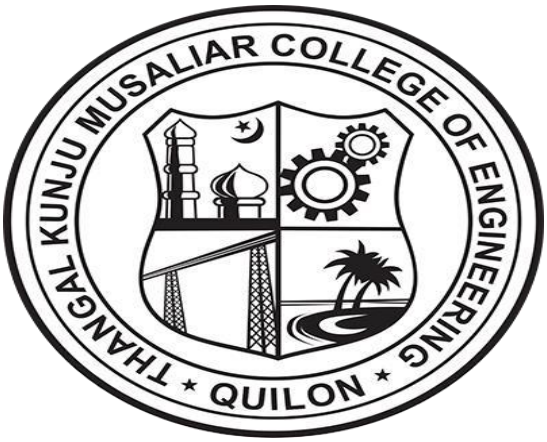
DEPARTMENT OF COMPUTER APPLICATION

TKM COLLEGE OF ENGINEERING

KOLLAM-691005



23MCAP103(B)- Mathematical Foundations for Computing

PRACTICAL RECORD BOOK

First Semester MCA

2025-2026

Submitted by

NAME:Neeraj N

REG NO: 25MCA042

DEPARTMENT OF COMPUTER APPLICATION

TKM COLLEGE OF ENGINEERING

KOLLAM-691005

Certificate

This is a bonafide record of the work done by Neeraj N in the First Semester for the lab course Mathematical Foundations for Computing(23MCAP103(B)) towards the partial fulfillment of the degree of Master of Computer Applications during the academic year 2025-2026

Staff Member in-charge *Examiner*

…… ……..

List and arrays

```
l = [1,2,3,4]
```

```
l
```

```
[1, 2, 3, 4]
```

```
type(l)
```

```
list
```

```
l.append("apple")
```

```
l
```

```
[1, 2, 3, 4, 'apple']
```

```
l[0]
```

```
1
```

```
l[-1]
```

```
'apple'
```

```
l[3]
```

```
4
```

```
l[4]
```

```
'apple'
```

```
list_1 = []
```

```
list_1.append(3)
```

```
list_1
```

```
[3]
```

```
list_1.insert(0,5)
list_1
```

```
[5, 3]
```

```
a = [2,3]
b = [5,6]
a.extend(b)
```

```
a
```

```
[2, 3, 5, 6]
```

```
colors = ['red','blue','green','blue']
colors.remove('blue')
colors
```

```
['red', 'green', 'blue']
```

```
colors.pop(1)
```

```
colors
```

```
['red', 'blue']
```

```
colors.clear()
```

```
colors
```

```
[]
```

```
l = [ 'apple', 'orange', 'banana', 'apple' ]
```

```
l.index('orange')
```

```
1
```

```
l_1 = [1,2,4,2,2,4,3,2,3]
l_1.count(2)
```

```
4
```

```
l_1.sort()
```

```
l_1
```

```
[1, 2, 2, 2, 2, 3, 3, 4, 4]
```

```
l_1.sort(reverse = True)
```

```
l_1
```

```
[4, 4, 3, 3, 2, 2, 2, 2, 1]
```

```
l_1.reverse()
l_1
```

```
[1, 2, 2, 2, 2, 3, 3, 4, 4]
```

```
l.reverse()
l
```

```
['apple', 'banana', 'orange', 'apple']
```

```
l = [ 1 , 2, 3, 4, 5]
```

```
l.append(6)
l
```

```
[1, 2, 3, 4, 5, 6]
```

```
l.insert(1,10)
l
```

```
[1, 10, 2, 3, 4, 5, 6]
```

```
l = [ 'red', 'violet', 'blue', 'black']
l.extend(colors)
l
```

```
['red', 'violet', 'blue', 'black']
```

```
a = [ 'color' ]
l.extend(a)
l
```

```
['red', 'violet', 'blue', 'black', 'color']
```

```
l.pop(3)
```

```
'black'
```

```
l.pop()
```

```
'color'
```

```
l.append('purple')
```

```
b = l.pop()
b
```

```
'purple'
```

```
print(b)
```

```
purple
```

```
print("purple")
```

```
purple
```

```
a = input("Enter a color: ")
```

```
Enter a color:  violet
```

```
print(a)
```

```
violet
```

```
import numpy as np
```

```
A = np.array([1,2,3,4,5])
print(A)
```

```
[1 2 3 4 5]
```

```
B = np.array([[2,3],[1,5]])
print(B)
```

```
[[2 3]
 [1 5]]
```

```
C = np.array([[0,7],[1,9]])
```

```
D = np.add(B,C)
```

```
print(D)
```

```
[[ 2 10]
 [ 2 14]]
```

```
E = np.subtract(B,C)
print(E)
```

```
[[ 2 -4]
 [ 0 -4]]
```

```
B + C
```

```
array([[ 2, 10],
       [ 2, 14]])
```

```
B * C
```

```
array([[ 0, 21],
       [ 1, 45]])
```

```
G = np.dot(B,C)
G
```

```
array([[ 3, 41],
       [ 5, 52]])
```

```
H = np.linalg.inv(B)
H
```

```
array([[ 0.71428571, -0.42857143],
```

```
2*B
```

```
array([[ 4,  6],
       [ 2, 10]])
```

```
B
```

```
array([[2, 3],
       [1, 5]])
```

```
np.transpose(B)
```

```
array([[2, 1],
       [3, 5]])
```

```
np.linalg.det(B)
```

```
np.float64(7.000000000000001)
```

```
np.linalg.matrix_rank(B)
```

```
np.int64(2)
```

```
A = np.array([[2, 4],[3, 5]])
B = np.array([[1 , 5],[5, 6]])
```

```
A
```

```
array([[2, 4],
       [3, 5]])
```

```
B
```

```
array([[1, 5],
       [5, 6]])
```

```
sum = A + B
```

```
sum
```

```
array([[ 3,  9],
       [ 8, 11]])
```

```
determinent_A= np.linalg.det(A)
determinent_A
```

```
np.float64(-2.000000000000001)
```

```
determinent_B = np.linalg.det(B)
determinent_B
```

```
np.float64(-18.999999999999996)
```

```
product = np.dot(A,B)
```

```
product
```

```
array([[22, 34],
       [28, 45]])
```

```
rank_A = np.linalg.matrix_rank(A)
rank_A
```

```
np.int64(2)
```

```
Start coding or ge_nerate with AI.
```

```
import sympy as sp
```

```
from sympy import *
```

```
matrix1 = sp.Matrix([[1,2],[1,0]])
```

```
sp.pprint(matrix1)
```

```
⎡1  2⎤
⎢    ⎥
⎣1  0⎦
```

```
matrix2 = sp.Matrix([[5,2],[1,2]])
```

```
sp.pprint(matrix2)
```

```
⎡5  2⎤
⎢    ⎥
⎣1  2⎦
```

```
sum = matrix1 + matrix2
print(sum)
```

```
Matrix([[6, 4], [2, 2]])
```

```
pprint(sum)
```

```
⎡6  4⎤
⎢    ⎥
⎣2  2⎦
```

```
display(sum)
```

```
Matrix_diff = matrix1 - matrix2
display(Matrix_diff)
```

```
Matrix_mul = matrix1 * matrix2
display(Matrix_mul)
```

```
scalar = 2
```

```
scaled_matrix = scalar*matrix1
display(scaled_matrix)
```

```
matrix1_T = matrix1.transpose()
print("The transpose of matrix1 is : ")
display(matrix1_T)
```

```
The transpose of matrix1 is :
```

```
Determinent = matrix1.det()
print(Determinent)
```

```
-2
```

```
matrix2_inv = matrix2.inv()
display(matrix2_inv)
```

```
matrix_1 = sp.Matrix([[1,2,3],[4,5,6],[7,8,9]])
display(matrix_1)
```

```
matrix_2 = sp.Matrix([[4,5,6],[7,8,9],[1,2,3]])
display(matrix_2)
```

```
sum_matrixes = matrix_1 + matrix_2
product_matrixes = matrix_1 * matrix_2
transpose_matrix_1 = matrix_1.transpose()
determinent_matrix_1 = matrix_1.det()
#inverse_matrix_1 = matrix_1.inv()
```

```
display(sum_matrixes)
display(product_matrixes)
display(transpose_matrix_1)
display(determinent_matrix_1)
```

```
determinent_matrix2 = matrix_2.det()
display(determinent_matrix2)
```

```
matrix3 = sp.Matrix([[1,0,2],[1,2,7],[2,3,1]])
determinent_matrix3 = matrix3.det()
display(determinent_matrix3)
```

```
inv_matrix3 = matrix3.inv()
display(inv_matrix3)
```

```
a = [[1,0,2],[1,2,7],[2,3,1]]
def inverse_matrix(a):
    matrix1 = sp.Matrix(a)
    determinent_m1 = matrix1.det()
    if determinent_m1:
        inverse = matrix1.inv()
        print("The inverse of the matrix is: ")
        display(inverse)
    else:
        print("No inverse for this matrix.")

inverse_matrix(a)
```

```
The inverse of the matrix is:
```

```
a = [[1,1,1],[1,1,1],[1,1,1]]
def symm_check(a):
    matrix1 = sp.Matrix(a)
    print("The original matrix is: ")
    display(matrix1)
    transpose_m = matrix1.transpose()
    print("The transpose of the matrix is: ")
    display(transpose_m)
    if(matrix1 == transpose_m):
        print("It is symmetric.")

    else:
        print("It is not symmetric.")

symm_check(a)
```

```
The original matrix is:
The transpose of the matrix is:
```

```
a = [[1,0,2],[1,2,7],[2,3,1]]
def inverse_check(a):
    matrix1 = sp.Matrix(a)
    determinent_m1 = matrix1.det()
    display(matrix1)
    print("\n")
    if determinent_m1:
        print("The Matrix is invertible.")
    else:
```

```
        print("The matrix is not invertible.")

inverse_check(a)
```

```
The Matrix is invertible.
```

```
def input_matrix():
    rows = int(input("Enter the number of rows: "))
    cols = int(input("Enter the number of columns: "))
    print("Enter the elements (one element at a time.):")
    if(rows != cols):
        print("Not a square matrix.")
    else:
        elements = []
        for i in range(rows):
            items = []
            for j in range(cols):
                value = input()
                items.append(value)
            elements.append(items)

        matrix1 = sp.Matrix(elements)
        display(matrix1)

input_matrix()
```

```
Enter the number of rows:  2
Enter the number of columns:  2
Enter the elements (space-separated):
 1
 2
 3
 4
```

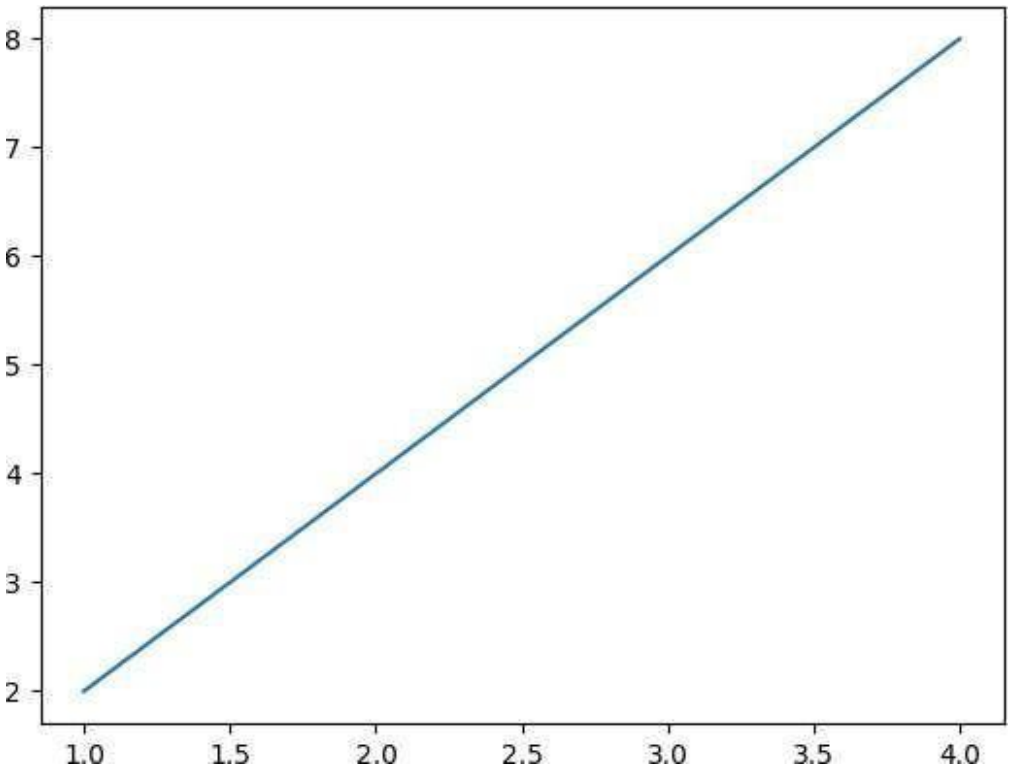
```
def input_matrix():
    rows = int(input("Enter the number of rows: "))
    cols = int(input("Enter the number of columns: "))
    print("Enter the elements (one element at a time.):")
    a = input("Enter the matrix: ").split()
    #simpify converts string into mathematic solution meaning it converts it to int.
    b = [x for x in a]
    c = sp.Matrix(rows,cols,b)
    display(c)

input_matrix()
```

```
Enter the number of rows:  2
Enter the number of columns:  2
Enter the elements (one element at a time.):
Enter the matrix:  1 2 3 4
```

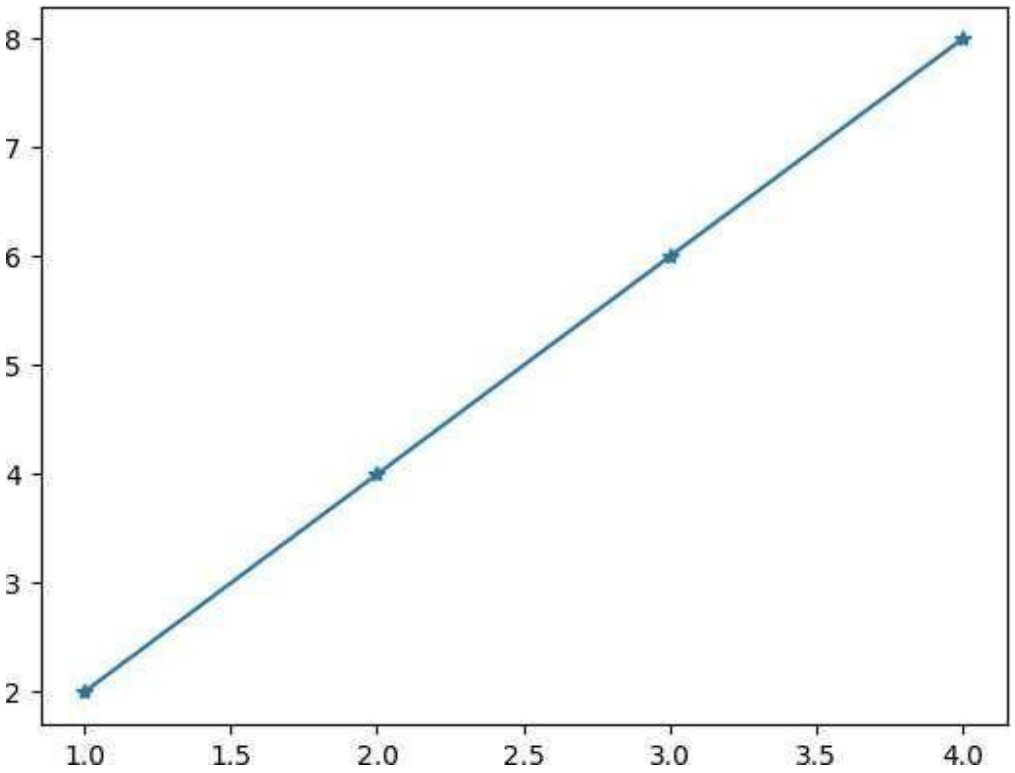
```
Start coding or ge_nerate with AI.
```

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
x=[1,2,3,4]
y=[2,4,6,8]
plt.plot(x,y)
plt.show()
```

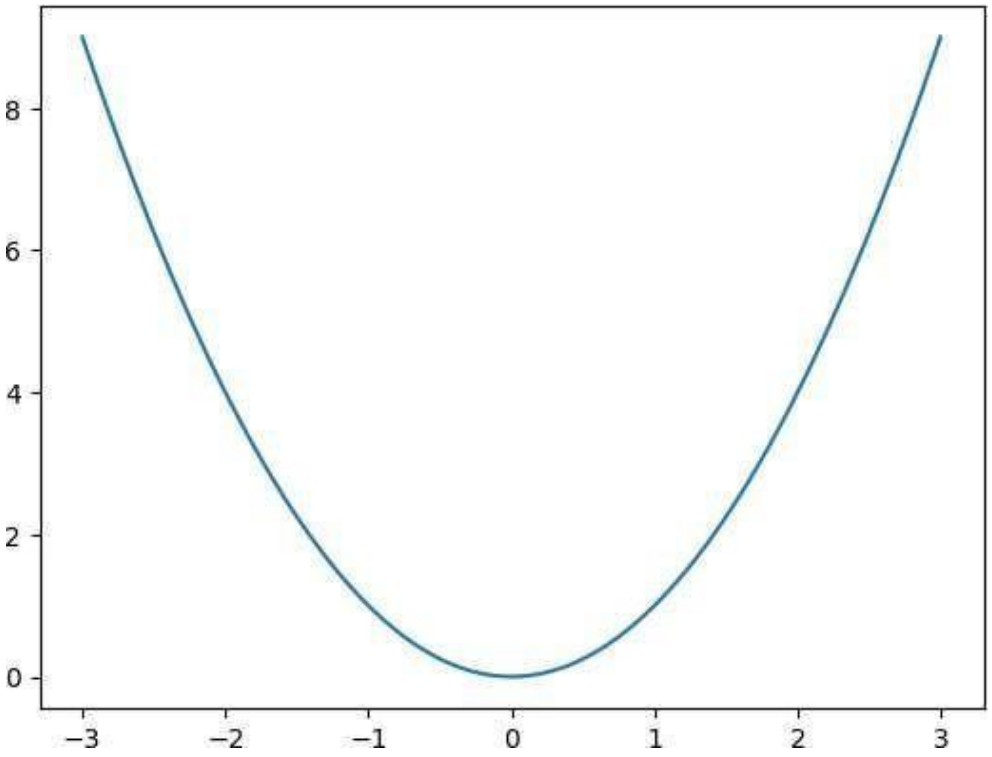


In [3]:
```
x=[1,2,3,4]
y=[2,4,6,8]
plt.plot(x,y,'*-')
plt.show()
```



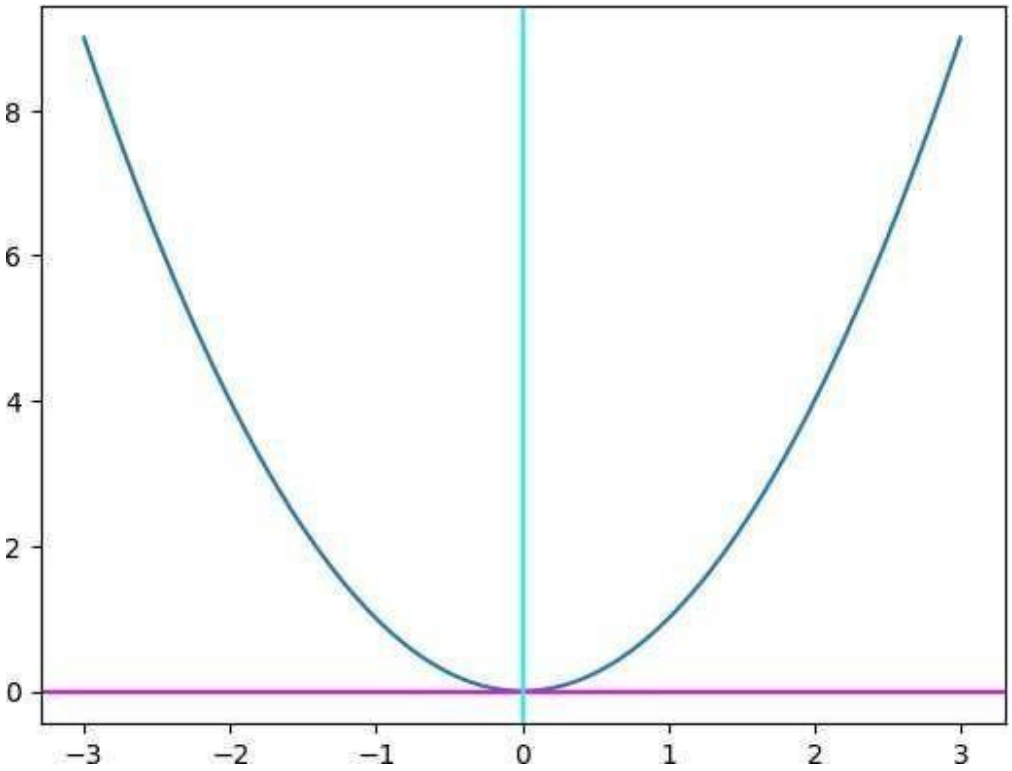
In [4]:
```
x=np.linspace(-3,3,100)
def f(x): return x**2
plt.plot(x,f(x))
```

Out[4]: [<matplotlib.lines.Line2D at 0x1bc793f0f50>]



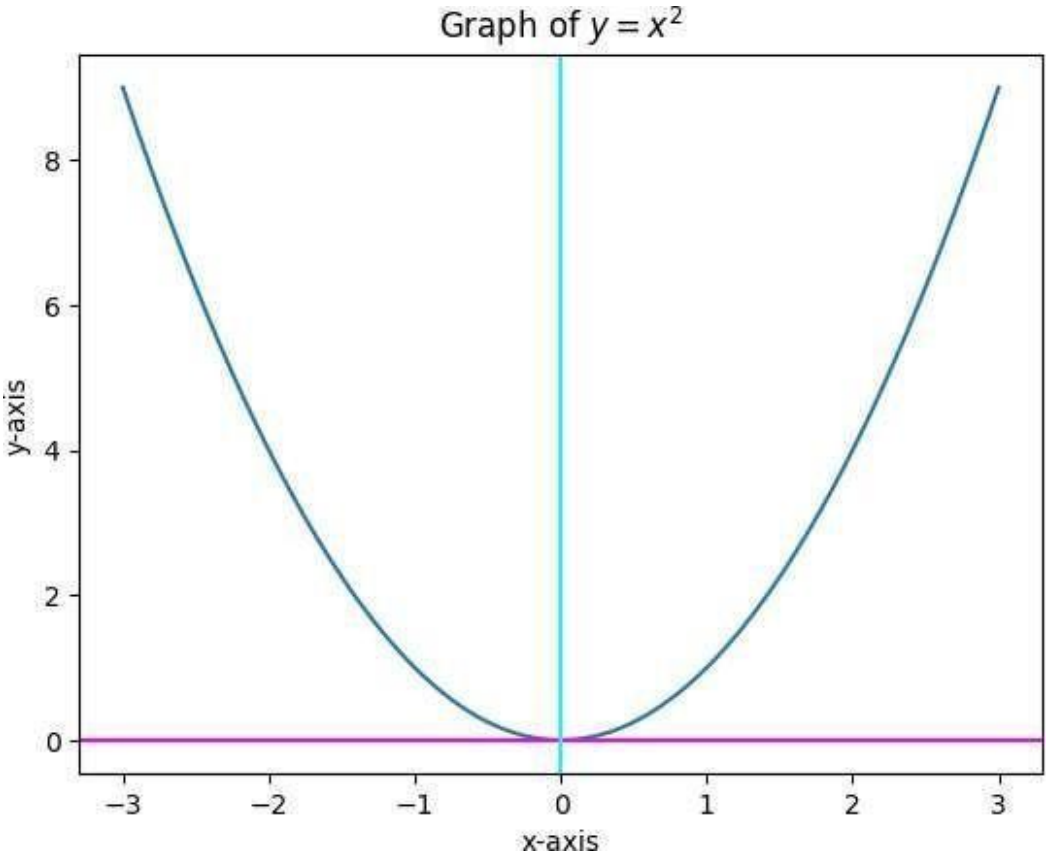
In [6]:
```
x=np.linspace(-3,3,100)
def f(x): return x**2
plt.plot(x,f(x))
plt.axhline(color='magenta')
plt.axvline(color='cyan')
```

Out[6]: <matplotlib.lines.Line2D at 0x1bc7cc78b90>



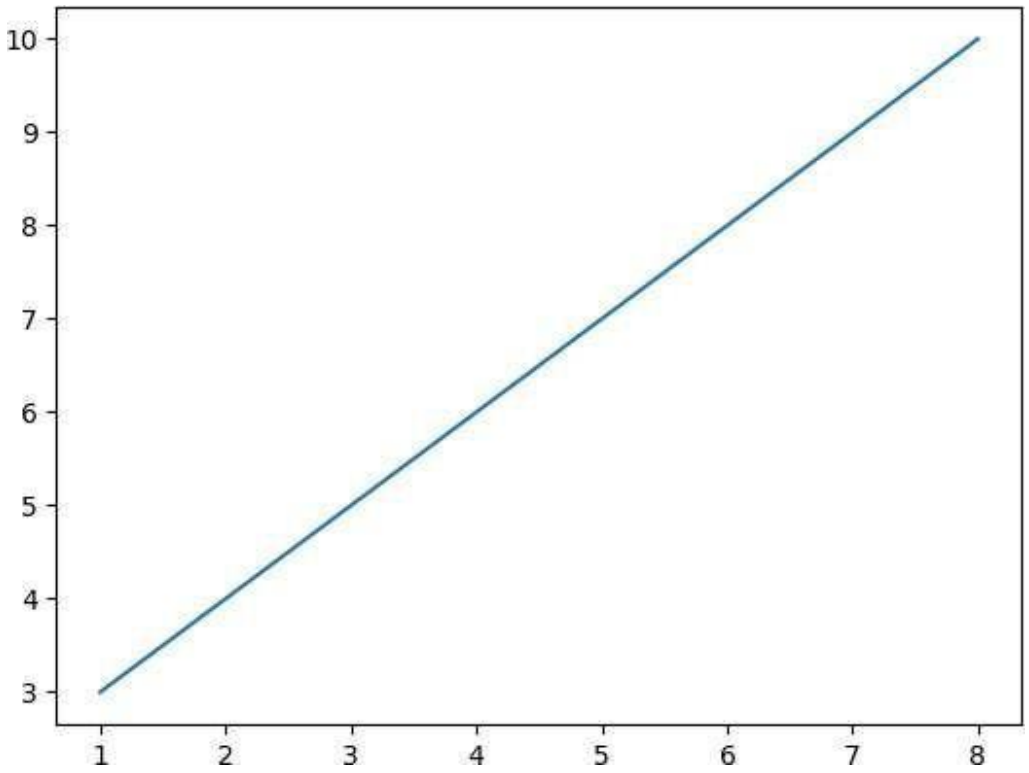
In [9]:
```
x=np.linspace(-3,3,100)
def f(x): return x**2
plt.plot(x,f(x))
plt.axhline(color='magenta')
plt.axvline(color='cyan')
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.title('Graph of $y=x^2$')
```

Out[9]: Text(0.5, 1.0, 'Graph of $y=x^2$')



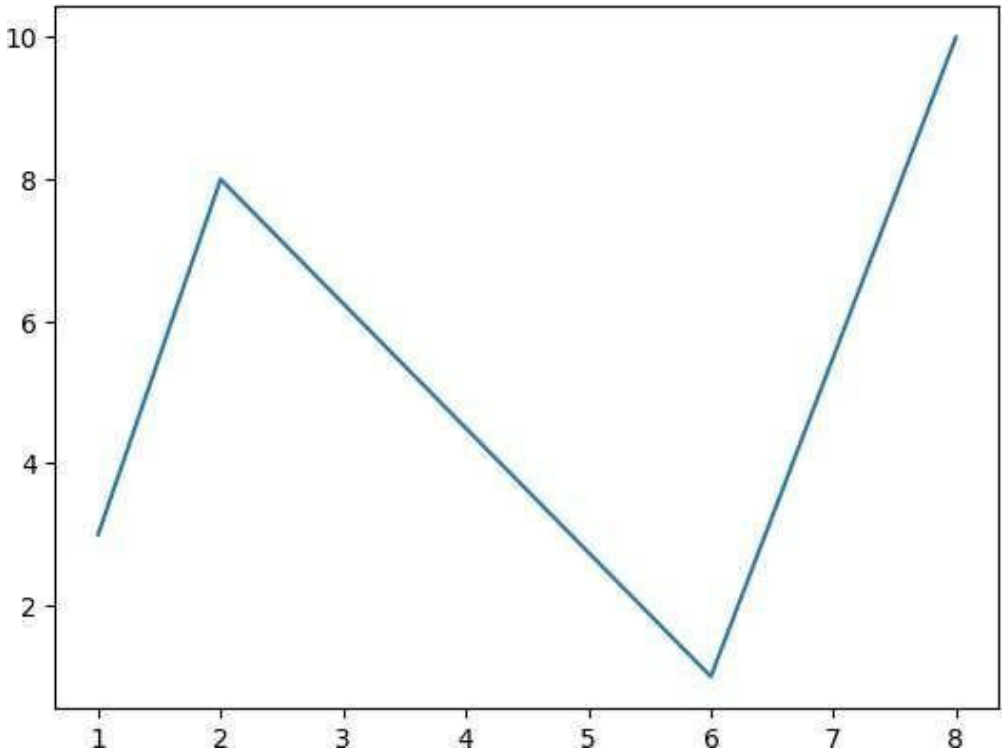
In [17]:
```
x=[1,8]
y=[3,10]
plt.plot(x,y)
```

Out[17]: [<matplotlib.lines.Line2D at 0x1bc7a7ed1d0>]



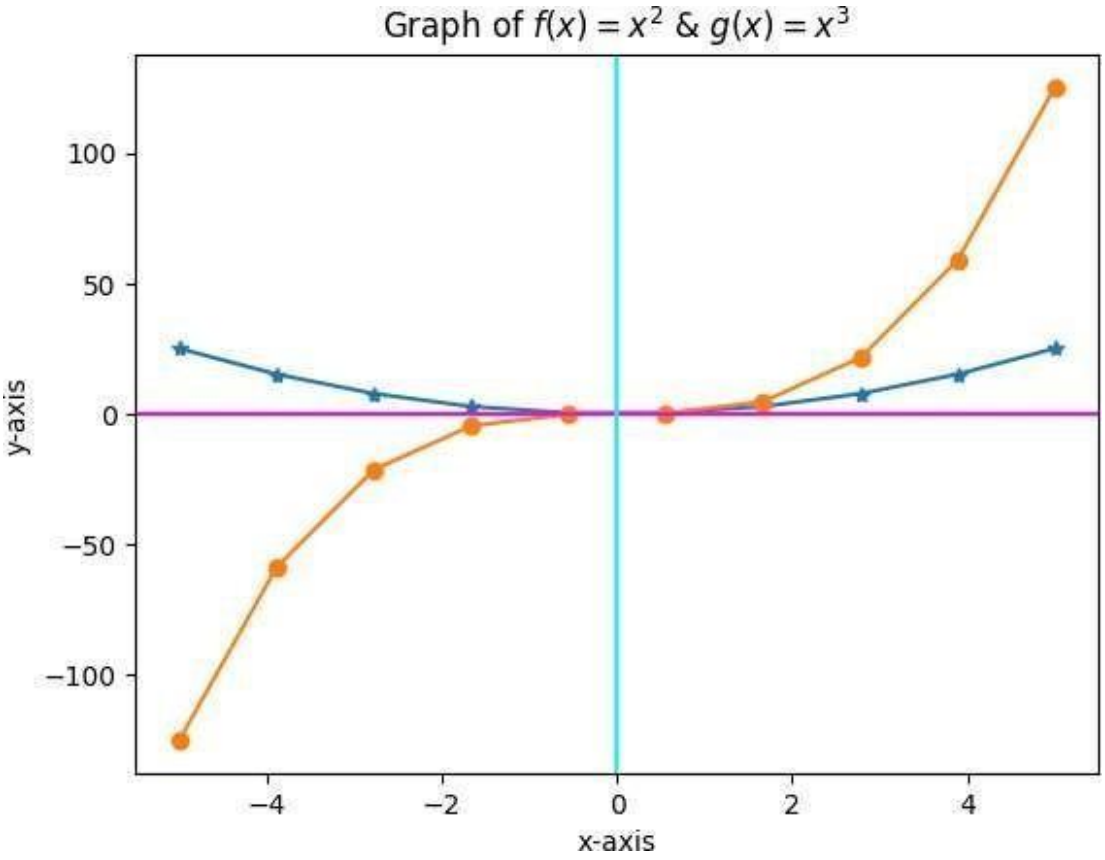
In [18]:
```
x=[1,2,6,8]
y=[3,8,1,10]
plt.plot(x,y)
```

Out[18]: [<matplotlib.lines.Line2D at 0x1bc7942d1d0>]



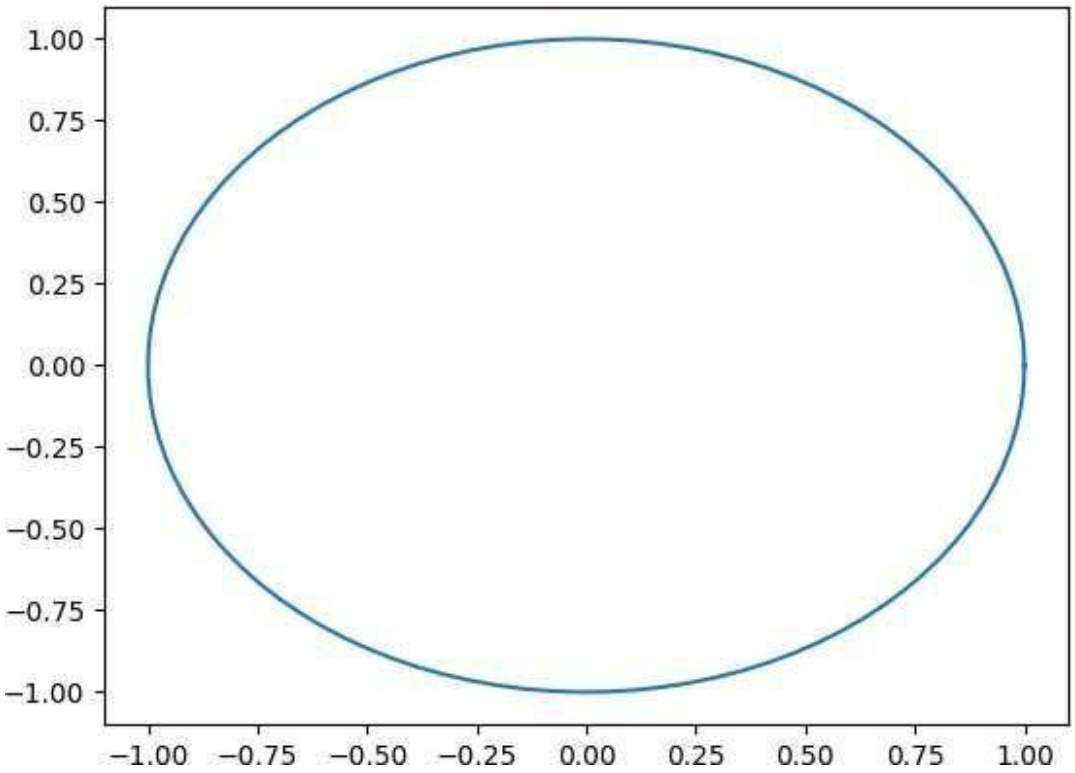
In [38]:
```
x=np.linspace(-5,5,10)
def f(x): return x**2
plt.plot(x,f(x),'*-')
def g(x): return x**3
plt.plot(x,g(x),'o-')
plt.axhline(color='magenta')
plt.axvline(color='cyan')
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.title('Graph of $f(x)=x^2$ & $g(x)=x^3$')
```

Out[38]: Text(0.5, 1.0, 'Graph of $f(x)=x^2$ & $g(x)=x^3$')



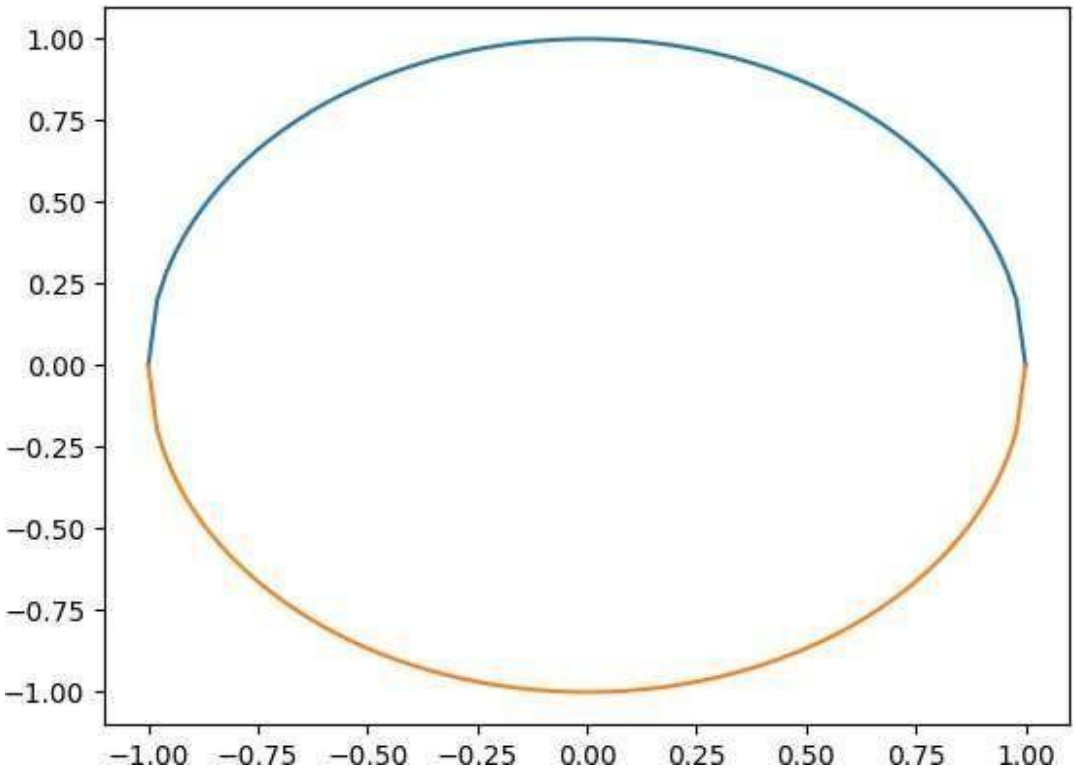
In [6]:
```
import matplotlib.pyplot as plt
import numpy as np
x=np.linspace(0,2*np.pi,100)
plt.plot(np.cos(x),np.sin(x))
```

Out[6]: [<matplotlib.lines.Line2D at 0x2703a5ae990>]



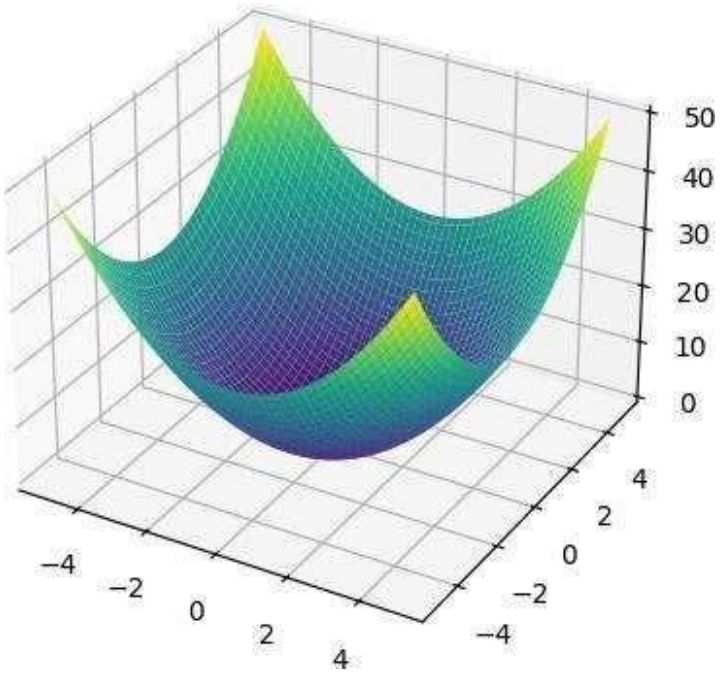
In [4]:
```
import matplotlib.pyplot as plt
import numpy as np
x=np.linspace(-1,1,100)
def f(x):return (1-x**2)**(1/2)
def g(x):return -(1-x**2)**(1/2)
plt.plot(x,f(x))
plt.plot(x,g(x))
```

Out[4]:
```
[<matplotlib.lines.Line2D at 0x2703680f110>]
```



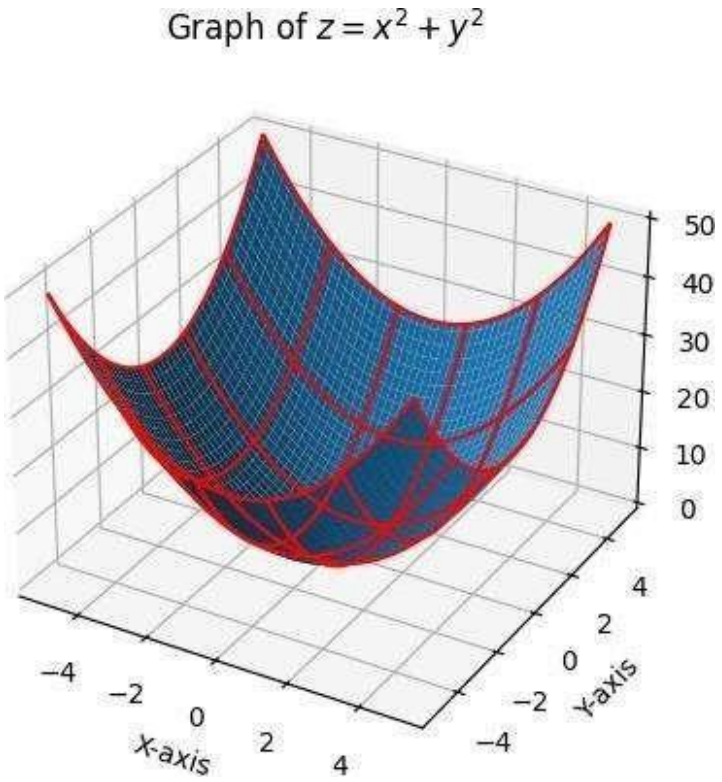
In [13]:
```
from numpy import *
from mpl_toolkits import mplot3d
x=linspace(-5,5,100)
y=linspace(-5,5,100)
X,Y= meshgrid(x,y)
def f(x,y): return x**2+y**2
D3=plt.axes(projection='3d')
D3.plot_surface(X,Y,f(X,Y),cmap='viridis')
```

Out[13]: <mpl_toolkits.mplot3d.art3d.Poly3DCollection at 0x2703d10ae90>



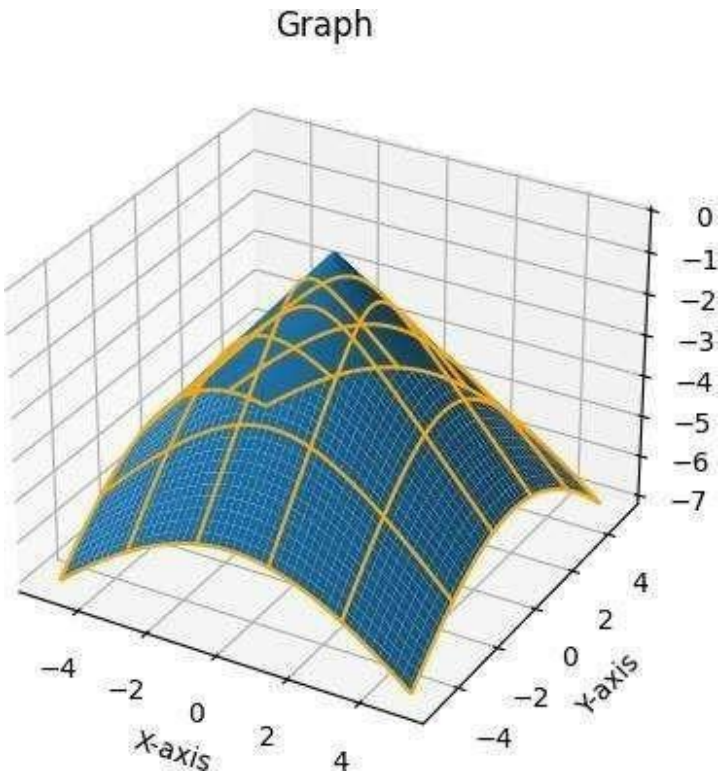
In [16]:
```
from numpy import *
from mpl_toolkits import mplot3d
x=linspace(-5,5,100)
y=linspace(-5,5,100)
X,Y= meshgrid(x,y)
def f(x,y): return x**2+y**2
D3=plt.axes(projection='3d')
D3.plot_surface(X,Y,f(X,Y))
D3.plot_wireframe(X,Y,f(X,Y),rcount=5,ccount=5,color='red')
D3.set_xlabel('X-axis')
D3.set_ylabel('Y-axis')
D3.set_title('Graph of $z=x^2+y^2$')
```

Out[16]:
```
Text(0.5, 0.92, 'Graph of $z=x^2+y^2$')
```



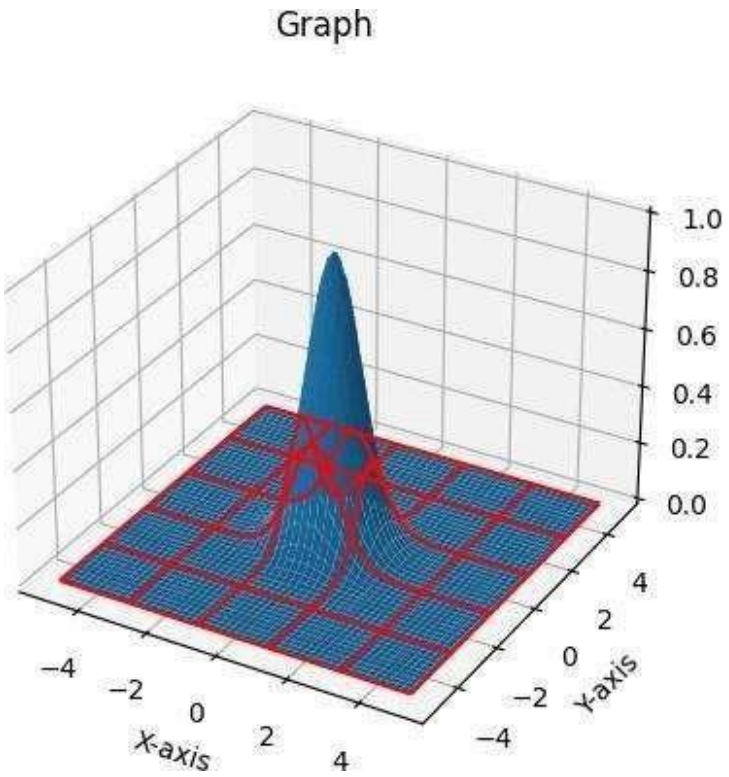
In [29]:
```
from numpy import *
from mpl_toolkits import mplot3d
x=linspace(-5,5,100)
y=linspace(-5,5,100)
X,Y= meshgrid(x,y)
def f(x,y): return -sqrt(x**2+y**2)
D3=plt.axes(projection='3d')
D3.plot_surface(X,Y,f(X,Y))
D3.plot_wireframe(X,Y,f(X,Y),rcount=5,ccount=5,color='orange')
D3.set_xlabel('X-axis')
D3.set_ylabel('Y-axis')
D3.set_title('Graph')
```

Out[29]:
```
Text(0.5, 0.92, 'Graph')
```



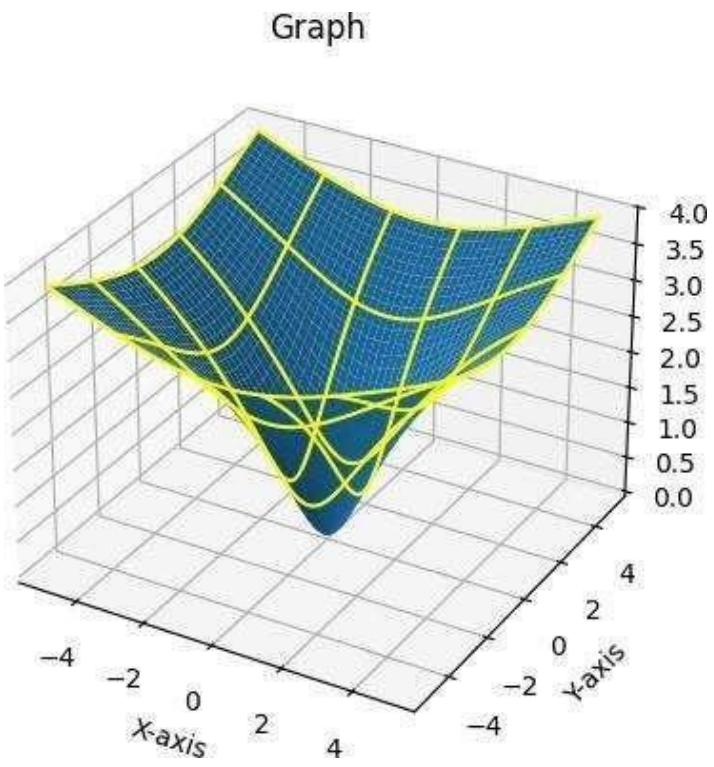
In [26]:
```
from numpy import *
from mpl_toolkits import mplot3d
x=linspace(-5,5,100)
y=linspace(-5,5,100)
X,Y= meshgrid(x,y)
def f(x,y): return e**(-(x**2+y**2))
D3=plt.axes(projection='3d')
D3.plot_surface(X,Y,f(X,Y))
D3.plot_wireframe(X,Y,f(X,Y),rcount=5,ccount=5,color='red')
D3.set_xlabel('X-axis')
D3.set_ylabel('Y-axis')
D3.set_title('Graph')
```

Out[26]:
```
Text(0.5, 0.92, 'Graph')
```



In [28]:
```
from numpy import *
from mpl_toolkits import mplot3d
x=linspace(-5,5,100)
y=linspace(-5,5,100)
X,Y= meshgrid(x,y)
def f(x,y): return log(x**2+y**2+1)
D3=plt.axes(projection='3d')
D3.plot_surface(X,Y,f(X,Y))
D3.plot_wireframe(X,Y,f(X,Y),rcount=5,ccount=5,color='yellow')
D3.set_xlabel('X-axis')
D3.set_ylabel('Y-axis')
D3.set_title('Graph')
```

Out[28]:
```
Text(0.5, 0.92, 'Graph')
```


Graph
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−4
−2
0
2
4
X-axis
−4
−2
0
2
4
Y-axis

```
import numpy as np
```

```
A=np.array([[2,5,7],[1,2,3],[7,8,2]])
```

```
m,n=np.linalg.eig(A)
```

```
print("Eigen values are :",m)
print("Eigen vectors are :",n)
```

```
Eigen values are : [11.72403766 -5.52328145 -0.2007562 ]
Eigen vectors are : [[-0.65205856 -0.57279484  0.68802677]
 [-0.2839392  -0.23679297 -0.67518945]
 [-0.70299229  0.78475166  0.26596687]]
```

```
from sympy import *
A=Matrix([[1,5,7],[0,2,3],[0,0,3]])
display(A)
P,D=A.diagonalize()

display("Diagonal matrix:",D)
display("Eigen vectors:",P)
```

```
'Diagonal matrix:'
'Eigen vectors:'
```

LU Decomposition

```
import scipy
import numpy as np
import scipy.linalg
M=np.array([[7,3,-1,2],[3,8,1,-4],[-1,1,4,-1],[2,4,-1,6]])
```

```
I,L,U=scipy.linalg.lu(M)
```

I → Permutation matrix (sometimes called P), representing row swaps. L →

Lower triangular matrix (with unit diagonal).

U → Upper triangular matrix.

```
print(I)
print(L)
print(U)
```

```
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]
[[ 1.          0.          0.          0.        ]
 [ 0.42857143  1.          0.          0.        ]
 [-0.14285714  0.21276596  1.          0.        ]
 [ 0.28571429  0.46808511 -0.38922156  1.        ]]
[[ 7.          3.         -1.          2.        ]
 [ 0.          6.71428571  1.42857143 -4.85714286]
 [ 0.          0.          3.55319149  0.31914894]
 [ 0.          0.          0.          7.82634731]]
```

```
import sympy
M=Matrix([[7,3,-1,2],[3,8,1,-4],[-1,1,4,-1],[2,4,-1,6]])
L,U,P=M.LUdecomposition()

#L → Lower triangular matrix (with 1s on the diagonal).
#U → Upper triangular matrix.
#P → Permutation matrix (accounts for row swaps during decomposition).
display(L)
display(U)
display(P)
```

```
# Convert perm list to permutation matrix
n = M.shape[0]
P = Matrix.eye(n)   # start with identity
```

```
for i, j in perm:
    P.row_swap(i, j)

print("Permutation matrix P =")
display(P)
```

```
Permutation matrix P =
```

```
np.dot(L,U)
```

```
array([[7, 3, -1, 2],
       [3, 8, 1, -4],
       [-1, 1, 4, -1],
       [2, 4, -1, 6]], dtype=object)
```

LU Decomposition

```
M=np.array([[7,3,-1,2],[3,8,1,-4],[-1,1,4,-1],[2,4,-1,6]])
Q,R=np.linalg.qr(M)
print(Q)
print(R)
```

```
[[-0.8819171   0.40483887 -0.24146029  0.0049642 ]
 [-0.37796447 -0.805051    0.02132945 -0.45670646]
 [ 0.12598816 -0.26603697 -0.89957402  0.32267304]
 [-0.25197632 -0.34237801  0.36332983  0.82902151]]
[[-7.93725393 -6.5513842   1.25988158 -1.88982237]
 [ 0.         -6.86144045 -1.93165974  2.24165064]
 [ 0.          0.         -3.69883618  2.51131461]
 [ 0.          0.          0.          6.48821029]]
```

```
import sympy as sp

M = sp.Matrix([[7,3,-1,2],
               [3,8,1,-4],
               [-1,1,4,-1],
               [2,4,-1,6]])

Q, R = M.QRdecomposition()

display(Q)
display(R)
display(Q*R)
```

singular value decomposition

```
from scipy.linalg import svd
M=np.array([[7,3,-1,2],[3,8,1,-4],[-1,1,4,-1],[2,4,-1,6]])
U,S,Vh=svd(M)
print(U)
print(S)
print(Vh)
```

```
[[-0.60431772  0.32041184 -0.53519869  0.49568004]
 [-0.66653766 -0.67198996  0.05593269 -0.31784994]
 [ 0.00776422 -0.33651863  0.51804203  0.78633796]
 [-0.43642566  0.57664631  0.6648807  -0.1869367 ]]
[11.44552268  8.25768894  4.5240679   3.05669653]
[[-0.62124315 -0.77612819  0.03540796 -0.1021188 ]
 [ 0.20789404 -0.29604137 -0.35301918  0.86285401]
 [-0.61158971  0.44637489  0.44173051  0.4812293 ]
 [ 0.44361587 -0.33276717  0.82401328  0.11607358]]
```

```
s=np.diag(S)
print(s)
```

```
[[11.44552268  0.          0.          0.        ]
 [ 0.          8.25768894  0.          0.        ]
 [ 0.          0.          4.5240679   0.        ]
 [ 0.          0.          0.          3.05669653]]
```

```
import sympy as sp
M = sp.Matrix([[7,3,-1,2],[3,8,1,-4],[-1,1,4,-1],[2,4,-1,6]])
U, S, Vh = M.singular_value_decomposition()
display(U)
display(S)
display(Vh)
```

```
# solving a linear system of equations.
import numpy as np
A=np.array([[1,2],[3,4]]) #x+2y=8 & 3x+4y=18
B=np.array([[8],[18]])
print(np.linalg.solve(A,B))
```

```
[[2.]
 [3.]]
```

```
from sympy import Symbol,solve
x=Symbol('x')
y=Symbol('y')
z=Symbol('z')
expr1=x+2*y-8
expr2=3*x+4*y-18
solve((expr1,expr2))
```

```
{x: 2, y: 3}
```

```
#solve the system of equations 2x+3y+z = 6, 3x+2y+2z = 12, 4x+y+3z = 18 for both numpy and sympy
from sympy import Symbol,solve
x=Symbol('x')
y=Symbol('y')
z=Symbol('z')
expr1=2*x+3*y+z-6
expr2=3*x+2*y+2*z-12
expr3=4*x+y+3*z-18
solve((expr1,expr2,expr3))
```

```
{x: 24/5 - 4*z/5, y: z/5 - 6/5}
```

```
import numpy as np
import sympy as sp
A = np.array([[1,2],[3,4]])
S = sp.Matrix(A)
S.rref()#reduced row echelon form.
```

```
(Matrix([
 [1, 0],
 [0, 1]]),
 (0, 1))
```

```
import numpy as np
import sympy as sp
A = np.array([[1,2,0],[1,2,3]])
S = sp.Matrix(A)
S.rref()[0]#only matrix shown
```

```
S.rref()[1]
```

```
(0, 2)
```

```
A = np.array([[1,2],[0,4]])
m,n = np.linalg.eig(A)
print("The eigenvalues are:",m)
print("The eigenvectors are:",n)
```

```
The eigenvalues are: [1. 4.]
The eigenvectors are: [[1.         0.5547002 ]
 [0.         0.83205029]]
```

```
A = np.array([[1,2],[0,3]])
m,n = np.linalg.eig(A)
print("The eigenvalues are:",m)
print("The eigenvectors are:",n)
```

```
The eigenvalues are: [1. 3.]
The eigenvectors are: [[1.         0.70710678]
```

```
A = np.array([[4,0,0],[0,0,0],[0,0,-1]])
m,n = np.linalg.eig(A)
print("The eigenvalues are:",m)
print("The eigenvectors are:",n)
```

```
The eigenvalues are: [ 4.  0. -1.]
The eigenvectors are: [[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

```
A = np.array([[4,2,-2],[2,5,0],[-2,0,3]])
m,n = np.linalg.eig(A)
print("The eigenvalues are:",m)
print("The eigenvectors are:",n)
```

```
The eigenvalues are: [1. 7. 4.]
The eigenvectors are: [[ 0.66666667 -0.66666667 -0.33333333]
 [-0.33333333 -0.66666667  0.66666667]
 [ 0.66666667  0.33333333  0.66666667]]
```

```
A = np.array([[-2,-2,-3],[2,1,-6],[-1,-2,0]])
m,n = np.linalg.eig(A)
print("The eigenvalues are:",m)
print("The eigenvectors are:",n)
```

```
The eigenvalues are: [-1.82842712  3.82842712 -3.        ]
The eigenvectors are: [[-9.33647701e-01 -7.65048437e-02  9.48683298e-01]
 [ 3.20377241e-01  8.91805812e-01  4.96506831e-17]
 [-1.60188621e-01 -4.45902906e-01  3.16227766e-01]]
```

```
A = np.array([[1,1,1,1],[1,2,3,4],[1,5,9,10]])
B = sp.Matrix(A)
B.rref()[0]
```

```
Start coding or ge_nerate with AI.
```