VIT BHOPAL

DATA PLOTTING USING PYTHON

Computational Physics

All Modules

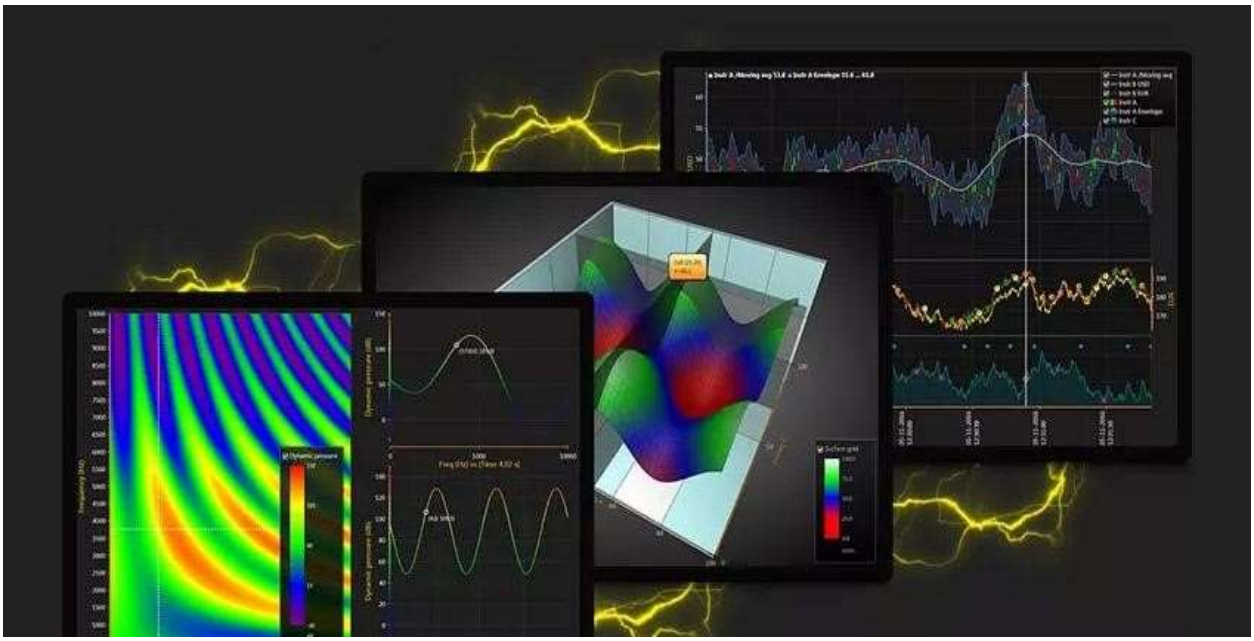
Submitted by ujjwal srivastava(20MIP10020)

Student from VIT-Bhopal

Integrated MTech CSE Specialization in Computational and Data Science

Project done under the guidance of Rajdeep Payal SASL

(100206)



Content

Module 1

- Basic line graph
- Basic line graph with axis labels
- My first graph
- My first graph Dashed Line
- Two lines on same graph
- Two lines on same graph with legend
- Bar graph plots
- My Bar Chart
- My Histogram
- My Histogram with Line Partition
- Pie Chart
- Pie Chart with Labels
- Scatter Plot
- Contour Plots
- 3D Plot

Module 2

- Displaying vectors in Horizontal and Vertical
- All operation in Matrix
- Dot Product 1
- Dot Product 2
- Cross Product
- Cross Product 2
- Printing Matrix
- Addition 3 X 3 matrix
- Subtraction 3 X 3 matrix
- Dot Product of 3 X 3 matrix 1
- Dot Product of 3 X 3 matrix 2
- Matrix Identity
- Transpose of Matrix
- Matrix Identity 2
- Identity Prove (AB)' = B'A'
- The (multiplicative) inverse of a Matrix
- Inverse of a
- Solving equations X values

Module 3

- *Graph of y=2x+1*
- *Graph for Multiple lines*

- *Solid Line Graph*
- *Simple Parabola y= x^2*
- *Graph for x^2 & x^3*
- *cos(x) plot*
- *One wave of cos(x) graph*
- *cos(x) & sin (x) plots*
- *One wave of cos(x) & sin (x) plots*
- *tan(x) plot*
- *cos^2(x) & sin^2(x) plot*
- *exp(x) plot*
- *Steeper exp(x) plot*
- *log(x) plot*
- *log(x) & log10(x)*
- *Finding the Mean of given data*
- *Finding the Mean of given data 2*
- *Finding the Median of given data*
- *Finding the Mode of given data*
- *Finding the Standard Deviation of sample*
- *Random number Decimal*
- *Random number Integer*
- *All Stats calculation*
- *All Stats calculation 2*
- *All Stats calculation with Graph*

Module 4:

- Freely Falling Body
- Collision of two particles at head on collision
- Collision of two particles at deflection at an angle
- Projectile motion using U_X and U_Y
- Projectile motion using equation of trajectory
- Projectile motion in 3D

Module 1:

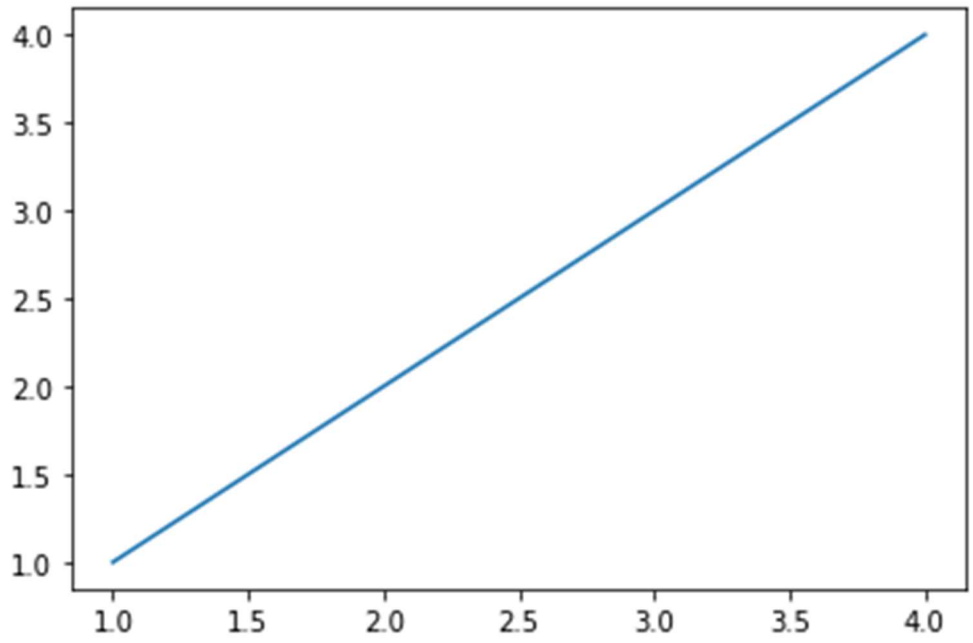
- Basic line graph
- Basic line graph with axis labels
- My first graph
- My first graph Dashed Line
- Two lines on same graph
- Two lines on same graph with legend
- Bar graph plots
- My Bar Chart
- My Histogram
- My Histogram with Line Partition
- Pie Chart
- Pie Chart with Labels
- Scatter Plot
- Contour Plots
- 3D Plot

Basic line graph

Code:

```
import matplotlib.pyplot as plt
x = [1,2,3,4]
y = [1,2,3,4]
plt.plot(x, y)
plt.show()
```

Output:

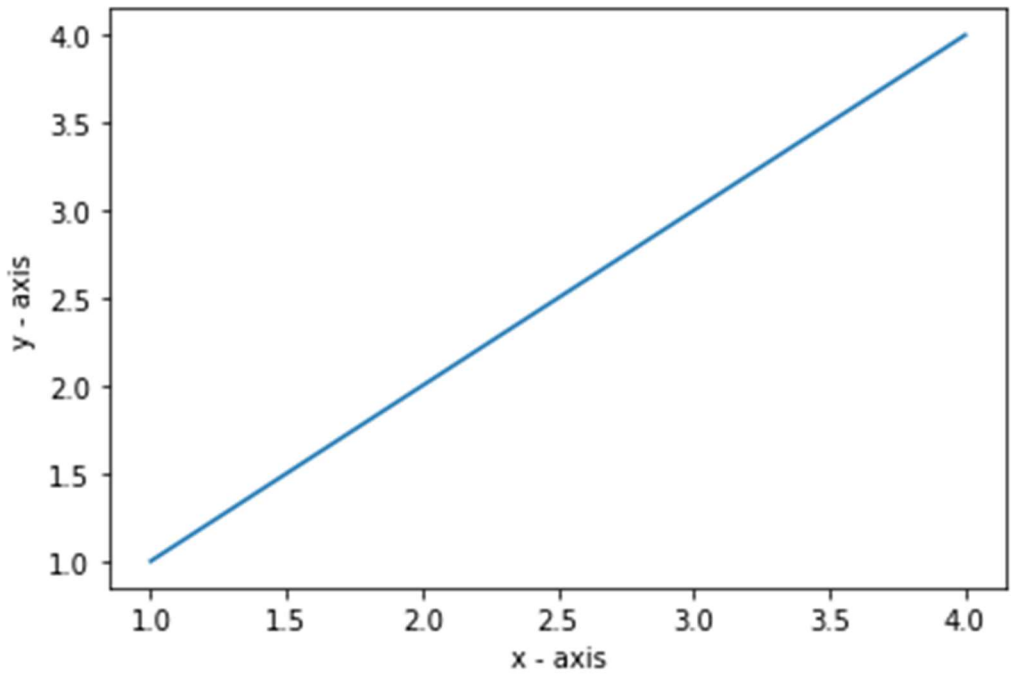


Basic line graph with axis labels

Code:

```
import matplotlib.pyplot as plt
x = [1,2,3,4]
y = [1,2,3,4]
plt.plot(x, y)
plt.xlabel('x - axis')
plt.ylabel('y - axis')
plt.show()
```

Output:

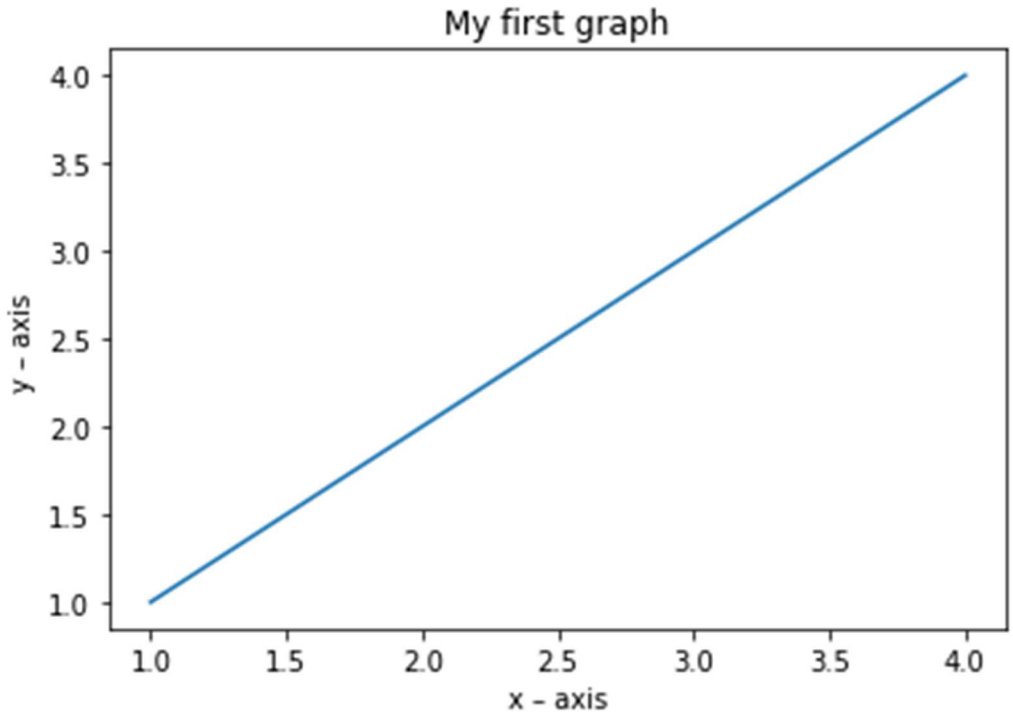


My first graph

Code:

```
import matplotlib.pyplot as plt
x = [1,2,3,4]
y = [1,2,3,4]
plt.plot(x, y)
plt.xlabel('x - axis')
plt.ylabel('y - axis')
plt.title('My first graph')
plt.show()
```

Output:

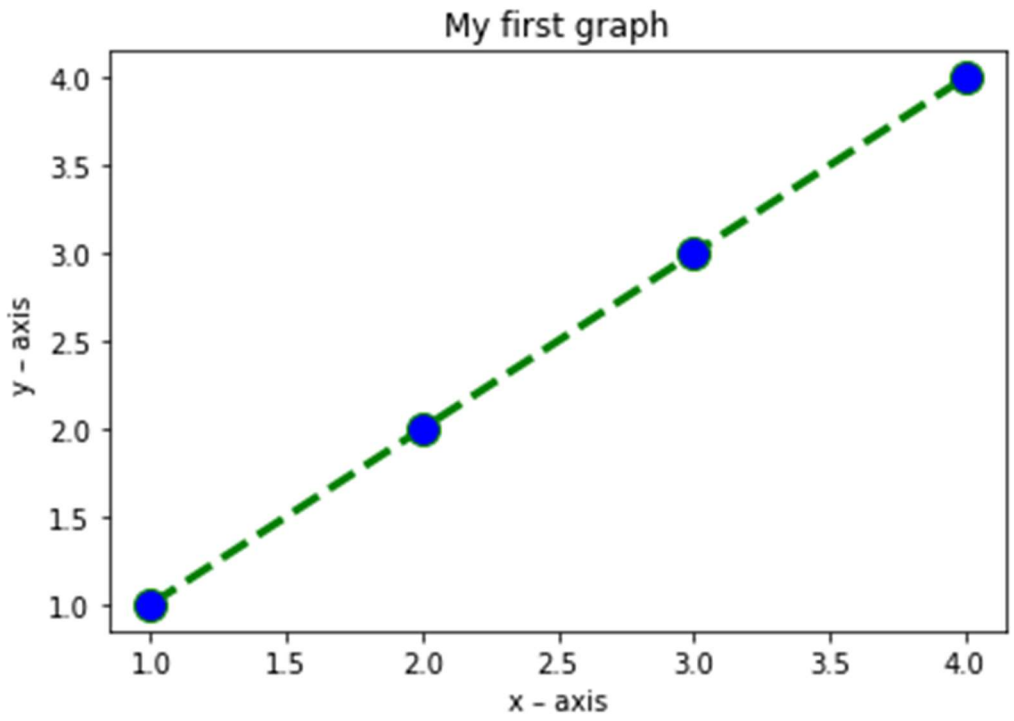


My first graph Dashed Line

Code:

```
import matplotlib.pyplot as plt
x = [1,2,3,4]
y = [1,2,3,4]
plt.plot(x, y, color='green', linestyle='dashed', linewidth = 3,
marker='o', markerfacecolor='blue', markersize=12)
plt.xlabel('x - axis')
plt.ylabel('y - axis')
plt.title('My first graph')
plt.show()
```

Output:

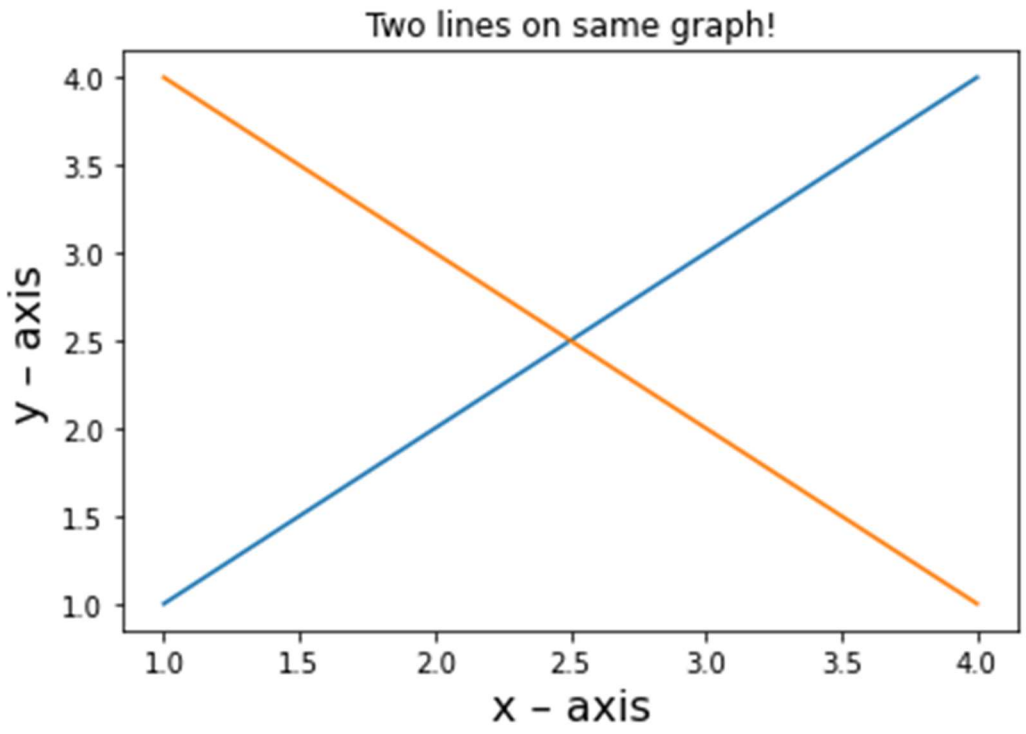


Two lines on same graph

Code:

```
import matplotlib.pyplot as plt
x1 = [1,2,3,4]
y1 = [1,2,3,4]
plt.plot(x1, y1)
x2 = [1,2,3,4]
y2 = [4,3,2,1]
plt.plot(x2,y2)
plt.xlabel('x - axis', fontsize=16)
plt.ylabel('y - axis', fontsize=16)
plt.title('Two lines on same graph!')
plt.show()
```

Output:

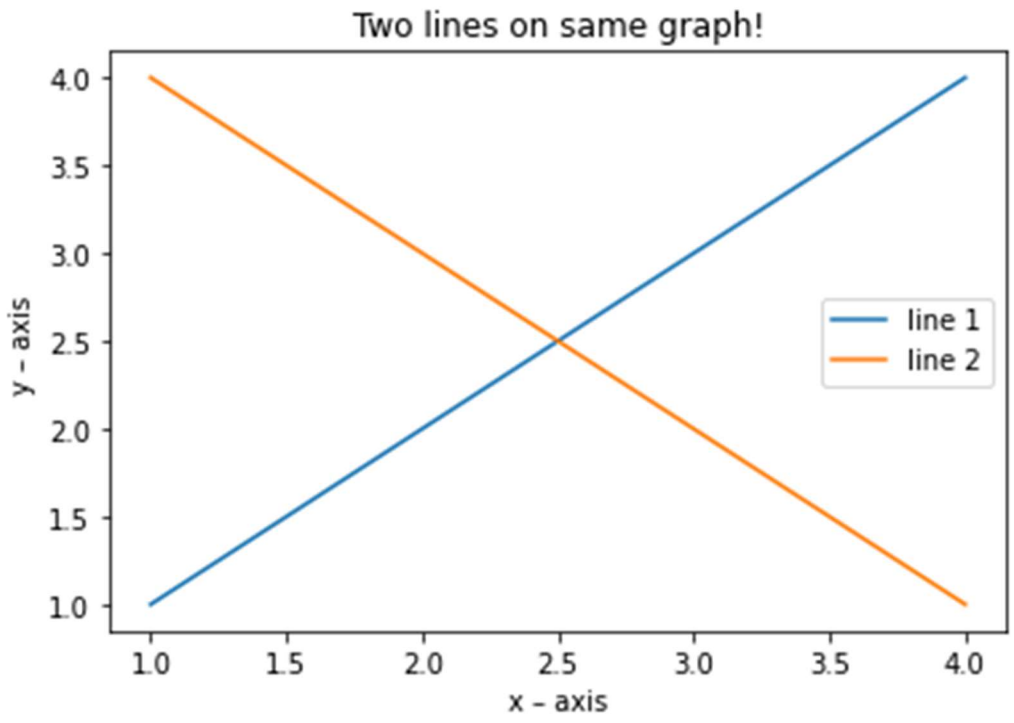


Two lines on same graph with legend

Code:

```
import matplotlib.pyplot as plt
x1 = [1,2,3,4]
y1 = [1,2,3,4]
plt.plot(x1, y1, label = "line 1")
x2 = [1,2,3,4]
y2 = [4,3,2,1]
plt.plot(x2,y2, label = "line 2")
plt.xlabel('x - axis')
plt.ylabel('y - axis')
plt.title('Two lines on same graph!')
plt.legend()
plt.show()
```

Output:

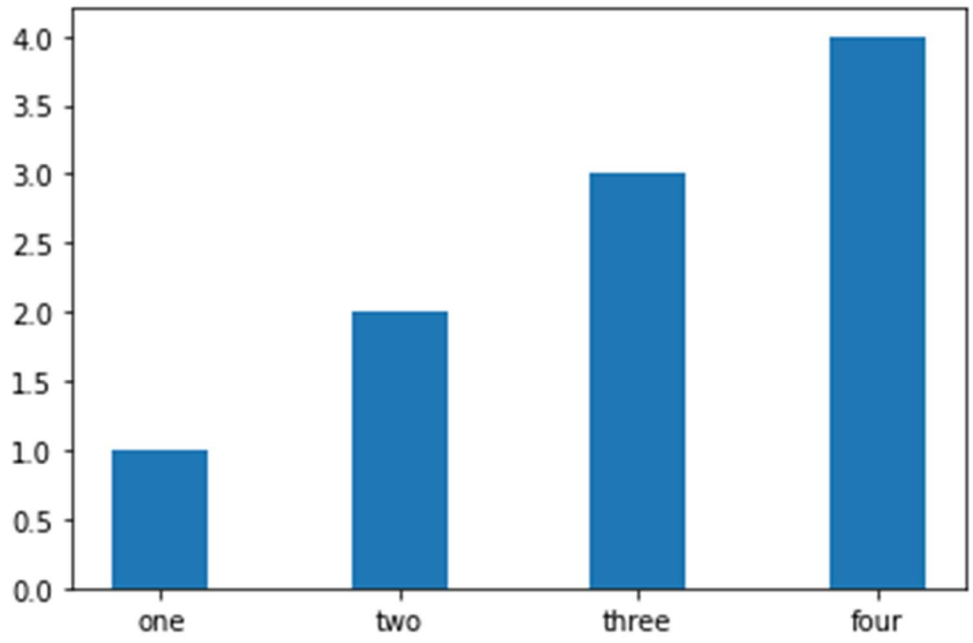


Bar graph plots

Code:

```
import matplotlib.pyplot as plt
import numpy as np
y = [1,2,3,4]
abc = ['one', 'two', 'three', 'four']
plt.bar(np.arange(len(y)),y,tick_label = abc, width=0.4)
plt.show()
```

Output:

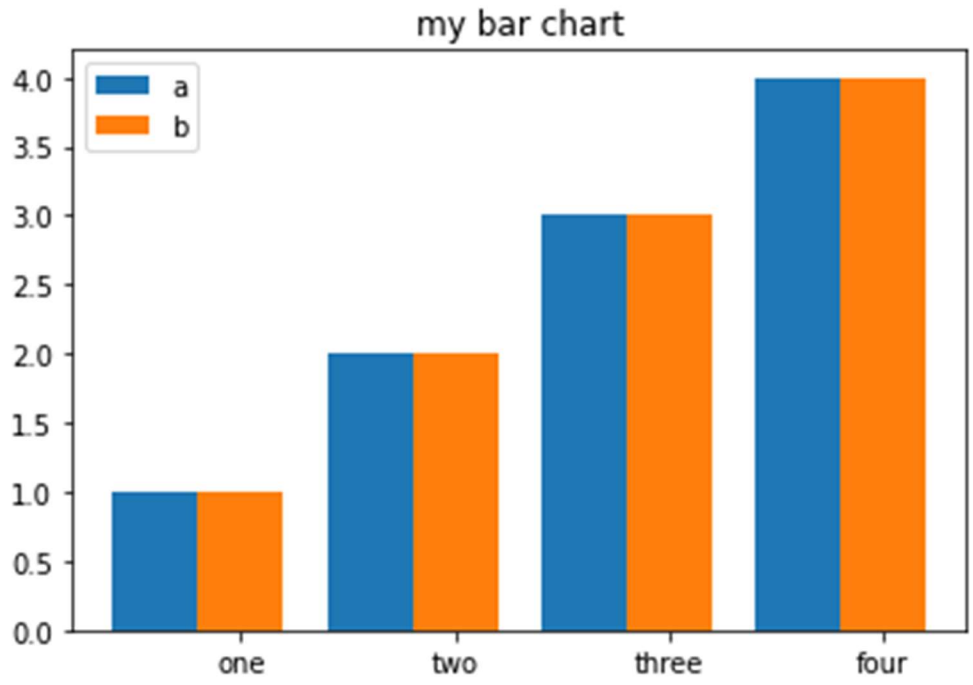


My Bar Chart

Code:

```
import matplotlib.pyplot as plt
import numpy as np
y1 = [1,2,3,4]
y2 = [1,2,3,4]
tick_label = ['one', 'two', 'three', 'four']
plt.bar(np.arange(len(y1))-0.2,y1,tick_label = tick_label, width=0.4)
plt.bar(np.arange(len(y2))+0.2,y2,tick_label = tick_label, width=0.4)
plt.title('my bar chart')
plt.legend(labels=['a','b'])
plt.show()
```

Output:

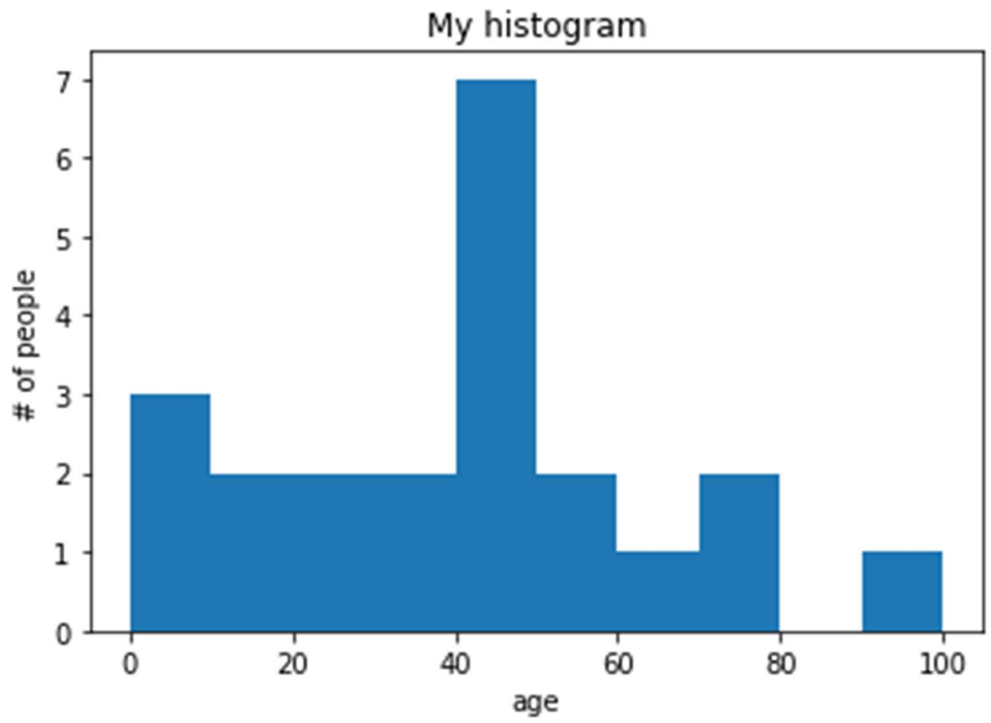


My Histogram

Code:

```
import matplotlib.pyplot as plt
ages =[2,5,70,40,30,45,50,45,43,40,44,60,7,13,57,18,90,77,32,21,20,40]
range = (0, 100)
bins = 10
plt.hist(ages, bins, range)
plt.xlabel('age')
plt.ylabel('# of people')
plt.title('My histogram')
plt.show()
```

Output:

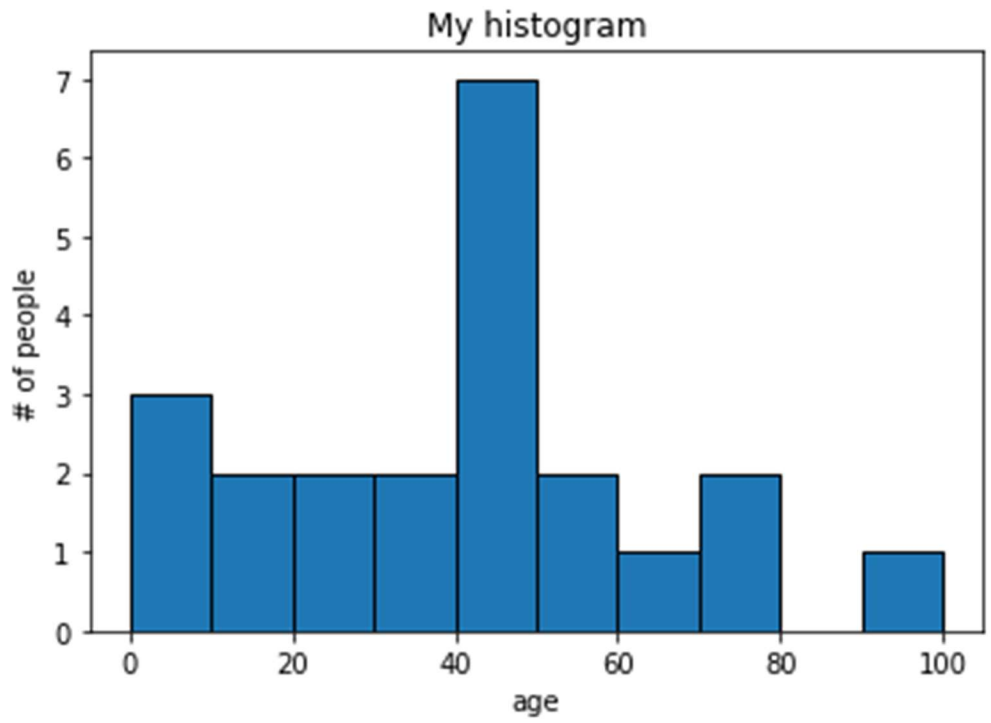


My Histogram with Line Partition

Code:

```
import matplotlib.pyplot as plt
ages =[2,5,70,40,30,45,50,45,43,40,44,60,7,13,57,18,90,77,32,21,20,40]
range = (0, 100)
bins = 10
plt.hist(ages, bins, range, edgecolor='black')
plt.xlabel('age')
plt.ylabel('# of people')
plt.title('My histogram')
plt.show()
```

Output:

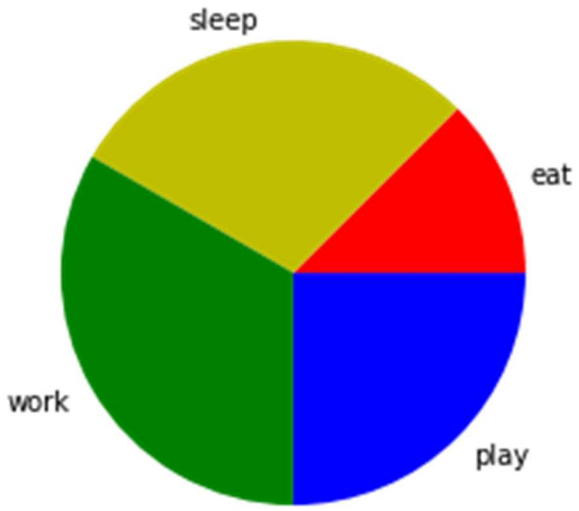


Pie Chart

Code:

```
import matplotlib.pyplot as plt
activities = ['eat', 'sleep', 'work', 'play']
slices = [3, 7, 8, 6]
colors = ['r', 'y', 'g', 'b']
plt.pie(slices, labels = activities, colors=colors)
plt.show()
```

Output:

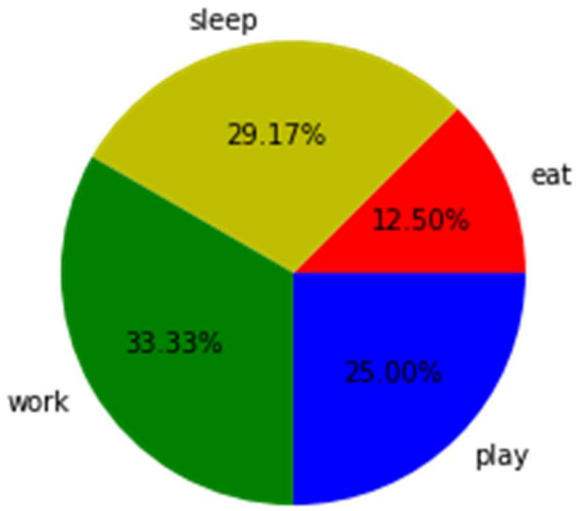


Pie Chart with Labels

Code:

```
import matplotlib.pyplot as plt
activities = ['eat', 'sleep', 'work', 'play']
slices = [3, 7, 8, 6]
colors = ['r', 'y', 'g', 'b']
plt.pie(slices, labels = activities, colors=colors,autopct = '%1.2f%%')
plt.show()
```

Output:

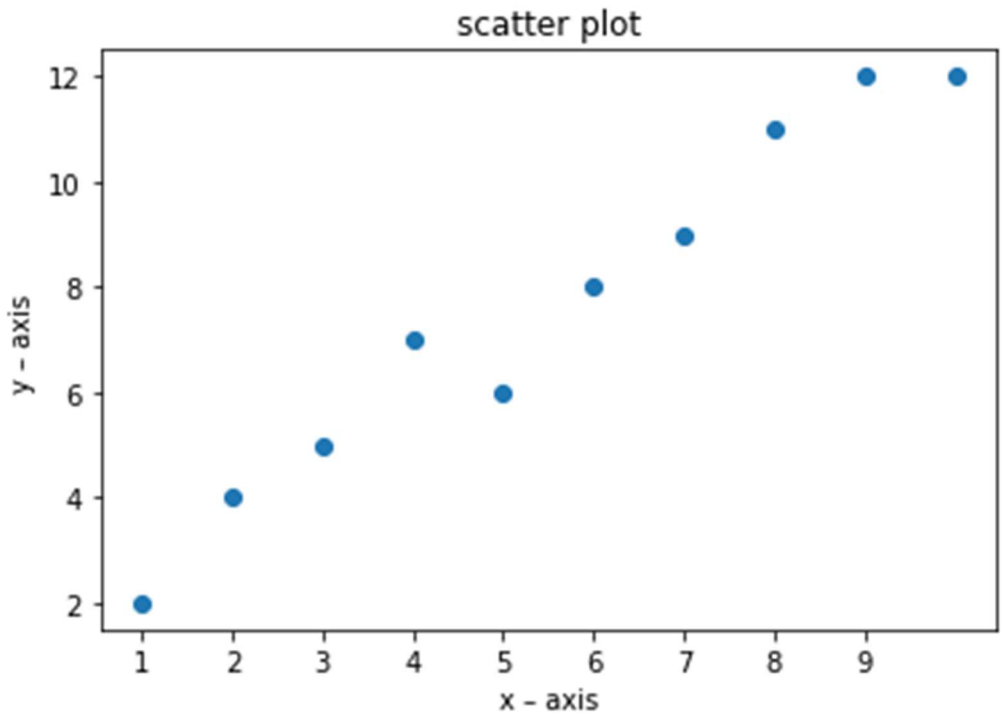


Scatter Plot

Code:

```
import matplotlib.pyplot as plt
import numpy as np
x = [1,2,3,4,5,6,7,8,9,10]
y = [2,4,5,7,6,8,9,11,12,12]
plt.scatter(x, y)
plt.xlabel('x - axis')
plt.ylabel('y - axis')
plt.title('scatter plot')
plt.xticks(np.arange(min(x), max(x),1.0))
plt.show()
```

Output:

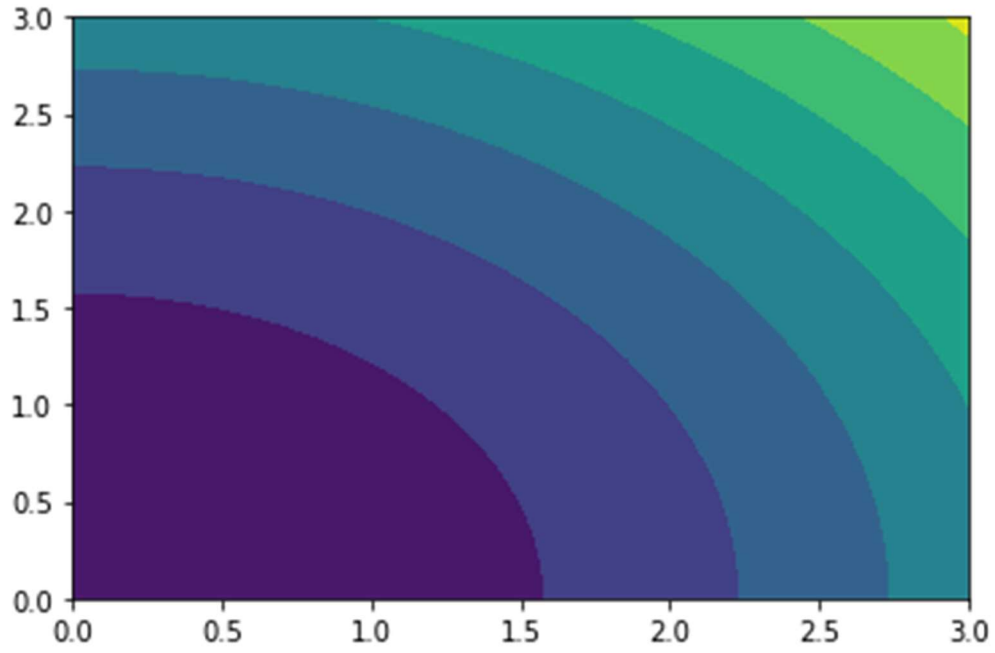


Contour Plots

Code:

```
import matplotlib.pyplot as plt
import numpy as np
feature_x = np.linspace(0, 3.0, 30)
feature_y = np.linspace(0, 3.0, 30)
[X, Y] = np.meshgrid(feature_x, feature_y)
Z = X ** 2 + Y ** 2
plt.contourf(X,Y,Z)
plt.show()
```

Output:

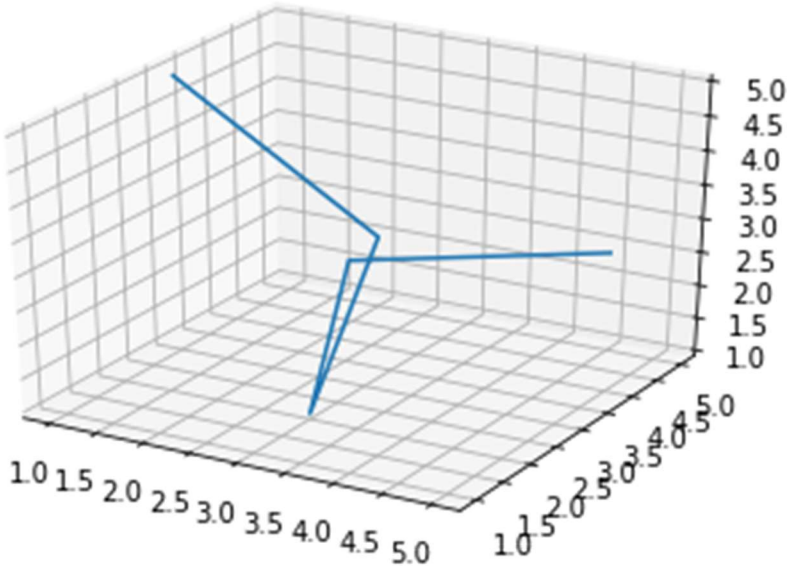


3D Plot

Code:

```
import matplotlib.pyplot as plt
import numpy as np
from mpl_toolkits import mplot3d
plt.axes(projection="3d")
x=[1,2,3,4,5]
y=[3,5,2,1,4]
z=[5,2,1,4,3]
plt.plot(x,y,z)
plt.show()
```

Output:



Module 2

- Displaying vectors in Horizontal and Vertical
- All operation in Matrix
- Dot Product 1
- Dot Product 2
- Cross Product
- Cross Product 2
- Printing Matrix
- Addition 3 X 3 matrix
- Subtraction 3 X 3 matrix
- Dot Product of 3 X 3 matrix 1
- Dot Product of 3 X 3 matrix 2
- Matrix Identity
- Transpose of Matrix
- Matrix Identity 2
- Identity Prove (AB)' = B'A'
- The (multiplicative) inverse of a Matrix
- Inverse of a
- Solving equations X values

Displaying vectors in Horizontal and Vertical

Code:

```
import numpy as np
list1 = [1, 2, 3]
list2 = [[10],[20],[30]]
vector1 = np.array(list1)
vector2 = np.array(list2)
print("Horizontal Vector")
print(vector1)
print("----------------")
print("Vertical Vector")
print(vector2)
```

Output:

```
Horizontal Vector
[1 2 3]
----------------
Vertical Vector
[[10]
 [20]
 [30]]
```

All operation in Matrix

Code:

```
import numpy as np
list1 = [5, 6, 9]
list2 = [1, 2, 3]
vector1 = np.array(list1)
print("First Vector : " + str(vector1))
vector2 = np.array(list2)
print("Second Vector : " + str(vector2))
addition = vector1 + vector2
print("Vector Addition : " + str(addition))
subtraction = vector1 - vector2
print("Vector Substraction : " + str(subtraction))
multiplication = vector1 * vector2
print("Vector Multiplication : " + str(multiplication))
division = vector1 / vector2
print("Vector Division : " + str(division))
```

Output:

```
First Vector : [5 6 9]
Second Vector : [1 2 3]
Vector Addition : [ 6  8 12]
Vector Substraction : [4 4 6]
Vector Multiplication : [ 5 12 27]
Vector Division : [5. 3. 3.]
```

Dot Product 1

Code:

```
import numpy as np
list1 = [5, 6, 9]
list2 = [1, 2, 3]
vector1 = np.array(list1)
print("First Vector : " + str(vector1))
vector2 = np.array(list2)
print("Second Vector : " + str(vector2))
dot_product = vector1.dot(vector2)
print("Dot Product : " + str(dot_product))
```

Output:

```
First Vector : [5 6 9]
Second Vector : [1 2 3]
Dot Product : 44
```

Dot Product 2

Code:

```
import numpy
import array
a=[5,6,9]
b=[1,3,4]
dot = 0.0;
for i in range(len(a)):
  dot += a[i] * b[i]
print("Dot Product = "+ str(dot))
```

Output:

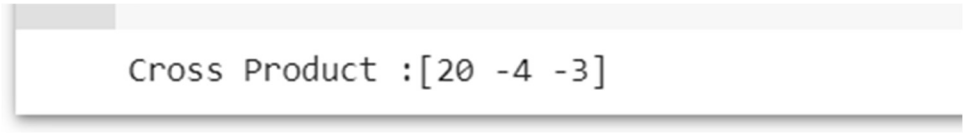
```
Dot Product = 59.0
```

Cross Product

Code:

```
import numpy as np
A =[2, 7, 4]
B =[3, 9, 8]
vector1=np.array(A)
vector2=np.array(B)
cross_product=np.cross(vector1, vector2)
print("Cross Product :" +str(cross_product))
```

Output:



Cross Product 2

Code:

```
vect_A = [3, -5, 4]
vect_B = [2, 6, 5]
cross_P = [ ]
cross_P.append(vect_A[1] * vect_B[2] - vect_A[2] * vect_B[1])
cross_P.append(vect_A[2] * vect_B[0] - vect_A[0] * vect_B[2])
cross_P.append(vect_A[0] * vect_B[1] - vect_A[1] * vect_B[0])
print("Cross product:", end =" ")
for i in range(0, 3):
  print(cross_P[i], end =" ")
```

Output:

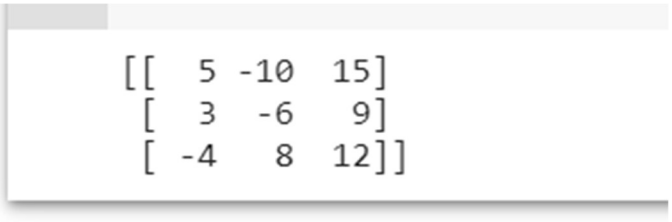
```
Cross product: -49 -7 28
```

Printing Matrix

Code:

```
import numpy as np
M1 = np.array([[5, -10, 15], [3, -6, 9], [-4, 8, 12]])
print(M1)
```

Output:



Addition 3 X 3 matrix

Code:

```
import numpy as np
M1 = np.array([[3, 6, 9], [5, -10, 15], [-7, 14, 21]])
M2 = np.array([[9, -18, 27], [11, 22, 33], [13, -26, 39]])
M3 = M1 + M2
print(M3)
```

Output:

```
[[ 12 -12  36]
 [ 16  12  48]
 [  6 -12  60]]
```

Subtraction 3 X 3 matrix

Code:

```
import numpy as np
M1 = np.array([[3, 6, 9], [5, -10, 15], [-7, 14, 21]])
M2 = np.array([[9, -18, 27], [11, 22, 33], [13, -26, 39]])
M3 = M1 - M2
print(M3)
```

Output:

```
[[ -6  24 -18]
 [ -6 -32 -18]
 [-20  40 -18]]
```

Dot Product of 3 X 3 matrix 1

Code:

```
import numpy as np
M1 = np.array([[3,6,9],[5,-10,15],[-7,14,21]])
M2 = np.array([[9,-18,27],[11,22,33],[13,-26,39]])
M3 = M1.dot(M2)
print(M3)
```

Output:

```
[[ 210 -156  630]
 [ 130 -700  390]
 [ 364 -112 1092]]
```

Dot Product of 3 X 3 matrix 2

Code:

```
import numpy as np
mat1 = ([1, 6, 5],[3 ,4, 8],[2, 12, 3])
mat2 = ([3, 4, 6],[5, 6, 7],[6, 56, 7])
res = np.dot(mat1,mat2)
print(res)
```

Output:

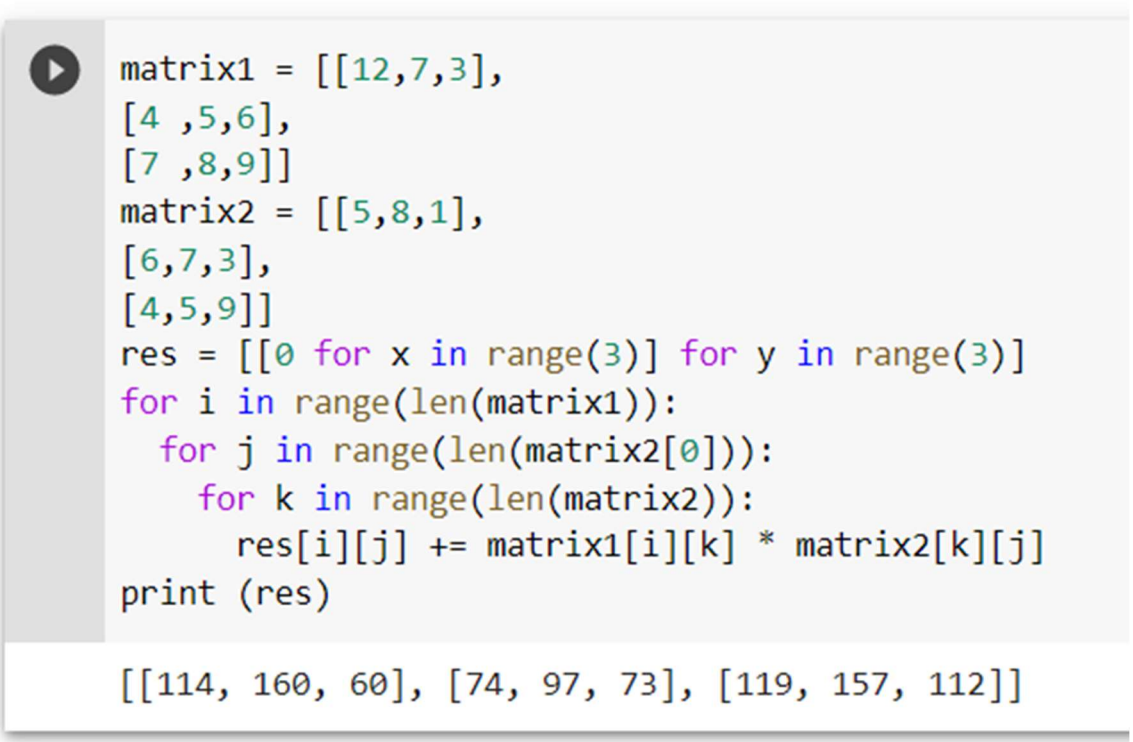
```
[[ 63 320  83]
 [ 77 484 102]
 [ 84 248 117]]
```

Matrix Identity

Code:

```
matrix1 = [[12,7,3],
[4 ,5,6],
[7 ,8,9]]
matrix2 = [[5,8,1],
[6,7,3],
[4,5,9]]
res = [[0 for x in range(3)] for y in range(3)]
for i in range(len(matrix1)):
  for j in range(len(matrix2[0])):
    for k in range(len(matrix2)):
      res[i][j] += matrix1[i][k] * matrix2[k][j]
print (res)
```

Output:



Transpose of Matrix

Code:

```
import numpy as np
M1 = np.array([[3, 6, 9], [5, -10, 15], [4,8,12]])
M2 = M1.transpose()
print("Original Matrix:")
print( M1)
print("")
print("Transpose of the Matrix:")
print(M2)
```

Output:

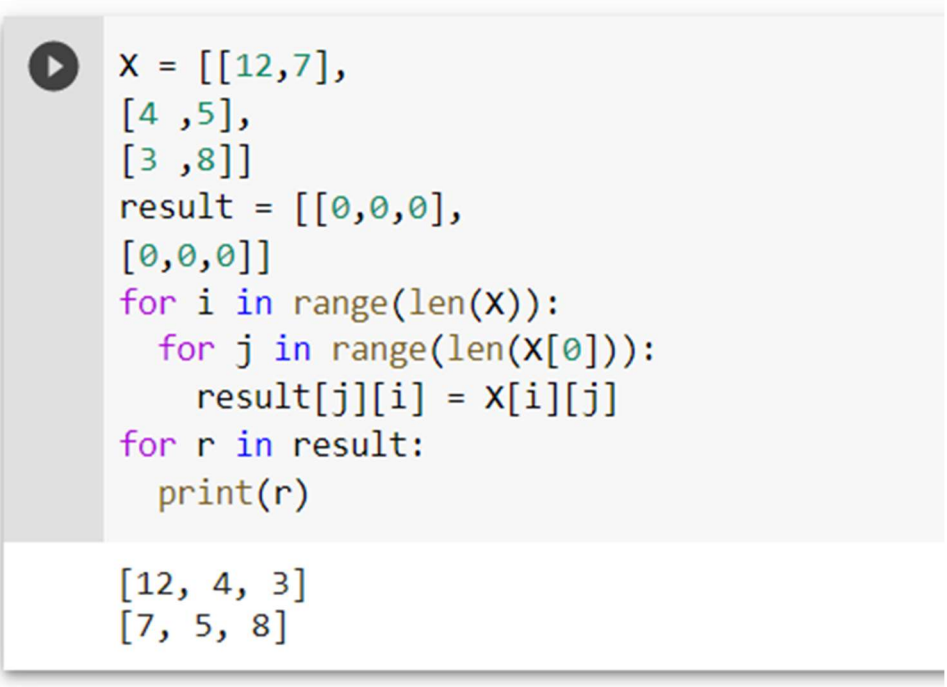
```
Original Matrix:
[[  3   6   9]
 [  5 -10  15]
 [  4   8  12]]

Transpose of the Matrix:
[[  3   5   4]
 [  6 -10   8]
 [  9  15  12]]
```

Matrix Identity 2

Code:

```
X = [[12,7],
[4 ,5],
[3 ,8]]
result = [[0,0,0],
[0,0,0]]
for i in range(len(X)):
  for j in range(len(X[0])):
    result[j][i] = X[i][j]
for r in result:
  print(r)
```

Output:



Identity Prove (AB)' = B'A'

Code:

```
import numpy as np

a = np.array([[3,3,2],[4,2,0],[4,1,2]])
b = np.array([[2,-1,2],[1,2,4],[3,6,1]])

print("a is: ")
print(a)
print(" ")

print("b is: ")
print(b)
print(" ")

dot_a_b = np.dot(a,b)
print("a.b= ")
print(dot_a_b)

tran_ab = dot_a_b.transpose()
print("")
print("LHS:")
print("Transpose of a.b is ")
print(tran_ab)

tran_b = b.transpose()
print(" ")
print(" b'  = ")
print(tran_b)

tran_a = a.transpose()
print(" ")
print(" a'  = ")
print(tran_a)

print(" ")
rhs = np.dot(tran_b,tran_a)
print("RHS: ")
print("b'.a' = ")
print(rhs)

print(" ")
```

```
print("(AB)' = B'A' ")
print("Hence proved RHS = LHS")
```

Output:

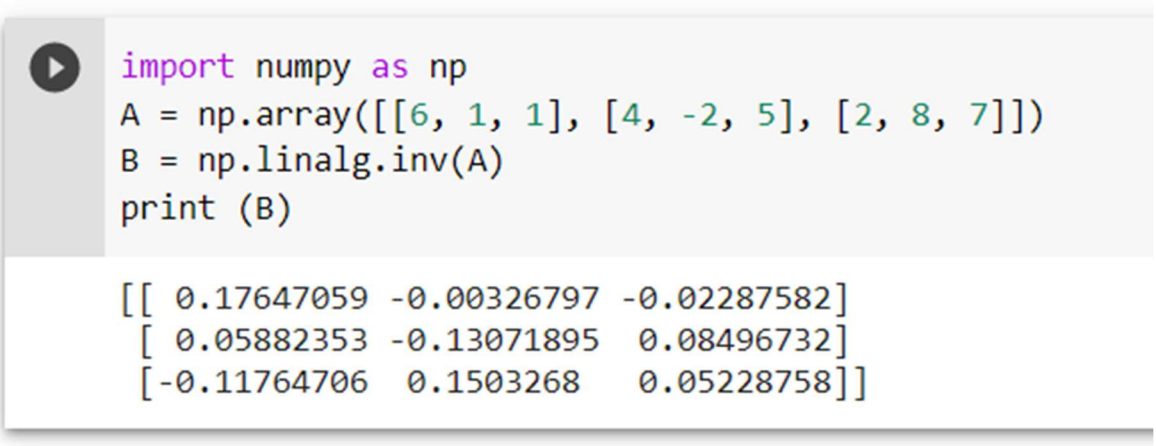
```
a is:
[[3 3 2]
 [4 2 0]
 [4 1 2]]

b is:
[[ 2 -1  2]
 [ 1  2  4]
 [ 3  6  1]]

a.b=
[[15 15 20]
 [10  0 16]
 [15 10 14]]

LHS:
Transpose of a.b is
[[15 10 15]
 [15  0 10]
 [20 16 14]]

 b'  =
[[ 2  1  3]
 [-1  2  6]
 [ 2  4  1]]

 a'  =
[[3 4 4]
 [3 2 1]
 [2 0 2]]

RHS:
b'.a' =
[[15 10 15]
 [15  0 10]
 [20 16 14]]

(AB)' = B'A'
Hence proved RHS = LHS
```

The (multiplicative) inverse of a Matrix

Code:

```
import numpy as np
A = np.array([[6, 1, 1], [4, -2, 5], [2, 8, 7]])
B = np.linalg.inv(A)
print (B)
```

Output:



Inverse of a

Code:

```
import numpy as np
a = np.array([[2,1,2,1],[6,-6,6,12],[4,3,3,-3],[2,2,-1,1]])
h = np.array([6,36,-1,10])
b = np.linalg.inv(a)
print(b)

print(" ")

x = b.dot(h)
print(x)
```

Output:

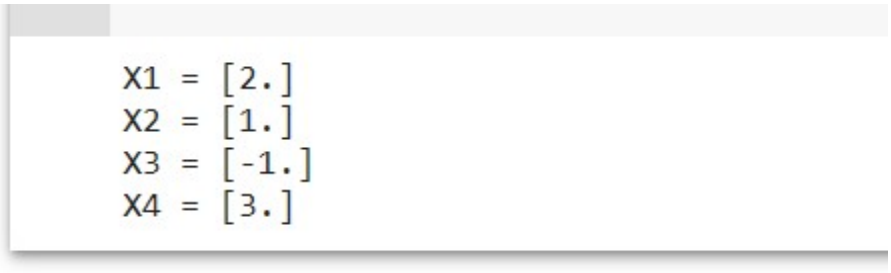
```
[[-0.69230769  0.11538462  0.30769231  0.23076923]
 [ 0.76923077 -0.12820513 -0.23076923  0.07692308]
 [ 0.58974359 -0.04273504 -0.07692308 -0.30769231]
 [ 0.43589744 -0.01709402 -0.23076923  0.07692308]]

[ 2.  1. -1.  3.]
```

Solving equations X values

Code:

```
import numpy as np

X = ['X1','X2', 'X3', 'X4']
A = np.array([[2,1,2,1],[6,-6,6,12],[4,3,3,-3],[2,2,-1,1]])
H = np.array([[6], [36], [-1], [10]])

A_inverse = np.linalg.inv(A)
D = A_inverse.dot(H)

for i in range(0,len(X)):
  print(f'{X[i]} = {D[i]}')
```

Output:



Module 3

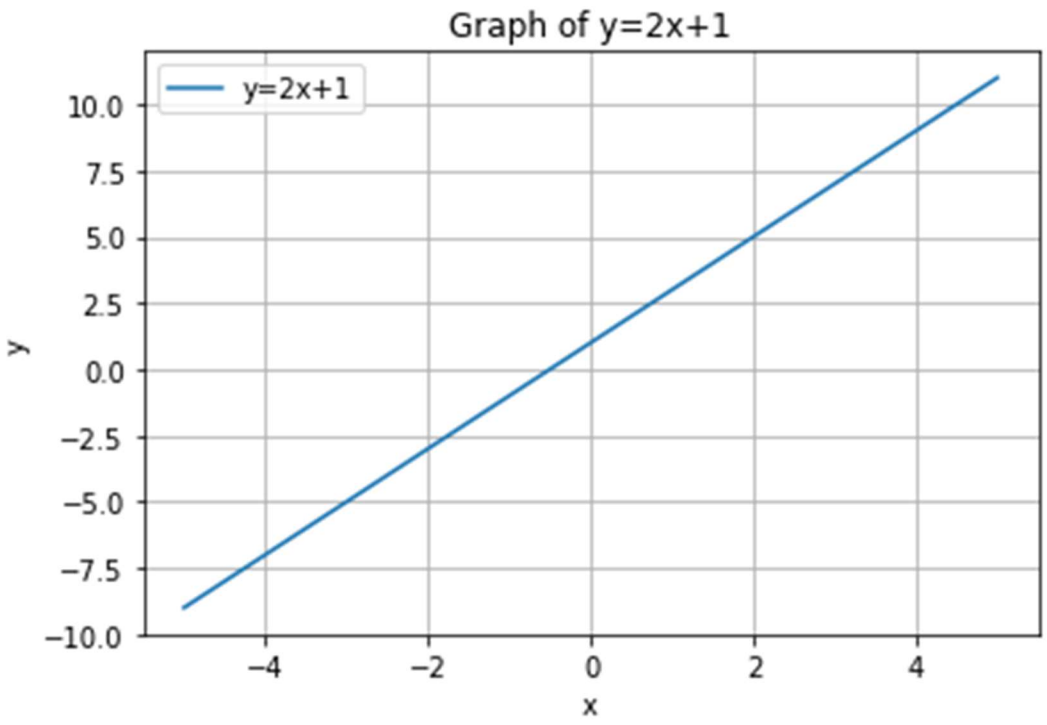
- *Graph of y=2x+1*
- *Graph for Multiple lines*
- *Solid Line Graph*
- *Simple Parabola y= x^2*
- *Graph for x^2 & x^3*
- *cos(x) plot*
- *One wave of cos(x) graph*
- *cos(x) & sin (x) plots*
- *One wave of cos(x) & sin (x) plots*
- *tan(x) plot*
- *cos^2(x) & sin^2(x) plot*
- *exp(x) plot*
- *Steeper exp(x) plot*
- *log(x) plot*
- *log(x) & log10(x)*
- *Finding the Mean of given data*
- *Finding the Mean of given data 2*
- *Finding the Median of given data*
- *Finding the Mode of given data*
- *Finding the Standard Deviation of sample*
- *Random number Decimal*
- *Random number Integer*
- *All Stats calculation*
- *All Stats calculation 2*
- *All Stats calculation with Graph*

Graph of y=2x+1

Code:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(-5,5,100)
y = 2*x+1
plt.plot(x, y, label='y=2x+1')
plt.title('Graph of y=2x+1')
plt.xlabel('x')
plt.ylabel('y')
plt.legend(loc='upper left')
plt.grid()
plt.show()
```

Output:

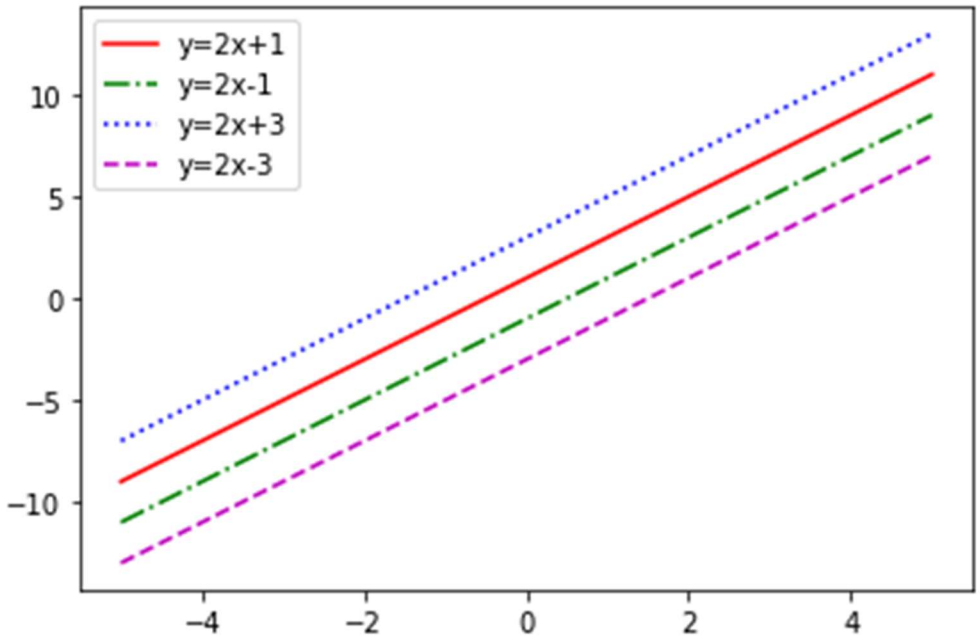


Graph for Multiple lines

Code:

```
import matplotlib.pyplot as plt
import numpy as np
fig = plt.figure()
x = np.linspace(-5,5,100)
plt.plot(x, 2*x+1, '-r', label='y=2x+1')
plt.plot(x, 2*x-1,'-.g', label='y=2x-1')
plt.plot(x, 2*x+3,':b', label='y=2x+3')
plt.plot(x, 2*x-3,'--m', label='y=2x-3')
plt.legend(loc='upper left')
plt.show()
```

Output:

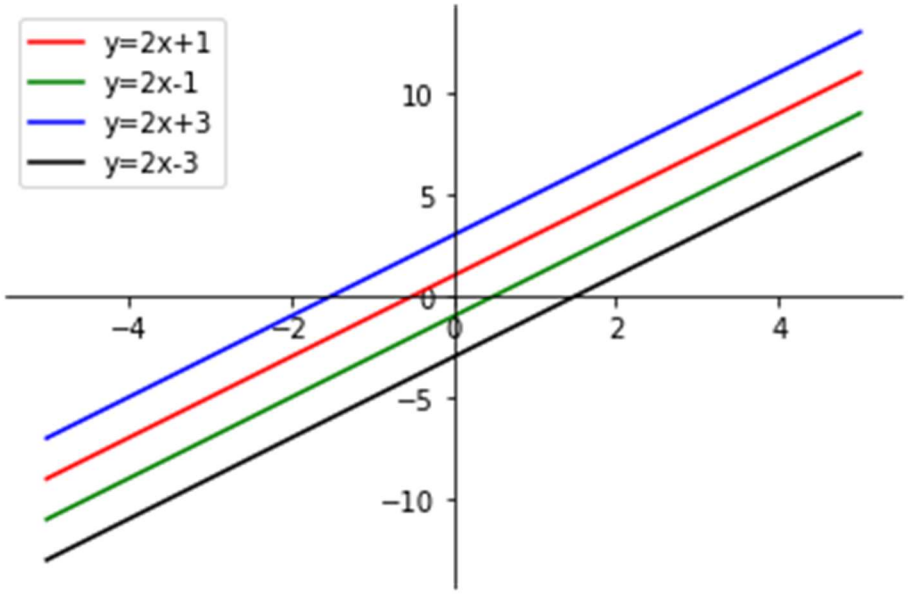


Solid Line Graph

Code:

```
import matplotlib.pyplot as plt
import numpy as np
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
x = np.linspace(-5,5,100)
ax.spines['left'].set_position('center')
ax.spines['bottom'].set_position('center')
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.xaxis.set_ticks_position('bottom')
ax.yaxis.set_ticks_position('left')
plt.plot(x, 2*x+1, 'r', label='y=2x+1')
plt.plot(x, 2*x-1,'g', label='y=2x-1')
plt.plot(x, 2*x+3,'b', label='y=2x+3')
plt.plot(x, 2*x-3,'k', label='y=2x-3')
plt.legend(loc='upper left')
plt.show()
```

Output:

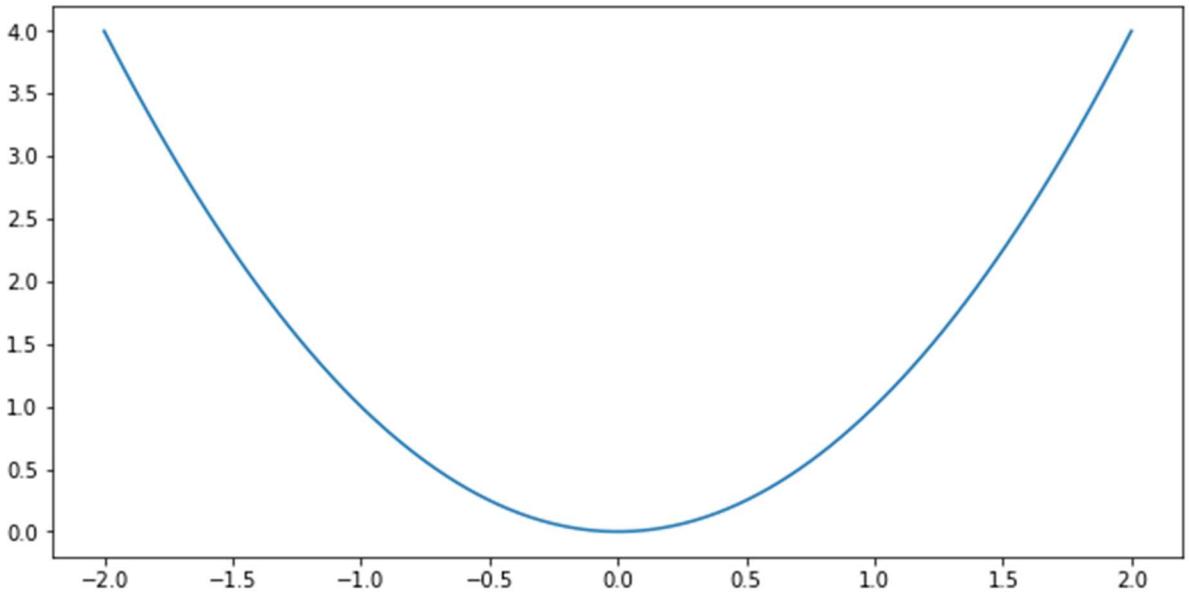


Simple Parabola y= x^2

Code:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(-2, 2, 100)
y = x ** 2
fig = plt.figure(figsize = (10, 5))
plt.plot(x, y)
plt.show()
```

Output:

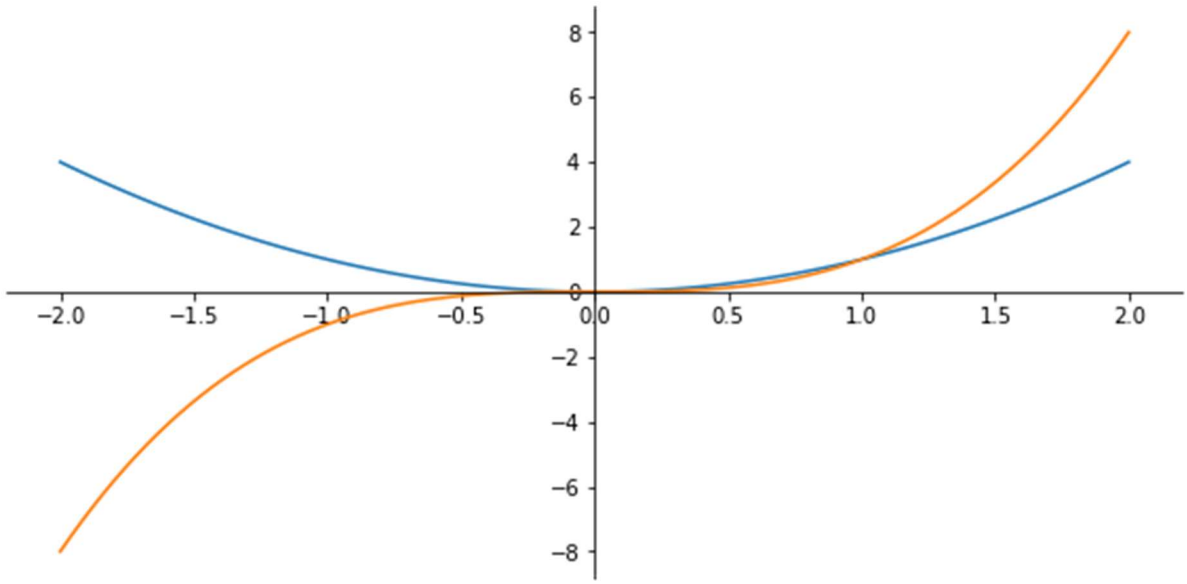


Graph for x^2 & x^3

Code:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(-2, 2, 100)
fig = plt.figure(figsize = (10, 5))
ax = fig.add_subplot(1,1,1)
ax.spines['left'].set_position('center')
ax.spines['bottom'].set_position('center')
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')

plt.plot(x,x**2)
plt.plot(x,x**3)

plt.show()
```

Output:

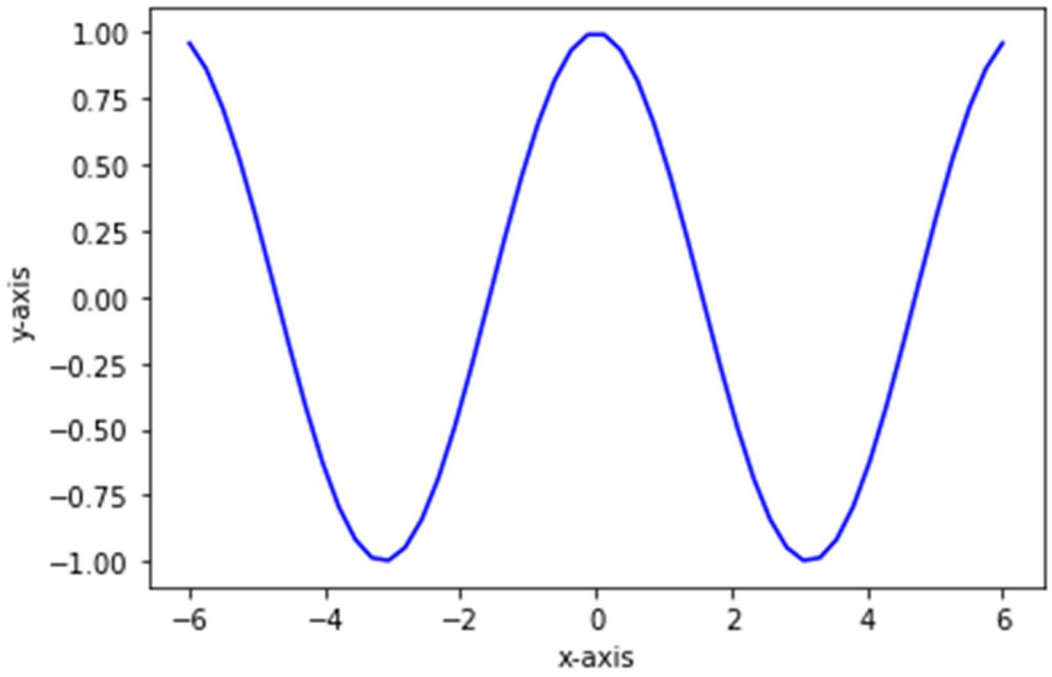


cos(x) plot

Code:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(-6, 6, 50)
y = np.cos(x)
plt.plot(x, y, 'b', label ='cos(x)')
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.show()
```

Output:

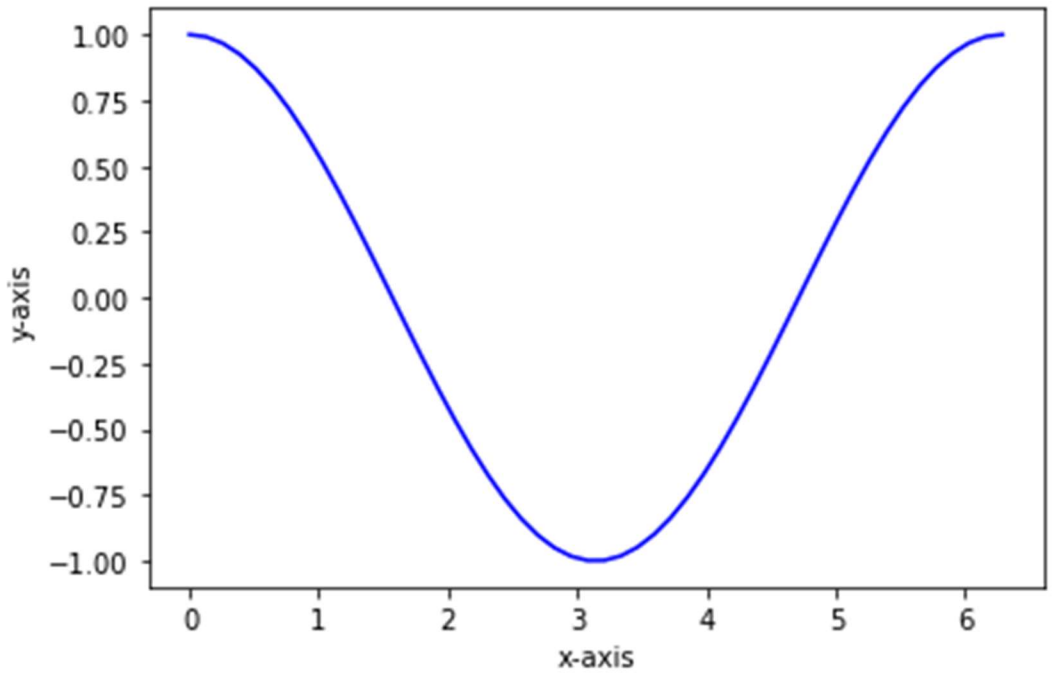


One wave of cos(x) graph

Code:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(0, 6.29, 50)
y = np.cos(x)
plt.plot(x, y, 'b', label ='cos(x)')
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.show()
```

Output:

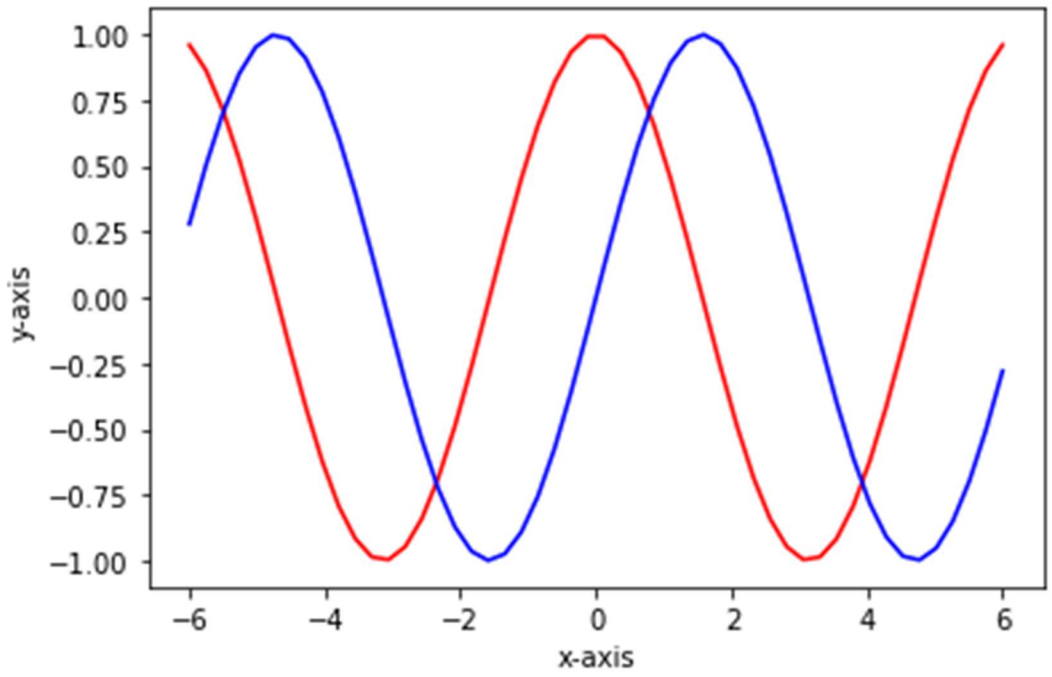


cos(x) & sin (x) plots

Code:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(-6, 6, 50)
plt.plot(x, np.cos(x),'b', label='cos(x)',color='r')
plt.plot(x, np.sin(x),'b', label='cos(x)')
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.show()
```

Output:

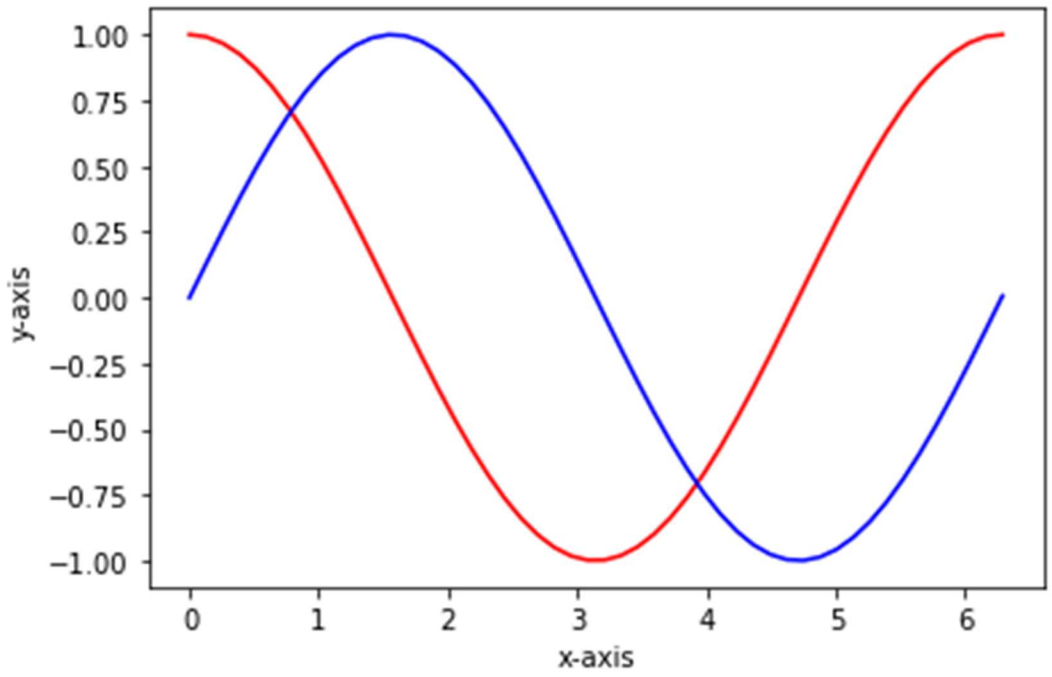


One wave of cos(x) & sin (x) plots

Code:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(0, 6.29, 50)
plt.plot(x, np.cos(x),'b', label='cos(x)',color='r')
plt.plot(x, np.sin(x),'b', label='cos(x)')
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.show()
```

Output:

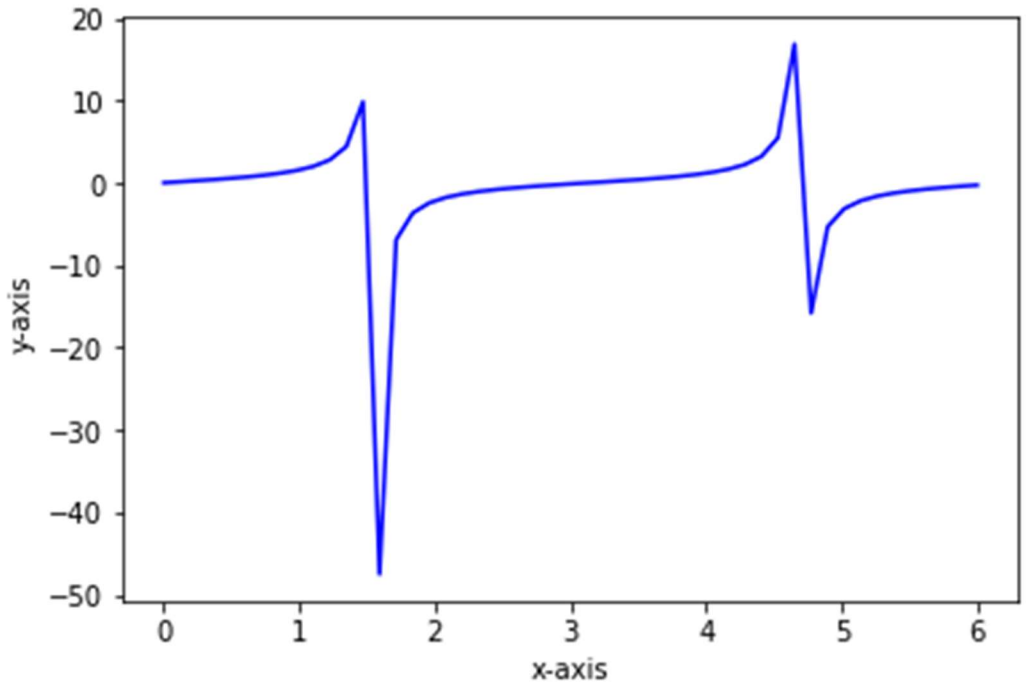


tan(x) plot

Code:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(0, 6, 50)
plt.plot(x, np.tan(x),'b', label='tan(x)')
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.show()
```

Output:

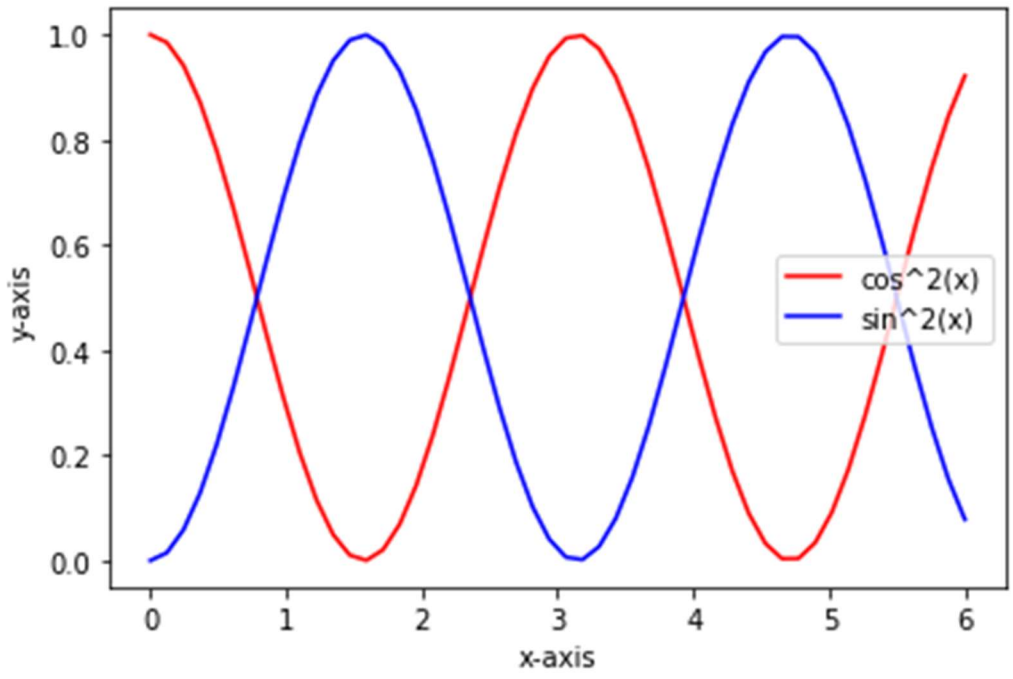


cos^2(x) & sin^2(x) plot

Code:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(0, 6, 50)
y = np.cos(x)
z = np.sin(x)
plt.plot(x, y ** 2, 'r', label='cos^2(x)')
plt.plot(x, z ** 2, 'b', label='sin^2(x)')
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.legend()
plt.show()
```

Output:

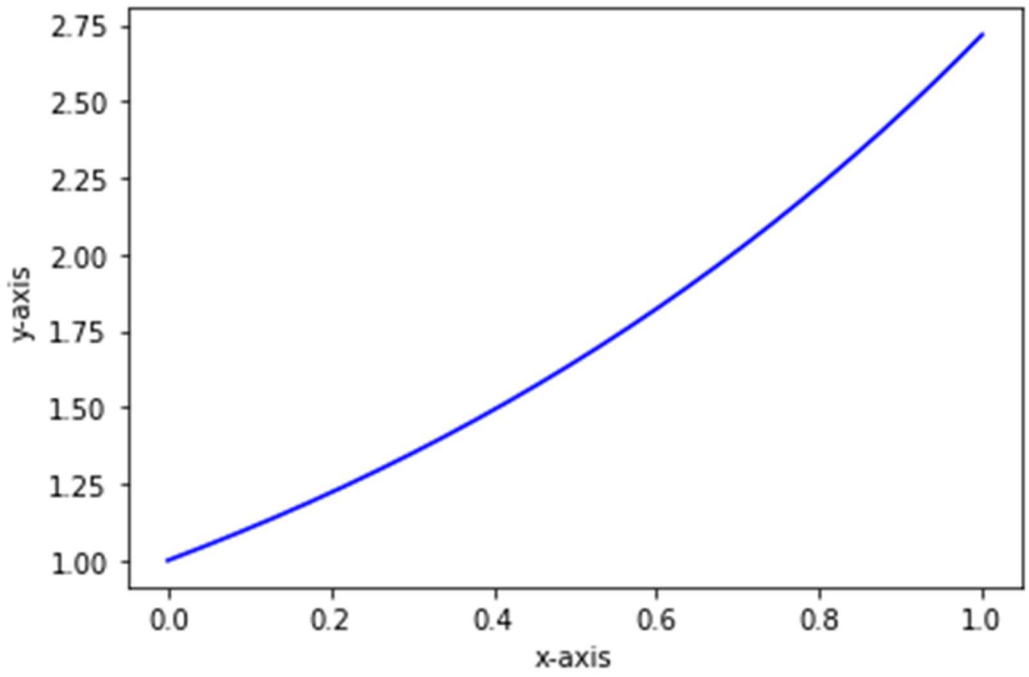


exp(x) plot

Code:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(0, 1, 1000)
y = np.exp(x)
plt.plot(x, y, 'b', label ='exp(x)')
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.show()
```

Output:

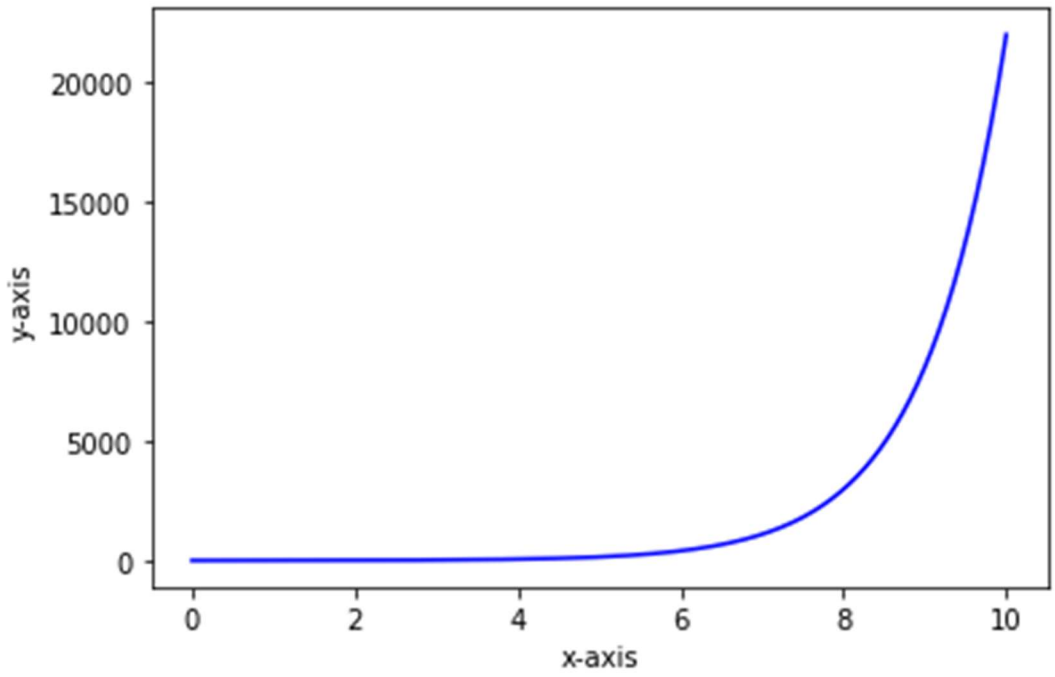


Steeper exp(x) plot

Code:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(0, 10, 1000) //only 10 is the changing part
y = np.exp(x)
plt.plot(x, y, 'b', label ='exp(x)')
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.show()
```

Output:

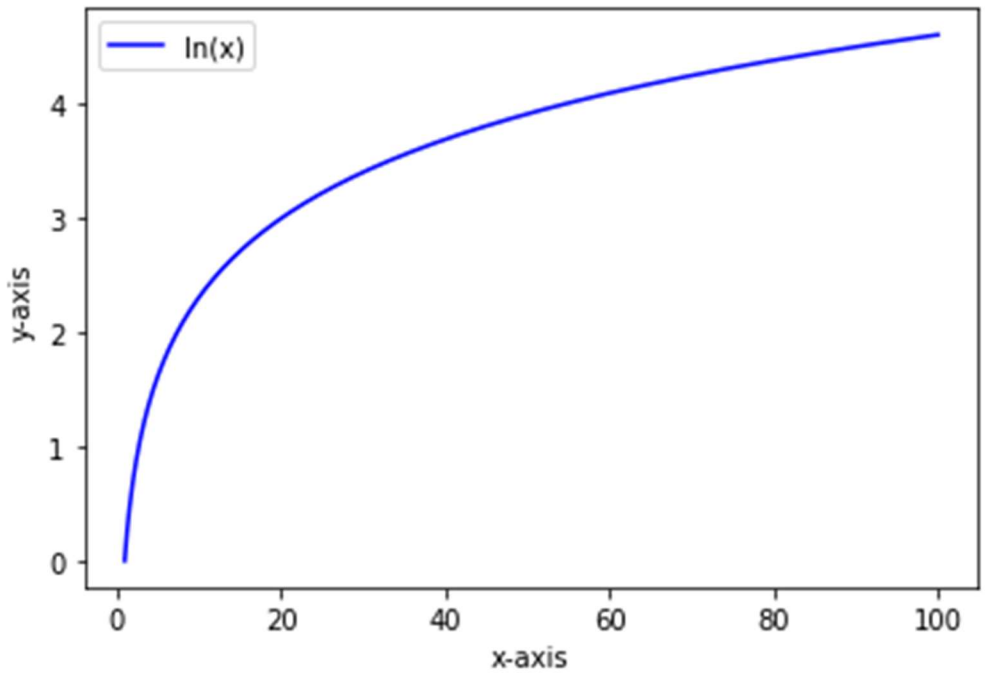


log(x) plot

Code:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(1, 100, 1000)
y = np.log(x)
plt.plot(x, y, 'b', label ='ln(x)')
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.legend()
plt.show()
```

Output:

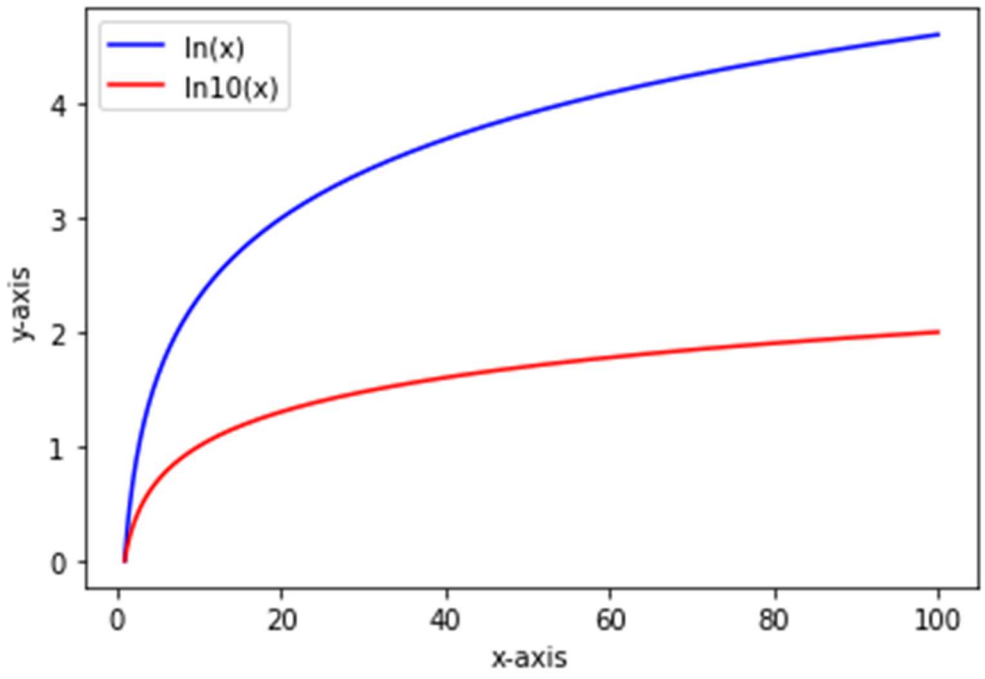


log(x) & log10(x)

Code:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(1, 100, 1000)
y = np.log(x)
z = np.log10(x)
plt.plot(x, y, 'b', label ='ln(x)')
plt.plot(x, z, 'r', label ='ln10(x)')
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.legend()
plt.show()
```

Output:

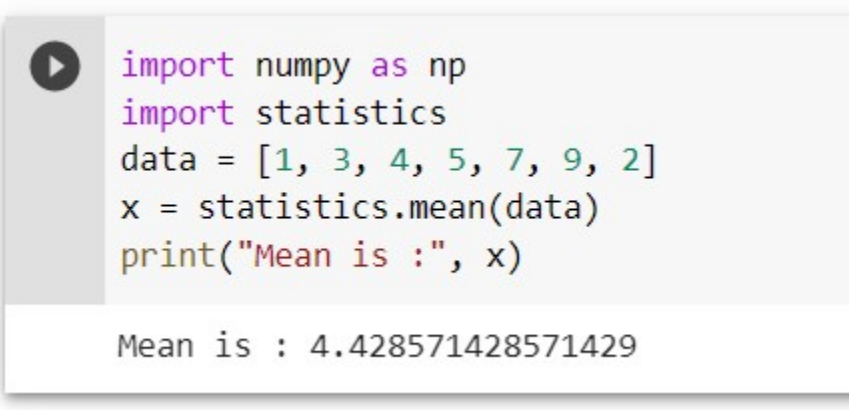


Finding the Mean of given data

Code:

```
import numpy as np
import statistics
data = [1, 3, 4, 5, 7, 9, 2]
x = statistics.mean(data)
print("Mean is :", x)
```

Output:

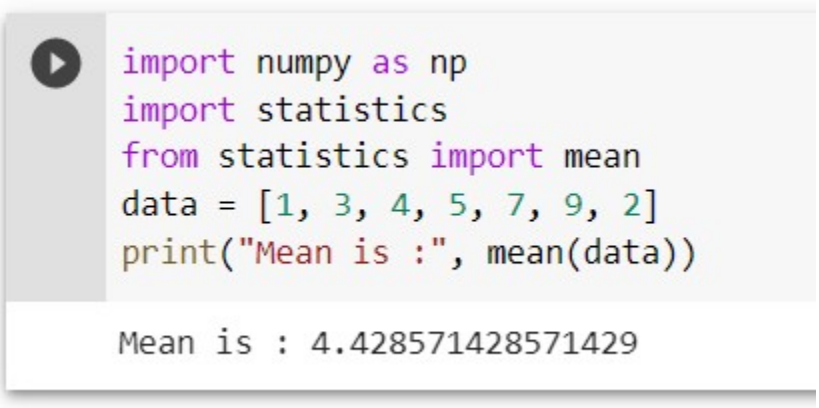


Finding the Mean of given data 2

Code:

```
import numpy as np
import statistics
from statistics import mean
data = [1, 3, 4, 5, 7, 9, 2]
print("Mean is :", mean(data))
```

Output:

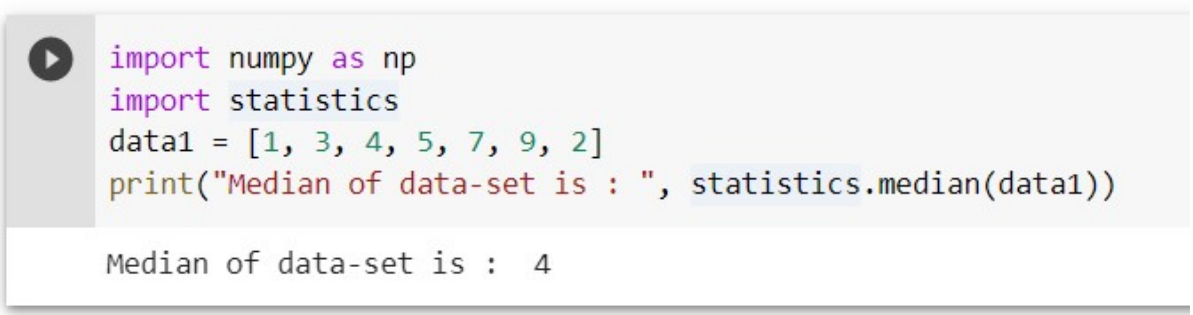


Finding the Median of given data

Code:

```
import numpy as np
import statistics
data1 = [1, 3, 4, 5, 7, 9, 2]
print("Median of data-set is : ", statistics.median(data1))
```

Output:

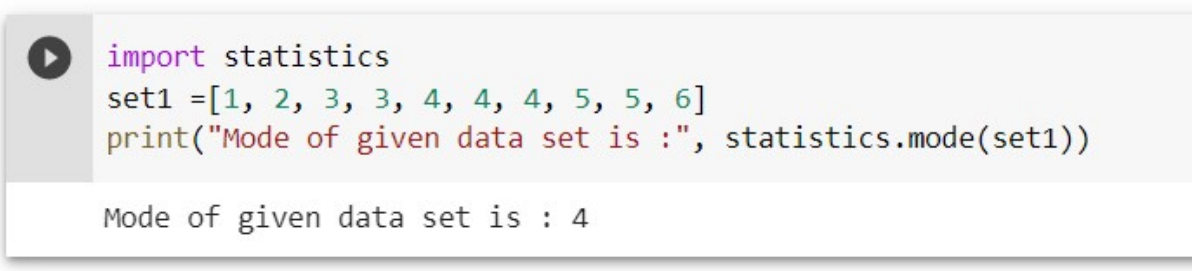


Finding the Mode of given data

Code:

```
import statistics
set1 =[1, 2, 3, 3, 4, 4, 4, 5, 5, 6]
print("Mode of given data set is :", statistics.mode(set1))
```

Output:



Finding the Standard Deviation of sample

Code:

```
import statistics
sample = [1, 2, 3, 4, 5]
print("Standard Deviation of sample is:" , (statistics.stdev(sample)))
```

Output:

Random number Decimal

Code:

```
from numpy.random import rand
for i in range(10):
  print(rand())
```

Output:

Random number Integer

Code:

```
from numpy.random import randint
for i in range(10):
  print(randint(0,10))
```

Output:

All Stats calculation

Code:

```
import statistics
from numpy.random import randint

list = []
for i in range(10):
  num = randint(0,10)
  list.append(num)
print(f"List of Random Numbers: {list}")

mean = statistics.mean(list)
median = statistics.median(list)
mode = statistics.mode(list)
standard_dev = statistics.stdev(list)

print(f"Mean of Random Numbers: {mean}")
print(f"Median of Random Numbers: {median}")
print(f"Mode of Random Numbers: {mode}")
print(f"standard Deviation of Random Numbers: {standard_dev}")
```

Output:

```
List of Random Numbers: [0, 5, 5, 1, 2, 5, 4, 3, 0, 7]
Mean of Random Numbers: 3.2
Median of Random Numbers: 3.5
Mode of Random Numbers: 5
standard Deviation of Random Numbers: 2.3944379994757297
```

All Stats calculation 2

Code:

```
import statistics
from numpy.random import randint

numbers = []
for i in range(10):
  j = randint(0,10)
  numbers.append(j)

print(f'THE RANDOM NUMBERS ARE {numbers}')

mean = statistics.mean(numbers)
print(f'THE MEAN OF THE RANDOM NUMBERS IS {mean}')

mode = statistics.mode(numbers)
print(f'THE MODE OF THE RANDOM NUMBERS IS {mode}')

median = statistics.median(numbers)
print(f'THE MODE OF THE RANDOM NUMBERS IS {median}')

standard_dev = statistics.stdev(numbers)
print(f'THE STANDARD DEVIATION OF THE RANDOM NUMBER IS {standard_dev}')
```

Output:

```
THE RANDOM NUMBERS ARE [6, 9, 3, 6, 1, 4, 5, 3, 3, 8]
THE MEAN OF THE RANDOM NUMBERS IS 4.8
THE MODE OF THE RANDOM NUMBERS IS 3
THE MODE OF THE RANDOM NUMBERS IS 4.5
THE STANDARD DEVIATION OF THE RANDOM NUMBER IS 2.485513584307633
```

All Stats calculation with Graph

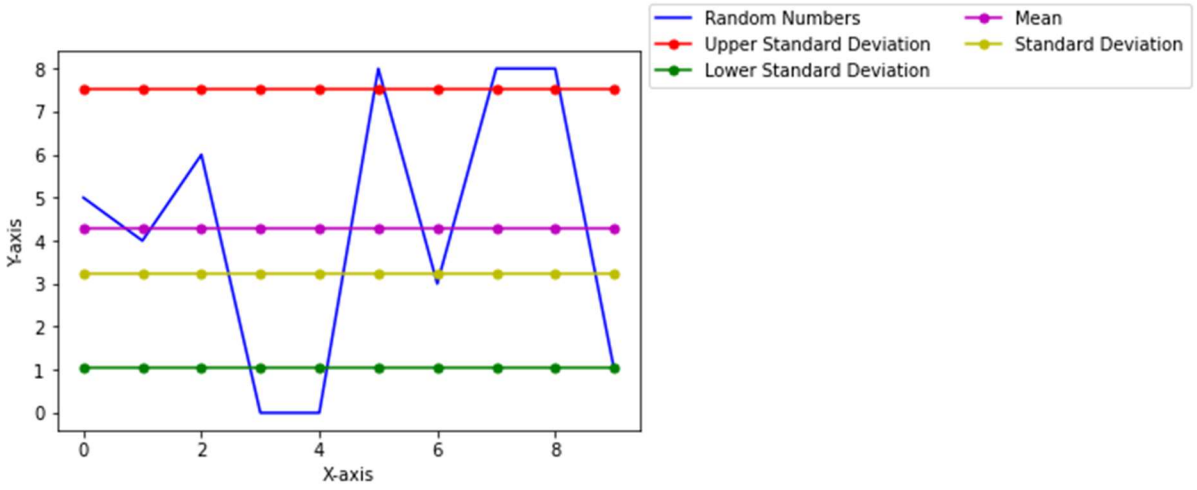
Code:

```
import statistics
import matplotlib.pyplot as plt
from numpy.random import randint

lst = []

for i in range(10):
  j = randint(0,10)
  lst.append(j)
print(f'List of Random Numbers: {lst}')

mean = statistics.mean(lst)
print(f'Mean of Random Numbers: {mean}')

median = statistics.median(lst)
print(f'Median of Random Numbers: {median}')

mode = statistics.mode(lst)
print(f'Mode of Random Numbers: {mode}')

standard_dev = statistics.stdev(lst)
print(f'Standard Deviation of Random Numbers: {standard_dev}')

up_standard_dev = mean + standard_dev
down_standard_dev = mean - standard_dev

q = []
for i in range(len(lst)):
  q.append(up_standard_dev)

r = []
for i in range(len(lst)):
  r.append(down_standard_dev)

s = []
for i in range(len(lst)):
  s.append(mean)

t = []
for i in range(len(lst)):
  t.append(standard_dev)
```

```
plt.plot(lst,'b',label='Random Numbers')
plt.plot(q,'r',label = 'Upper Standard Deviation', marker='.',markersize='
10')
plt.plot(r,'g',label = 'Lower Standard Deviation', marker='.',markersize='
10')
plt.plot(s,'m',label = 'Mean', marker='.',markersize='10')
plt.plot(t,'y',label = 'Standard Deviation', marker='.',markersize='10')

plt.xlabel('X-axis')
plt.ylabel('Y-axis')

plt.legend(bbox_to_anchor =(1.00, 1.15), ncol = 2)
plt.show()
```

Output:

```
List of Random Numbers: [5, 4, 6, 0, 0, 8, 3, 8, 8, 1]
Mean of Random Numbers: 4.3
Median of Random Numbers: 4.5
Mode of Random Numbers: 8
Standard Deviation of Random Numbers: 3.2335051500740732
```



Module 4

- Freely Falling Body
- Collision of two particles at head on collision
- Collision of two particles at deflection at an angle
- Projectile motion using U_X and U_Y
- Projectile motion using equation of trajectory
- Projectile motion in 3D

Freely Falling Body

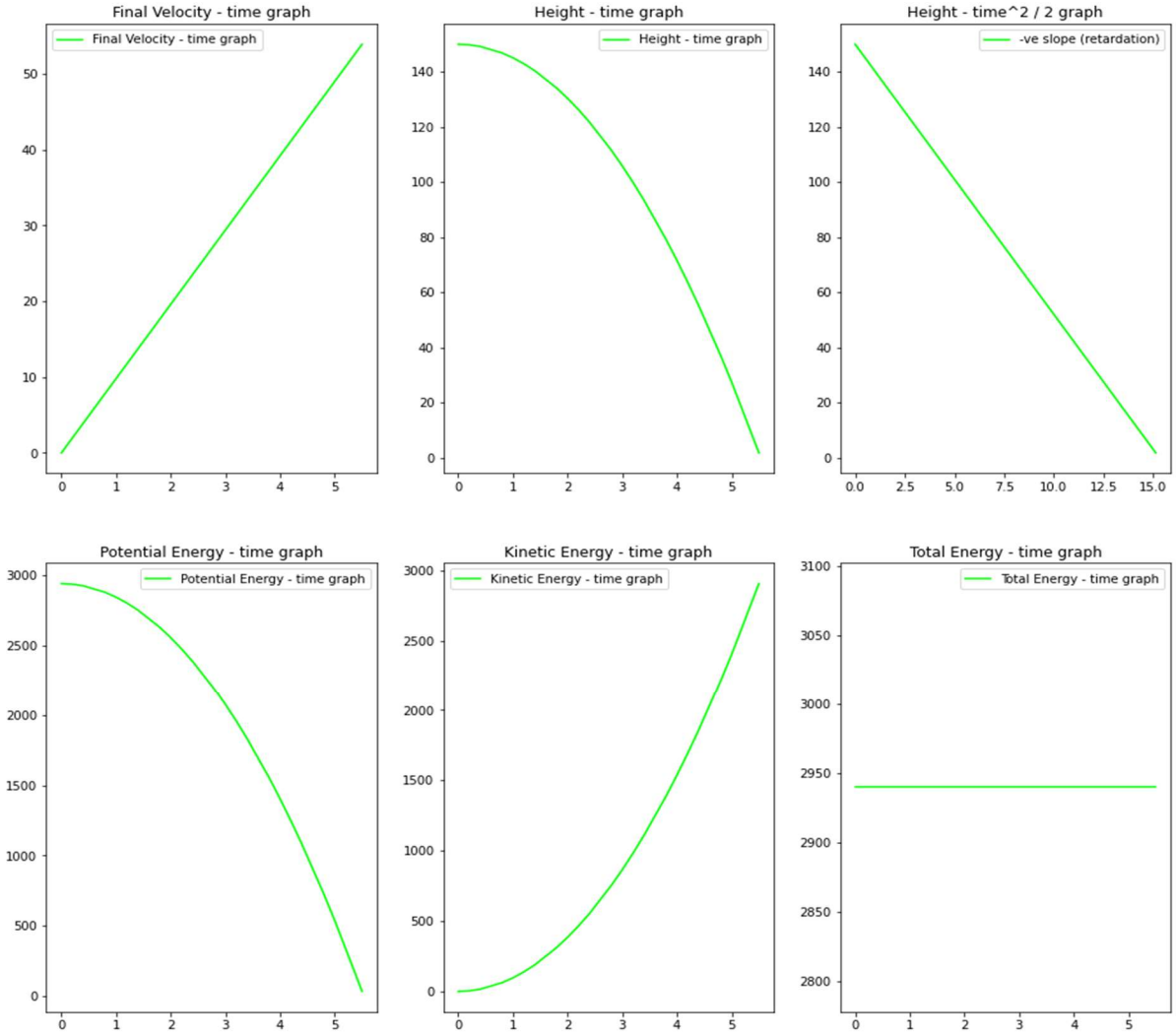
Code:

```
import matplotlib.pyplot as plt
import math

mass_of_the_particle = 2
initial_velocity_of_the_particle = 0
acceleration_due_to_gravity = 9.8
initial_height_of_the_particle = 150
initial_potential_energy_of_the_particle = (mass_of_the_particle * acceler
ation_due_to_gravity *
                                            initial_height_of_the_particle
)
initial_kinetic_energy_of_the_particle = 0
initial_total_energy_of_the_particle = initial_potential_energy_of_the_par
ticle + initial_kinetic_energy_of_the_particle

time_to_reach_the_particle_to_hit_the_ground = math.sqrt((2 * initial_heig
ht_of_the_particle)
                                                        / acceleration_du
e_to_gravity)

time_to_reach_the_particle_to_hit_the_ground = round(time_to_reach_the_par
ticle_to_hit_the_ground, 1)
times_in_between_the_fall = []
for i in range(0, int(time_to_reach_the_particle_to_hit_the_ground) + 1):
    times_in_between_the_fall.append(i)
times = []
for i in range(0, len(times_in_between_the_fall)):
    X = times_in_between_the_fall[i]
    if i != len(times_in_between_the_fall) - 1:
        if X != times_in_between_the_fall[i + 1]:
            a = X + 0.2
            b = X + 0.4
            c = X + 0.6
            d = X + 0.8
            times.extend([a, b, d, d])
    else:
        break

for i in range(0, len(times_in_between_the_fall)):
    times_in_between_the_fall[i] = float(times_in_between_the_fall[i])
times_in_between_the_fall.extend(times)
```

```
times_in_between_the_fall.sort()
times_in_between_the_fall.append(time_to_reach_the_particle_to_hit_the_gro
und)

final_velocity_in_between_the_fall = []
height_of_the_particle_in_between_the_fall = []
height_of_the_particle_in_between_the_fall_with_change_in_time = []  # EXT
RA COMPLETED
potential_energy_in_between_the_fall = []
kinetic_energy_in_between_the_fall = []
total_energy_in_between_the_fail = []

for i in range(len(times_in_between_the_fall)):
    a = initial_velocity_of_the_particle + (acceleration_due_to_gravity *
times_in_between_the_fall[i])
    b = initial_height_of_the_particle - (0.5 * acceleration_due_to_gravit
y * (times_in_between_the_fall[i] ** 2))
    c = b
    d = (mass_of_the_particle * acceleration_due_to_gravity * b)
    e = (0.5 * mass_of_the_particle * (a ** 2))
    f = d + e
    final_velocity_in_between_the_fall.append(a)
    height_of_the_particle_in_between_the_fall.append(b)
    height_of_the_particle_in_between_the_fall_with_change_in_time.append(
c)
    potential_energy_in_between_the_fall.append(d)
    kinetic_energy_in_between_the_fall.append(e)
    total_energy_in_between_the_fail.append(f)

fig, ax = plt.subplots(2, 3, figsize=(16, 16))
for i in range(0, 2):
    for j in range(0, 3):
        if i == 0 and j == 0:
            X = final_velocity_in_between_the_fall
            l = 'Final Velocity - time graph'
        elif i == 0 and j == 1:
            X = height_of_the_particle_in_between_the_fall
            l = 'Height - time graph'
        elif i == 0 and j == 2:
            X = height_of_the_particle_in_between_the_fall_with_change_in_
time
            l = 'Height - time^2 / 2 graph'
        elif i == 1 and j == 0:
            X = potential_energy_in_between_the_fall
            l = 'Potential Energy - time graph'
```

```
        elif i == 1 and j == 1:
            X = kinetic_energy_in_between_the_fall
            l = 'Kinetic Energy - time graph'
        else:
            X = total_energy_in_between_the_fail
            l = 'Total Energy - time graph'
        if i == 0 and j == 2:
            t = []
            for a in range(0, len(times_in_between_the_fall)):
                b = ((times_in_between_the_fall[a] ** 2) / 2)
                t.append(b)
            ax[i, j].plot(t, X, label='-
ve slope (retardation)', color='lime')  # this graph gives the -
ve slope of
            # the plot which acceleration
            ax[i, j].set_title(l)
        else:
            ax[i, j].plot(times_in_between_the_fall, X, label=l, color='li
me')
            ax[i, j].set_title(l)
        ax[i, j].legend()
plt.show()

fig, ax = plt.subplots(2, 2, figsize=(16, 16))
for i in range(0, 2):
    for j in range(0, 2):
        if i == 0 and j == 0:
            X = final_velocity_in_between_the_fall
            l = 'Final Velocity - Height graph'
        elif i == 0 and j == 1:
            X = potential_energy_in_between_the_fall
            l = 'Potential Energy - Height graph'
        elif i == 1 and j == 0:
            X = kinetic_energy_in_between_the_fall
            l = 'Kinetic Energy - Height graph'
        else:
            X = total_energy_in_between_the_fail
            l = 'Total Energy - Height graph'
        ax[i, j].plot(height_of_the_particle_in_between_the_fall, X, label
=l, color='lime')
        ax[i, j].set_title(l)
        ax[i, j].legend()
plt.show()
```

Output:


Final Velocity - time graph
Final Velocity - time graph
Height - time graph
Height - time graph
Height - time^2 / 2 graph
-ve slope (retardation)
Potential Energy - time graph
Potential Energy - time graph
Kinetic Energy - time graph
Kinetic Energy - time graph
Total Energy - time graph
Total Energy - time graph

Collision of two particles at head on collision

Code:

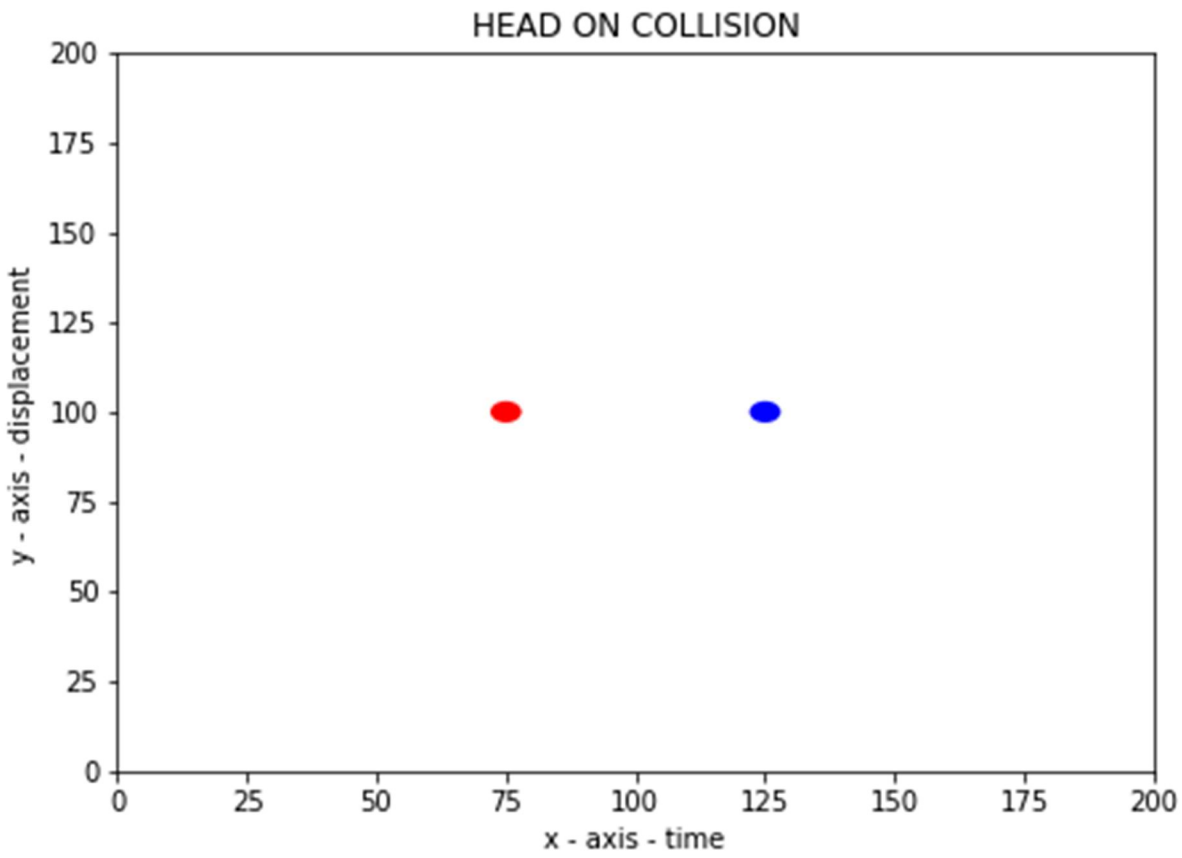
```
import matplotlib.pyplot as plt
import matplotlib.animation as animation
m1 = float(input('ENTER THE MASS OF FIRST PARTICLE-> '))
m2 = float(input('ENTER THE MASS OF SECOND PARTICLE-> '))
u1 = float(input('ENTER THE INITIAL VELOCITY OF FIRST PARTICLE-> '))
u2 = float(input('ENTER THE INITIAL VELOCITY OF SECOND PARTICLE-> '))
v1 = (((m1 - m2) / (m1 + m2)) * u1) + (((2 * m2) / (m1 + m1)) * u2)
v2 = (((2 * m1) / (m1 + m2)) * u1) + (((m2 - m1) / (m1 + m2)) * u2)
print(f'\nTHE FINAL VELOCITY OF THE FIRST PARTICLE IS {v1}\nTHE FINAL VELO
CITY OF THE SECOND PARTICLE IS {v2}')
fig = plt.figure()
fig.set_size_inches(7, 5)
ax = plt.axes(xlim=(0, 200), ylim=(0, 200))
Circle_1, Circle_2 = plt.Circle((75, 100), 3, fc='red'), plt.Circle((125,
100), 3, fc='blue')


def init():
    Circle_1.center, Circle_2.center = (75, 100), (125, 100)
    ax.add_patch(Circle_1), ax.add_patch(Circle_2)
    return Circle_1, Circle_2,


def animation_of_particles(i):
    (x1, y1), (x2, y2) = Circle_1.center, Circle_2.center
    y1, y2 = 100, 100
    if u1 == 0:
        value = 30
        if m1 - m2 > 100:
            temp_11, temp_12, temp_21, temp_22 = 0, 1.5, 0.1, 1.25
        elif m2 - m1 > 100:
            temp_11, temp_12, temp_21, temp_22 = 0, 1.5, 2.5, -1
        elif m1 == m2:
            temp_11, temp_12, temp_21, temp_22 = 0, 1.5, 1.5, 0
        elif m1 > m2:
            temp_11, temp_12, temp_21, temp_22 = 0, 1.5, 0.5, 1
        else:
            temp_11, temp_12, temp_21, temp_22 = 0, 1.5, 1, 0.5
    elif u2 == 0:
        value = 30
        if m1 - m2 > 100:
```

```
            temp_11, temp_12, temp_21, temp_22 = 1.5, 0, -1, 2.5
        elif m2 - m1 > 100:
            temp_11, temp_12, temp_21, temp_22 = 1.5, 0, 1.25, 0.1
        elif m1 == m2:
            temp_11, temp_12, temp_21, temp_22 = 1.5, 0, 0, 1.5
        elif m1 > m2:
            temp_11, temp_12, temp_21, temp_22 = 1.5, 0, 1, 0.5
        else:
            temp_11, temp_12, temp_21, temp_22 = 1.5, 0, 0.5, 1
    elif u1 == u2:
        value = 15
        if m1 - m2 > 100:
            temp_11, temp_12, temp_21, temp_22 = 1.5, 1.5, -1, 2.5
        elif m2 - m1 > 100:
            temp_11, temp_12, temp_21, temp_22 = 1.5, 1.5, 2.5, -1
        elif m1 == m2:
            temp_11, temp_12, temp_21, temp_22 = 1.5, 1.5, 1.5, 1.5
        elif m1 > m2:
            temp_11, temp_12, temp_21, temp_22 = 1.5, 1.5, 0.75, 1.5
        else:
            temp_11, temp_12, temp_21, temp_22 = 1.5, 1.5, 1.5, 0.75
    elif u1 > u2:
        value = 15
        if m1 - m2 > 100:
            temp_11, temp_12, temp_21, temp_22 = 1.75, 1.5, -0.75, 1.5
        elif m2 - m1 > 100:
            temp_11, temp_12, temp_21, temp_22 = 1.75, 1.5, 1.5, -0.75
        elif m1 == m2:
            temp_11, temp_12, temp_21, temp_22 = 1.75, 1.5, 1.5, 1.75
        elif m1 > m2:
            temp_11, temp_12, temp_21, temp_22 = 1.75, 1.5, 0.75, 1.5
        else:
            temp_11, temp_12, temp_21, temp_22 = 1.75, 1.5, 1.5, 0.75
    else:
        value = 15
        if m1 - m2 > 100:
            temp_11, temp_12, temp_21, temp_22 = 1.5, 1.75, -0.75, 1.5
        elif m2 - m1 > 100:
            temp_11, temp_12, temp_21, temp_22 = 1.5, 1.75, 1.5, -0.75
        elif m1 == m2:
            temp_11, temp_12, temp_21, temp_22 = 1.5, 1.75, 1.5, 0.75
        elif m1 > m2:
            temp_11, temp_12, temp_21, temp_22 = 1.5, 1.75, 0.75, 1.5
        else:
            temp_11, temp_12, temp_21, temp_22 = 1.5, 1.75, 1.25, 0.75
```

```
    if i <= value:
        x1 = x1 + temp_11
        x2 = x2 - temp_12
    else:
        x1 = x1 - temp_21
        x2 = x2 + temp_22
    if x1 <= 1 or x2 >= 199:
        exit()
    else:
        Circle_1.center, Circle_2.center = (x1, y1), (x2, y2)
        return Circle_1, Circle_2,


anime = animation.FuncAnimation(fig, animation_of_particles, init_func=ini
t, interval=30)
plt.xlabel('x - axis - time'), plt.ylabel('y - axis - displacement'), plt.
title('HEAD ON COLLISION')
plt.show()
```

Output:

```
ENTER THE MASS OF FIRST PARTICLE-> 2
ENTER THE MASS OF SECOND PARTICLE-> 3
ENTER THE INITIAL VELOCITY OF FIRST PARTICLE-> 4
ENTER THE INITIAL VELOCITY OF SECOND PARTICLE-> 6

THE FINAL VELOCITY OF THE FIRST PARTICLE IS 8.2
THE FINAL VELOCITY OF THE SECOND PARTICLE IS 4.4
```


HEAD ON COLLISION
200
175
150
125
100
75
50
25
0
y - axis - displacement
0
25
50
75
100
125
150
175
200
x - axis - time

Collision of two particles at deflection at an angle

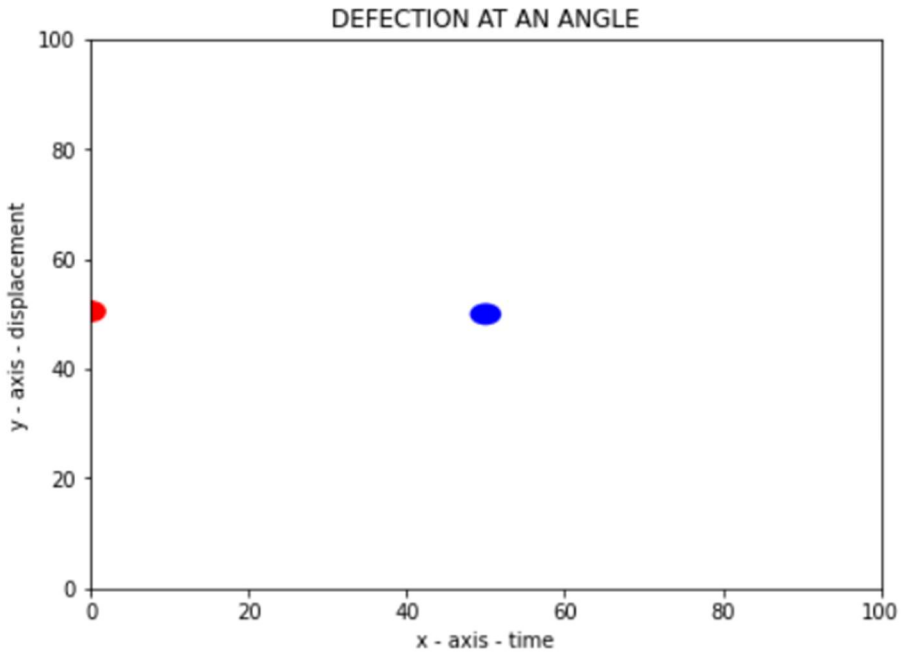
Code:

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation

Angle = float(input('ENTER THE DEFLECTION AT AN ANGLE-> '))
THETA_1, THETA_2 = (Angle * (22 / 7 / 180)), ((90 - Angle) * (22 / 7 / 180
))
fig = plt.figure()
fig.set_size_inches(7, 5)
ax = plt.axes(xlim=(0, 100), ylim=(0, 100))
Circle_1, Circle_2 = plt.Circle((0, 50.5), 2, fc='red'), plt.Circle((50, 5
0), 2, fc='blue')


def init():
    Circle_1.center, Circle_2.center = (0, 50.5), (50, 50)
    ax.add_patch(Circle_1), ax.add_patch(Circle_2)
    return Circle_1, Circle_2,


def animation_of_particles(i):
    (x1, y1), (x2, y2) = Circle_1.center, Circle_2.center
    if i > 23:
        x1 = x1 + np.cos(THETA_1)
        y1 = y1 + np.cos(THETA_1)
        x2 = x2 + np.cos(THETA_2)
        y2 = y2 - np.cos(THETA_2)
        if x1 >= 99 or x2 >= 99:
            exit()
        else:
            Circle_1.center, Circle_2.center = (x1, y1), (x2, y2)
            return Circle_1, Circle_2,
    else:
        x1 = x1 + 2
        y1 = y1
        x2 = x2
        y2 = y2
        Circle_1.center, Circle_2.center = (x1, y1), (x2, y2)
        return Circle_1, Circle_2,
```

```
anime = animation.FuncAnimation(fig, animation_of_particles, init_func=ini
t, interval=50)
plt.xlabel('x - axis - time'), plt.ylabel('y - axis - displacement'), plt.
title('DEFECTION AT AN ANGLE')
plt.show()
```

Output:

```
ENTER THE DEFLECTION AT AN ANGLE-> 30
```

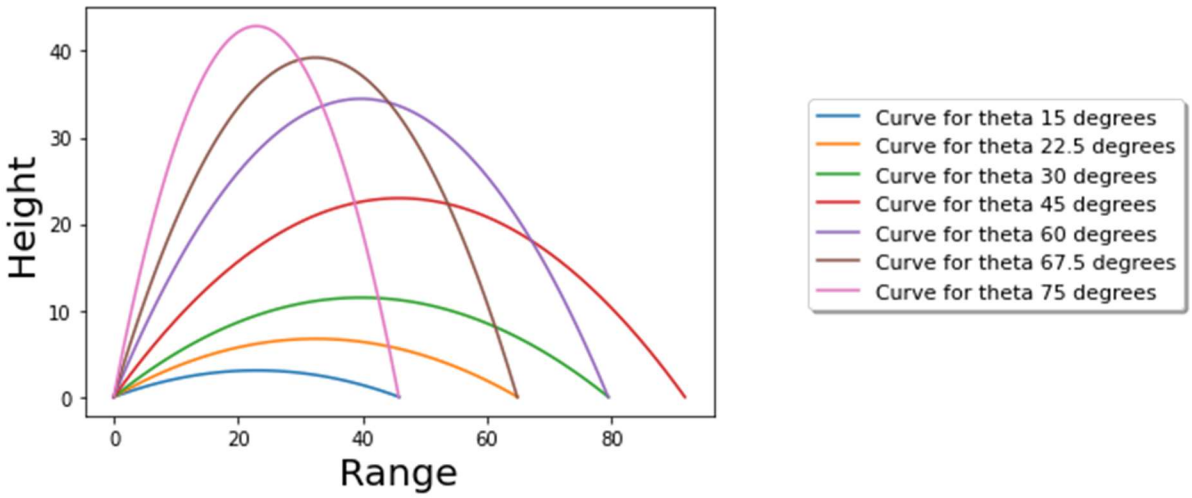


Projectile motion using U_X and U_Y

Code:

```
import matplotlib.pyplot as plt
import numpy as np

u, g = 30, 9.8
THETA = [15,22.5,30,45,60,67.5,75]
THETA_VALUES_IN_RADIANS = [(THETA[i] * ((22 / 7) / 180)) for i in range(0,
 len(THETA))]
RANGE_OF_ALL_CURVES = [(((u ** 2) * np.sin(2 * i)) / g) for i in THETA_VAL
UES_IN_RADIANS]
for i in range (0, len(THETA_VALUES_IN_RADIANS)):
  Range = np.linspace(0, RANGE_OF_ALL_CURVES[i], 100)
  X, Y = [], []
  for x in Range:
    A = np.tan(THETA_VALUES_IN_RADIANS[i])
    B = (g / (2 * (u ** 2) * (np.cos(THETA_VALUES_IN_RADIANS[i]) ** 2)))
    j = ((A * x) - (B * (x ** 2)))
    X.append(x)
    Y.append(j)
  plt.plot(X, Y, label = f'Curve for theta {THETA[i]} degrees')

plt.xlabel('Range', fontsize=20)
plt.ylabel('Height', fontsize=20)
plt.legend(loc='upper center', bbox_to_anchor=(1.45, 0.8), shadow=True, nc
ol=1,fontsize=11)

plt.show()
```

Output:

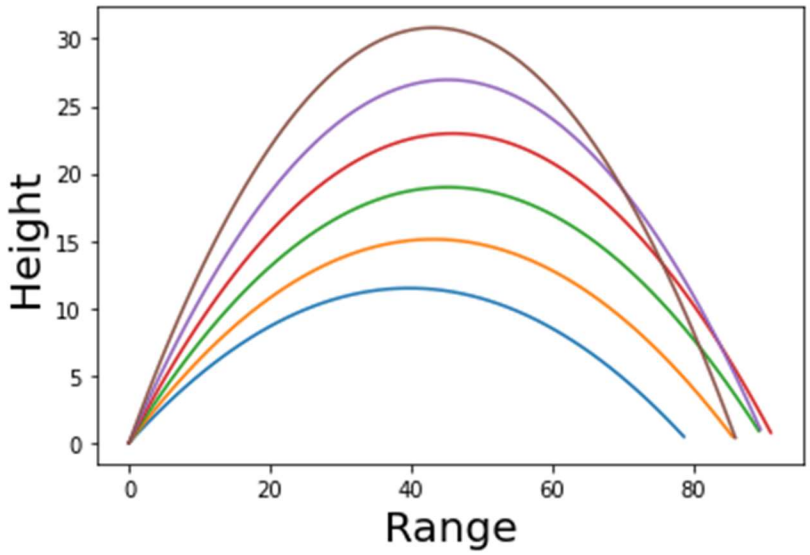

Height
Range
0
10
20
30
40
0
20
40
60
80
Curve for theta 15 degrees
Curve for theta 22.5 degrees
Curve for theta 30 degrees
Curve for theta 45 degrees
Curve for theta 60 degrees
Curve for theta 67.5 degrees
Curve for theta 75 degrees

Projectile motion using equation of trajectory

Code:

```
import numpy as np
import matplotlib.pylab as plot
import math as m

v = 30
g = 9.8

theta = np.arange(m.pi/6, m.pi/3, m.pi/36)
t = np.linspace(0, 5, num=100)
for i in theta:
    x1 = []
    y1 = []
    for k in t:
        x = ((v*k)*np.cos(i))
        y = ((v*k)*np.sin(i))-((0.5*g)*(k**2))
        x1.append(x)
        y1.append(y)
    p = [i for i, j in enumerate(y1) if j < 0]
    for i in sorted(p, reverse = True):
        del x1[i]
        del y1[i]
    plot.plot(x1, y1)
    plot.xlabel('Range', fontsize=20)
    plot.ylabel('Height', fontsize=20)
plot.show()
```

Output:



Projectile motion in 3D

Code:

```
import matplotlib.pyplot as plt
import numpy as np

u, g = 30, 9.8
THETA = [30, 45, 60]
THETA_VALUES_IN_RADIANS = [(THETA[i] * ((22 / 7) / 180)) for i in range(0,
 len(THETA))]
RANGE_OF_ALL_CURVES = [(((u ** 2) * np.sin(2 * i)) / g) for i in THETA_VAL
UES_IN_RADIANS]

ax = plt.axes(projection="3d")

for i in range(0, len(THETA_VALUES_IN_RADIANS)):
    Range = np.linspace(0, RANGE_OF_ALL_CURVES[i], 100)
    x_points = []
    y_points = []
    z_points = []
    for x in Range:
        A = np.tan(THETA_VALUES_IN_RADIANS[i])
        B = (g / (2 * (u ** 2) * (np.cos(THETA_VALUES_IN_RADIANS[i]) ** 2)
))
        j = ((A * x) - (B * (x ** 2)))
        x_points.append(x)
        y_points.append(x)
        z_points.append(j)
```

```
    plt.plot(x_points, y_points, z_points, label=f'Curve for theta {THETA[
i]} degrees')
    plt.legend(loc='upper center', bbox_to_anchor=(1.45, 0.8), shadow=True
, ncol=1,fontsize=11)

ax.set_xlabel('x - axis', fontsize=20)
ax.set_ylabel('y - axis', fontsize=20)
ax.set_zlabel('z - axis', fontsize=20)
ax.set_title('3D PLOT FOR PROJECTILE MOTION', fontsize=20)
plt.show()
```

Output: