# Lab Manual MAT 551


## Name: Arnav Sinha

## Register Number: 2040811

## Class: 5 EMS


## Matrix Theory

Matrix, a set of numbers arranged in rows and columns so as to form a rectangular array. The numbers are called the elements, or entries, of the matrix. Historically, it was not the matrix but a certain number associated with a square array of numbers called the determinant that was first recognized. Only gradually did the idea of the matrix as an algebraic entity emerge. The term matrix was introduced by the 19th-century English mathematician James Sylvester, but it was his friend the mathematician Arthur Cayley who developed the algebraic aspect of matrices in two papers in the 1850s. Cayley first applied them to the study of systems of linear equations, where they are still very useful. They are also important because, as Cayley recognized, certain sets of matrices form algebraic systems in which many of the ordinary laws of arithmetic (e.g., the associative and distributive laws) are valid but in which other laws (e.g., the commutative law) are not valid.

## Linear Algebra

Linear algebra, mathematical discipline that deals with vectors and matrices and, more generally, with vector spaces and linear transformations. Unlike other parts of mathematics that are frequently invigorated by new ideas and unsolved problems, linear algebra is very well understood. Its value lies in its many applications, from mathematical physics to modern algebra and coding theory. Vectors are often expressed using coordinates. For example, in two dimensions a vector can be defined by a pair of coordinates (a1, a2) describing an arrow going from the origin (0, 0) to the point (a1, a2). If one vector is (a1, a2) and another is (b1, b2), then their sum is (a1 + b1, a2 + b2); this gives the same result as the parallelogram (see the figure). In three dimensions a vector is expressed using three coordinates (a1, a2, a3), and this idea extends to any number of dimensions.

# Lab 1

## Topic: Revision on Python functions and Programming

## Date: 18/07/22 to 23/07/22

**Aim: The aim of this lab is to revise the previously learnt concepts in Python programming such as loops, performing basic arithmetic and mathematical questions and plotting various graphs.**

**Source codes and output:**

**1. Write a python program which accepts the user's first and last name and print it with a space between them:**

In [1]:
```
fname = input("First name:")
lname = input("Last name:")
print(fname + " " + lname)
```

```
First name:Arnav
Last name:Sinha
Arnav Sinha
```

**2. Write a python program to check whether a number is even or odd:**

In [2]:
```
x = int(input("Enter number: "))
if x%2==0:
    print(x, "is an even number.")
else:
    print(x, "is odd")
```

```
Enter number: 2
2 is an even number.
```

**3. Write a program to check if two numbers are divisible:**

In [3]:
```
x = int(input("Enter the first number: "))
y = int(input("Enter the second number: "))
if (max(x,y)%min(x,y)==0):
    print("The numbers are divisible.")
else:
    print("The numbers are not divisible.")
```

```
Enter the first number: 4
Enter the second number: 2
The numbers are divisible.
```

**4. Write a python program to find the distance between two points in 2D:**

**4. Write a python program to find the distance between two points in 2D:**

In [4]:
```
from math import sqrt
x1 = int(input("Enter the value of x1: "))
x2 = int(input("Enter the value of x2: "))
y1 = int(input("Enter the value of y1: "))
y2 = int(input("Enter the value of y2: "))
dist = sqrt((x1-x2)**2 + (y1-y2)**2)
print("The distance between (",x1,",",x2,")","and (",y1,",",y2,")","is: ", d
```

```
Enter the value of x1: 1
Enter the value of x2: 3
Enter the value of y1: 4
Enter the value of y2: 2
The distance between ( 1 , 3 ) and ( 4 , 2 ) is:  2.8284271247461903
```

**5. Find the factorial of a number:**

In [5]:
```
a = int(input("Enter a number: "))
fact = 1
while a>1:
    fact=fact*a
    a=a-1
print("The factorial is: ",fact)
```

```
Enter a number: 3
The factorial is:  6
```

**6. Check whether a number lies between two other numbers:**

In [6]:
```
a = int(input("Enter the first number:"))
b = int(input("Enter the second number:"))
c=a-b
d=abs(c)-1
if(c==1|c==-1):
    print("There are no numbers between ",a, " and ",b)
elif (d==1):
    print("There is ",d," number in between ",a, " and ",b)
else:
    print("There are ",d," numbers in between ",a, " and ",b)
```

```
Enter the first number:1
Enter the second number:3
There is  1  number in between  1  and  3
```

**7. Write a program to find the largest of 3 numbers**

In [7]:
```
n1=int(input("Enter the first number:"))
n2=int(input("Enter the second number:"))
n3=int(input("Enter the third number:"))

if(n1>n2):
    if(n1>n3):
        print("The largest number is ",n1)
    else:
        print("The largest number is ",n3)
elif(n2>n1):
    if(n2>n3):
        print("The largest number is ",n2)
    else:
        print("The largest number is ",n3)
elif(n3>n2):
    if(n3>n1):
        print("The largest number is ",n3)
    else:
        print("The largest number is ",n1)

else:
    print("The three numbers are equal.")
```

```
Enter the first number:7
Enter the second number:5
Enter the third number:4
The largest number is  7
```

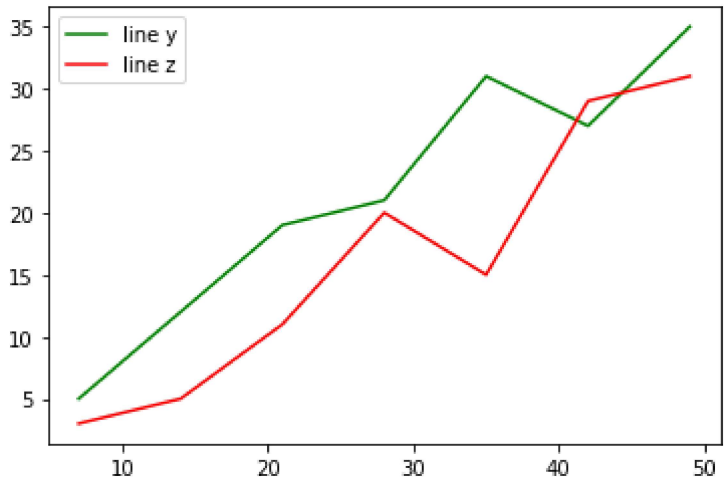
**8. Write a program to find the nth tern in a Fibonacci sequence**

In [9]:
```
import math

def fibonacci(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonacci(n-1) + fibonacci(n-2)
fibonacci(6)
```

Out[9]: 8

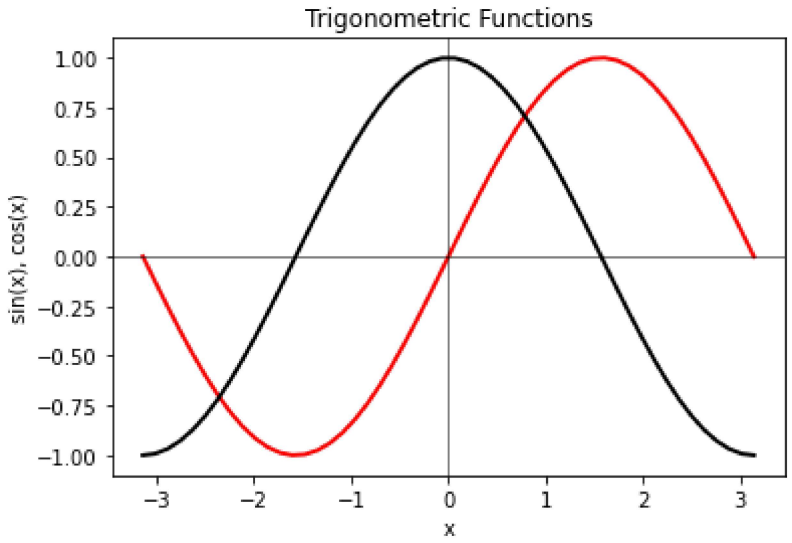
**9. Plot two lines in the same graph using matplotlib library:**

In [10]:
```
import matplotlib.pyplot as plt

x=[7,14,21,28,35,42,49]
y=[5,12,19,21,31,27,35]
z=[3,5,11,20,15,29,31]

##plot a simple line chart with green color and label it as line y
plt.plot(x,y,'g', label='line y')

## plot another line on the same graph with red color and label it as line z
plt.plot(x,z,'r', label='line z')

plt.legend()
plt.show()
```




**10. Plot the graphs of the trigonometric functions Sin(x) and Cos(x) in a single plot**

In [12]:
```
import numpy as np
import matplotlib.pyplot as plt

x=np.linspace(-np.pi, np.pi)
Y1=np.sin(x)
Y2=np.cos(x)

plt.plot(x,Y1, lw=2, color='red', label='Sin(x)')
plt.plot(x,Y2, lw=2, color='black', label='cos(x)')

plt.title('Trigonometric Functions')
plt.xlabel('x')
plt.ylabel('sin(x), cos(x)')
plt.axhline(0, lw=0.5, color='black')
plt.axvline(0, lw=0.5, color='black')
None
```




**11. Plot the graphs of the equations y = x, y = x^2, y = x^3, and y = x^4 as subplots.**

subplot(m,n,p)- m=no of rows, n=now of columns, p=specific position\

divides the current figure into an m-by-n grid and creates axes in the position specified by p\

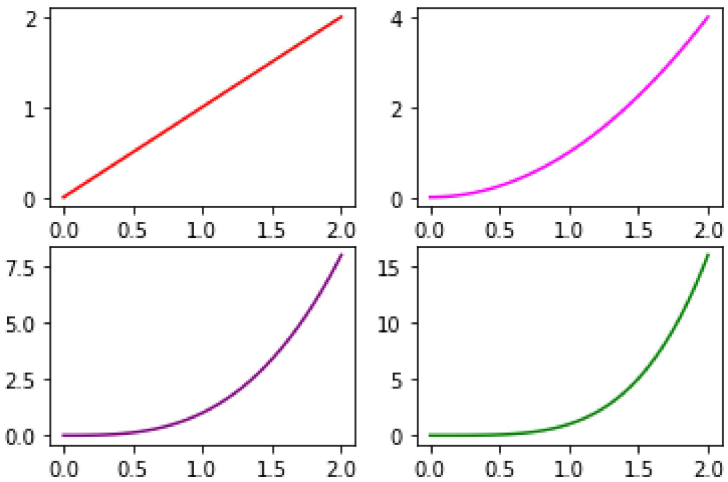
Python subplot positions by row\

The first subplot is the first column if he first row\

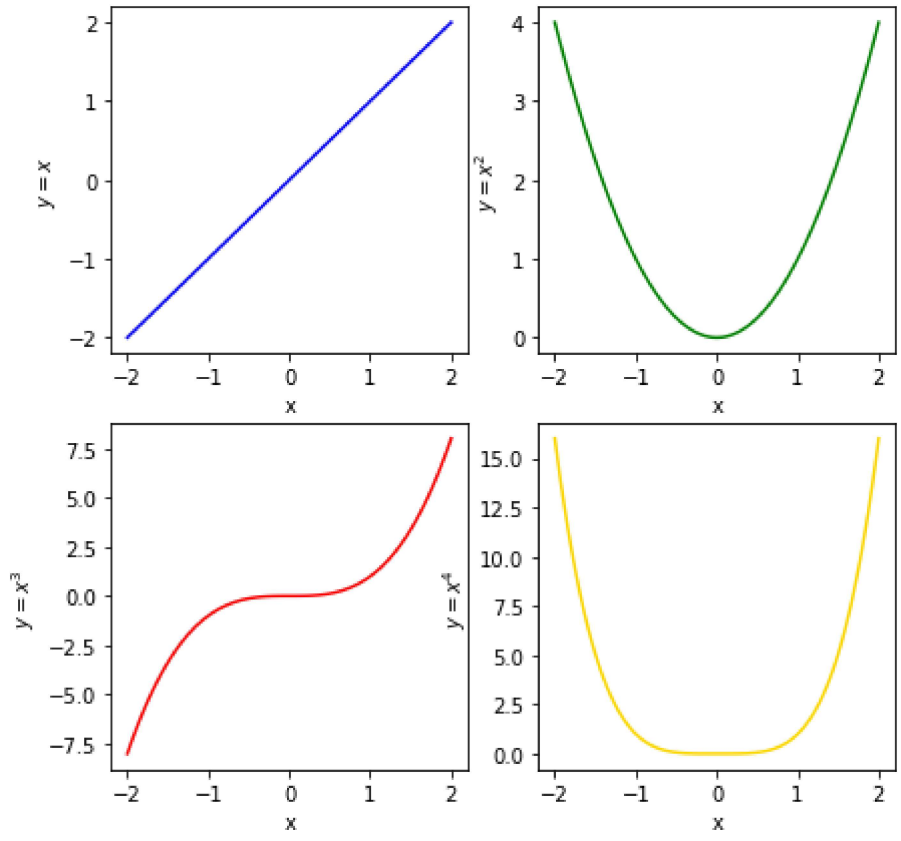
The second subplot is the second column of hte first row\

If axes exist in the specified position, then this command makes the axes the current axes\

In [13]:
```
import numpy as np
import matplotlib.pyplot as plt


x=np.linspace(0,2)
y1=x
y2=x**2
y3=x**3
y4=x**4

plt.subplot(2,2,1)
plt.plot(x,y1,color='red')


plt.subplot(2,2,2)
plt.plot(x,y2,color='magenta')

plt.subplot(2,2,3)
plt.plot(x,y3,color='purple')

plt.subplot(2,2,4)
plt.plot(x,y4,color='green')

None


```

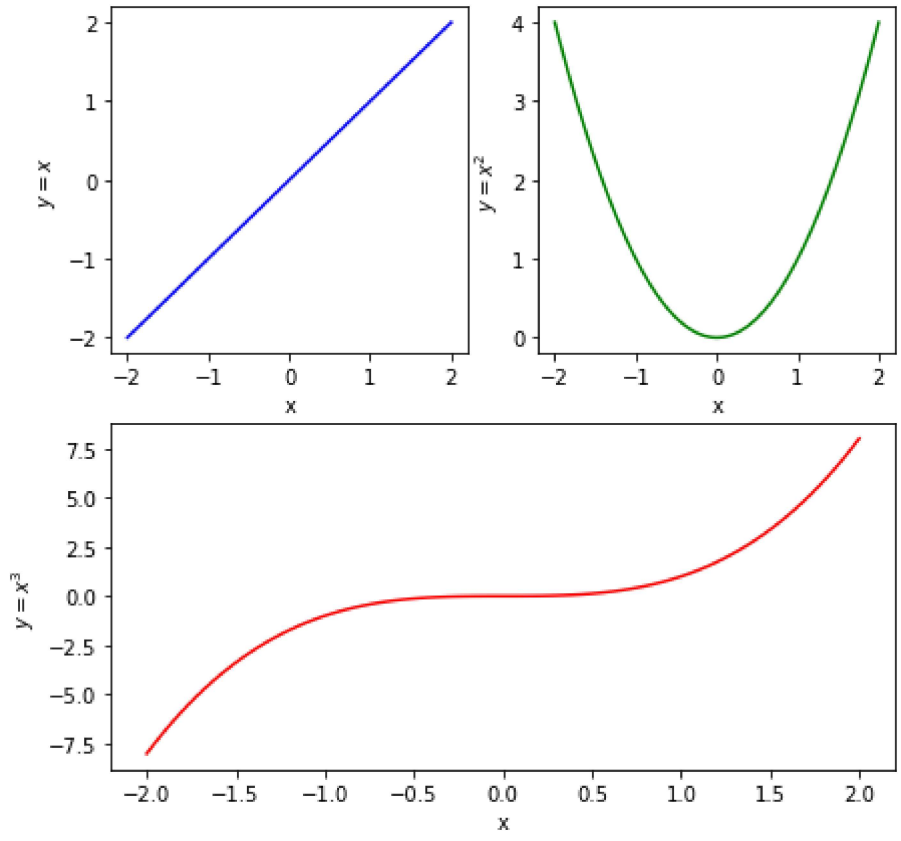


In [14]:
```
plt.figure(figsize=(7,7))#creates a figure of size 7 inches wide and
xvals=np.linspace(-2,2,100)

plt.subplot(2,2,1)#creates an mxn grid and places the plot in position 1
yvals=xvals #y=x
plt.plot(xvals,yvals,color='blue')
plt.xlabel('x')
plt.ylabel('$y=x$')

plt.subplot(2,2,2)#creates an mxn grid and places the plot in position 2
yvals=xvals**2 #y=x^2
plt.plot(xvals,yvals,color='green')
plt.xlabel('x')
plt.ylabel('$y=x^2$')

plt.subplot(2,2,3)#creates an mxn grid and places the plot in position 3
yvals=xvals**3 #y=x^3
plt.plot(xvals,yvals,color='red')
plt.xlabel('x')
plt.ylabel('$y=x^3$')

plt.subplot(2,2,4)#creates an mxn grid and places the plot in position 4
yvals=xvals**4 #y=x^4
plt.plot(xvals,yvals,color='gold')
plt.xlabel('x')
plt.ylabel('$y=x^4$')

None
```



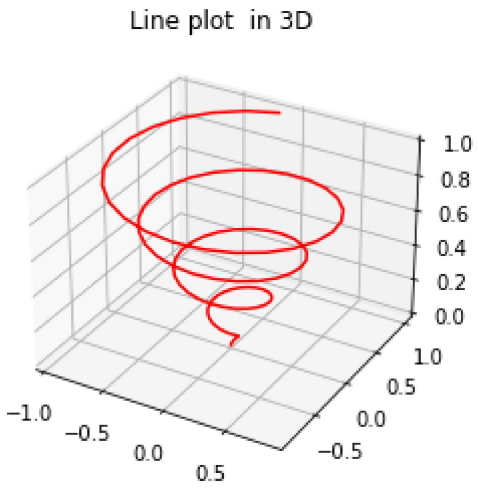
**Q12. For the same program expand the third subplot the next column**

In [15]:
```
plt.figure(figsize=(7,7))#creates a figure of size 7 inches wide and
xvals=np.linspace(-2,2,100)

plt.subplot(2,2,1)#creates an mxn grid and places the plot in position 1
yvals=xvals #y=x
plt.plot(xvals,yvals,color='blue')
plt.xlabel('x')
plt.ylabel('$y=x$')

plt.subplot(2,2,2)#creates an mxn grid and places the plot in position 2
yvals=xvals**2 #y=x^2
plt.plot(xvals,yvals,color='green')
plt.xlabel('x')
plt.ylabel('$y=x^2$')

plt.subplot(2,2,(3,4))#creates an mxn grid and places the plot in position 3
yvals=xvals**3 #y=x^3
plt.plot(xvals,yvals,color='red')
plt.xlabel('x')
plt.ylabel('$y=x^3$')



None
```



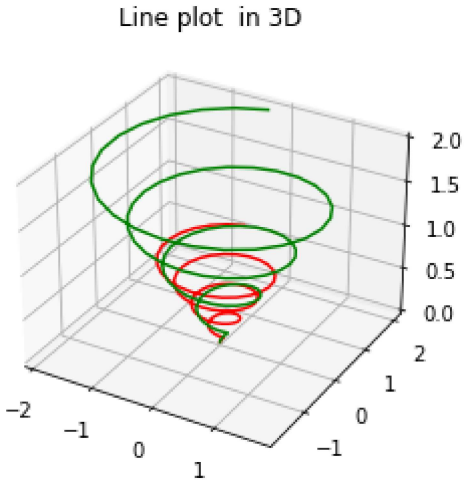

**Q13. Write a python program to draw a 3D projection of a line.**

In [16]:
```
#defining all 3 axes

z=np.linspace(0,1,100)
x=z*np.sin(25*z)
y=z*np.cos(25*z)
ax = plt.axes(projection = '3d') #syntax for 3-D projection

#plotting
ax.plot3D(x,y,z,'red')
ax.set_title('Line plot  in 3D')
plt.show()
```




**Q14. Write a python program to draw a 3D projection of multiple lines in a single plot.**

In [17]:
```
z=np.linspace(0,1,100)
x=z*np.sin(25*z)
y=z*np.cos(25*z)
ax = plt.axes(projection = '3d') #syntax for 3-D projection

#plotting
ax.plot3D(x,y,z,'red')
ax.plot3D(2*x, 2*y, 2*z, 'green')
ax.set_title('Line plot  in 3D')
plt.show()
```




**Conclusion: In this lab, we revised various concepts in Python programming such as loops, basic arithmetic and mathematical questions and plotting various graphs.**

# Lab 2

## Topic: Operations on Matrices

## Date: 25/07/22 to 30/07/22

**Aim: The aim of this lab is to generate matrices, construct various special types of matrices, extract elements from given matrices, perform necessary operations using functions as well as loops and finding the transpose of a matrix. We will also learn the usage of sympy in generating matrices.**

**Source codes and output:**

**Q1. Generate following matrices:**

a. Identity matrix of order 3x3

b. Unity matrix of order 3x3

c. Diagonal matrix of order 4x4

d. Two matrices X and Y of order 5x5

In [18]:
```
import numpy as np

np.identity(3)
```

Out[18]:
```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

In [19]:
```
np.ones([3,3])
```

Out[19]:
```
array([[1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.]])
```

In [20]:
```
np.diag([1,2,3,4])
```

Out[20]:
```
array([[1, 0, 0, 0],
       [0, 2, 0, 0],
       [0, 0, 3, 0],
       [0, 0, 0, 4]])
```

In [21]:
```
X=np.array([[1,2,3,4,5],[6,7,8,9,10],[4,5,6,7,8],[9,7,6,5,4],[2,3,5,6,7]])
X
```

Out[21]:
```
array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10],
       [ 4,  5,  6,  7,  8],
       [ 9,  7,  6,  5,  4],
       [ 2,  3,  5,  6,  7]])
```

In [22]:
```
Y=np.array([[4,5,6,7,8],[2,3,4,1,2],[5,6,3,4,5],[6,7,4,2,1],[8,7,9,8,7]])
Y
```

Out[22]:
```
array([[4, 5, 6, 7, 8],
       [2, 3, 4, 1, 2],
       [5, 6, 3, 4, 5],
       [6, 7, 4, 2, 1],
       [8, 7, 9, 8, 7]])
```

**Q2. Extract elements from a matrix**

In [23]:
```
#a.
print(X[0][0])
print(X[0][3])
print(X[2][3])
print(X[3][3])
```

```
1
4
7
5
```

In [24]:
```
#b.
print(Y[2])
```

```
[5 6 3 4 5]
```

In [25]:
```
#c.
print(X[0])
```

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

In [26]:
```
#d.
print(Y[:,2])
```

```
[6 4 3 4 9]
```

In [27]:
```
#e.
print(Y[:,0])
```

```
[4 2 5 6 8]
```

In [28]:
```
#f.
a,b = X.shape
for i in range(a):
    if i%2 ==0:
        continue
    else:
        print(X[i])
```

```
[ 6  7  8  9 10]
[9 7 6 5 4]
```

In [29]:
```
#g.
a,b = Y.shape
for i in range(a):
    if i%2 == 0:
        print(Y[:,i])
```

```
[4 2 5 6 8]
[6 4 3 4 9]
[8 2 5 1 7]
```

In [30]:
```
#h.
print(np.diag(Y))
```

```
[4 3 3 2 7]
```

**Q3. Extract submatrix**

In [31]:
```
#a.
print(X[1:3,1:3])
```

```
[[7 8]
 [5 6]]
```

In [32]:
```
#b.
print(Y[2:5,2:5])
```

```
[[3 4 5]
 [4 2 1]
 [9 8 7]]
```

**Q4. Perform the following operations on Matrices**

In [33]:
```
#a.
print(X + Y)
```

```
[[ 5  7  9 11 13]
 [ 8 10 12 10 12]
 [ 9 11  9 11 13]
 [15 14 10  7  5]
 [10 10 14 14 14]]
```

In [34]:
```
print(X - Y)
```

```
[[-3 -3 -3 -3 -3]
 [ 4  4  4  8  8]
 [-1 -1  3  3  3]
 [ 3  0  2  3  3]
 [-6 -4 -4 -2  0]]
```

In [35]:
```
print(X*Y)
```

```
[[ 4 10 18 28 40]
 [12 21 32  9 20]
 [20 30 18 28 40]
 [54 49 24 10  4]
 [16 21 45 48 49]]
```

In [36]:
```
print(X/Y)
```

```
[[0.25       0.4        0.5        0.57142857 0.625     ]
 [3.         2.33333333 2.         9.         5.        ]
 [0.8        0.83333333 2.         1.75       1.6       ]
 [1.5        1.         1.5        2.5        4.        ]
 [0.25       0.42857143 0.55555556 0.75       1.        ]]
```

In [37]:
```
print(X.dot(Y))
```

```
[[ 87  92  84  69  66]
 [212 232 214 179 181]
 [162 176 162 135 135]
 [142 165 156 136 149]
 [131 140 126 105 102]]
```

In [38]:
```
#b.
print(np.transpose(X))
```

```
[[ 1  6  4  9  2]
 [ 2  7  5  7  3]
 [ 3  8  6  6  5]
 [ 4  9  7  5  6]
 [ 5 10  8  4  7]]
```

In [39]:
```
#c.
print(np.sqrt(X))
```

```
[[1.         1.41421356 1.73205081 2.         2.23606798]
 [2.44948974 2.64575131 2.82842712 3.         3.16227766]
 [2.         2.23606798 2.44948974 2.64575131 2.82842712]
 [3.         2.64575131 2.44948974 2.23606798 2.        ]
 [1.41421356 1.73205081 2.23606798 2.44948974 2.64575131]]
```

In [40]:
```
#d.
sum = 0
a,b = X.shape
for i in range (a):
    for j in range (b):
        sum = sum + X[i][j]

print(sum)
```

```
139
```

**Q5. Write a python program to find addition of two matrices**

In [1]:

```
import numpy as np

a=int(input("Enter number of rows: "))
b=int(input("Enter number of columns: "))

matrix=[]
for i in range(a):
    c=[]
    for j in range(b):
        j=int(input("Enter elements for first matrix: "))
        c.append(j)
    matrix.append(c)

matrix1=[]
for i in range(a):
    d=[]
    for j in range(b):
        j=int(input("Enter elements for second matrix: "))
        d.append(j)
    matrix1.append(d)

print("First matrix")
for i in range(a):
    for j in range(b):
        print(matrix[i][j],end=" ")
    print()

print("Second matrix")
for i in range(a):
    for j in range(b):
        print(matrix1[i][j],end=" ")
    print()

result=[]
for i in range(a):
    n=[]
    for j in range(b):
        j=0
        n.append(j)
    result.append(n)

for i in range(a):
    for j in range(b):
        result[i][j]=matrix[i][j]+matrix1[i][j]

print("Result matrix addition")
for i in range(a):
    for j in range(b):
        print(result[i][j],end=" ")
    print()
```

```
Enter number of rows: 3
Enter number of columns: 3
Enter elements for first matrix: 1
Enter elements for first matrix: 1
Enter elements for first matrix: 1
```

```
Enter elements for first matrix: 1
Enter elements for first matrix: 1
Enter elements for first matrix: 1
Enter elements for first matrix: 1
Enter elements for first matrix: 1
Enter elements for first matrix: 1
Enter elements for second matrix: 1
Enter elements for second matrix: 1
Enter elements for second matrix: 1
Enter elements for second matrix: 1
Enter elements for second matrix: 1
Enter elements for second matrix: 1
Enter elements for second matrix: 1
Enter elements for second matrix: 1
Enter elements for second matrix: 1
First matrix
1 1 1
1 1 1
1 1 1
Second matrix
1 1 1
1 1 1
1 1 1
Result matrix addition
2 2 2
2 2 2
2 2 2
```

**Q6. Write a python program to find multiplication of two matrices**

In [3]:

```
# import numpy as np

a=int(input("Enter number of rows: "))
b=int(input("Enter number of columns: "))

matrix=[]
for i in range(a):
    c=[]
    for j in range(b):
        j=int(input("Enter elements for first matrix: "))
        c.append(j)
    matrix.append(c)

matrix1=[]
for i in range(a):
    d=[]
    for j in range(b):
        j=int(input("Enter elements for second matrix: "))
        d.append(j)
    matrix1.append(d)

print("First matrix")
for i in range(a):
    for j in range(b):
        print(matrix[i][j],end=" ")
    print()

print("Second matrix")
for i in range(a):
    for j in range(b):
        print(matrix1[i][j],end=" ")
    print()

result=[]
for i in range(a):
    n=[]
    for j in range(b):
        j=0
        n.append(j)
    result.append(n)

for i in range(a):
    for j in range(b):
        for k in range(b):
            result[i][j]+=matrix[i][k]*matrix1[k][j]

print("Result matrix multiplication")
for i in range(a):
    for j in range(b):
        print(result[i][j],end=" ")
    print()
```

```
Enter number of rows: 2
Enter number of columns: 2
Enter elements for first matrix: 1
Enter elements for first matrix: 1
```

```
Enter elements for first matrix: 2
Enter elements for first matrix: 3
Enter elements for second matrix: 2
Enter elements for second matrix: 4
Enter elements for second matrix: 2
Enter elements for second matrix: 1
First matrix
1 1
2 3
Second matrix
2 4
2 1
Result matrix multiplication
4 5
10 11
```

**Q7. Write a program to find transpose of a matrix**

In [4]:

```
## Square matrix

X=[[1,2],[3,4]]
result=[[0,0],[0,0]]

# iterate through rows
for i in range(len(X)):
   # iterate through columns
   for j in range(len(X[0])):
        result[j][i] = X[i][j]

for r in result:
    print(r)
```

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

**Q8. Write a program to square root every element of a matrix**

In [7]:
```
import math
import numpy as np
a=int(input("Enter number of rows: "))
b=int(input("Enter number of columns: "))

matrix=[]
for i in range(a):
    c=[]
    for j in range(b):
        j=int(input("Enter elements for first matrix: "))
        c.append(j)
    matrix.append(c)

print("First matrix")
for i in range(a):
    for j in range(b):
        print(matrix[i][j],end=" ")
    print()

print("Result matrix with sqrt of every element")
for i in range(a):
    for j in range(b):
        print(round(math.sqrt(matrix[i][j]),2),end=" ")
    print()
```

```
Enter number of rows: 1
Enter number of columns: 2
Enter elements for first matrix: 1
Enter elements for first matrix: 2
First matrix
1 2
Result matrix with sqrt of every element
1.0 1.41
```

**Q9. Check if the matrix is symmetric or skew-symmetric**

In [8]:
```
A=np.matrix([[1,2,3],[4,5,6],[7,8,9]])
if(np.array_equal(A,A.T)):
    print("The matrix is symmetric")
elif(np.array_equal(A,-A.T)):
    print("The matrix is skew symmetric")
else:
    print("The matrix is neither skew symmetric nor symmetric")
```

```
The matrix is neither skew symmetric nor symmetric
```

**Q10. Express matrix as a sum of symmetric and skew symmetric matrix**

In [9]:
```
A=np.matrix([[1,2,3],[4,5,6],[7,8,9]])
symB=1/2*(A+A.T)
skewsymC=1/2*(A-A.T)
print("A = symmetric matrix + skew symmetric matrix")
print(A,'=',symB,'+',skewsymC)
```

```
A = symmetric matrix + skew symmetric matrix
[[1 2 3]
 [4 5 6]
 [7 8 9]] = [[1. 3. 5.]
 [3. 5. 7.]
 [5. 7. 9.]] + [[ 0. -1. -2.]
 [ 1.  0. -1.]
 [ 2.  1.  0.]]
```

## Matrix using sympy

In [10]:
```
from sympy import *
from sympy.abc import * ## calls all variable symbols
```

In [11]:
```
A=Matrix([[1,2,3],[4,5,6]])
display(A)
B=Matrix([[1,x,3],[4,y,6]])
display(B)
```

$$\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\end{bmatrix}$$

$$\begin{bmatrix}1 & x & 3\\4 & y & 6\end{bmatrix}$$

In [12]:
```
A+B
```

Out[12]:
$$\begin{bmatrix}2 & x+2 & 6\\8 & y+5 & 12\end{bmatrix}$$

In [13]:
```
3*A+2*B
```

Out[13]:
$$\begin{bmatrix}5 & 2x+6 & 15\\20 & 2y+15 & 30\end{bmatrix}$$

In [14]:
```
A-B
```

Out[14]:
$$\begin{bmatrix}0 & 2-x & 0\\0 & 5-y & 0\end{bmatrix}$$

In [15]:
```
# A/B does not work
```

In [16]:
```
A.T #transpose
```

Out[16]:
$$\begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

In [17]:
```
ones(3)
```

Out[17]:
$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

In [18]:
```
zeros(2,3)
```

Out[18]:
$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

In [19]:
```
eye(3)
```

Out[19]:
$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

In [20]:
```
diag(1,2,3)
```

Out[20]:
$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{bmatrix}$$

In [21]:
```
diag(-1,ones(2,2),Matrix([3,2,1]))
```

Out[21]:
$$\begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 3 \\ 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

In [22]:
```
1 diag(-1,ones(2,2),Matrix([[3,2,1]]))
```

Out[22]:
$$\begin{bmatrix} -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 2 & 1 \end{bmatrix}$$

In [23]:
```
1 S=Matrix([[1,x,3],[2,4,5],[7,1,-x]])
2 display(S)
3 print("The determinant of S is:",S.det())
```

$$\begin{bmatrix} 1 & x & 3 \\ 2 & 4 & 5 \\ 7 & 1 & -x \end{bmatrix}$$

```
The determinant of S is: 2*x**2 + 31*x - 83
```

In [24]:
```
1 solve(S.det())
```

Out[24]:
```
[-31/4 + 5*sqrt(65)/4, -5*sqrt(65)/4 - 31/4]
```

In [25]:
```
1 S.rref()
```

Out[25]:
```
(Matrix([
 [1, 0, 0],
 [0, 1, 0],
 [0, 0, 1]]),
 (0, 1, 2))
```

**Conclusion: We learnt how to generate matrices, construct various special types of matrices, extract elements from given matrices, perform necessary operations using functions as well as loops and finding the transpose of a matrix. We also learnt the usage of sympy in generating matrices.**

# Lab 3

## Topic: Finding Rank of Matrices

## Date: 01/08/22 to 06/08/22

**Aim: The aim of this practical is to obtain the rank of matrices using the basic function. We will also learn to perform elementary row and column operations and use it to find ranks individually as well as using normal form.**

**Source codes and output:**

In [26]:
```
## echelon matrix
import sympy
sympy.Matrix(A).echelon_form()
```

Out[26]:
$$\begin{bmatrix} 1 & 2 & 3 \\ 0 & -3 & -6 \end{bmatrix}$$

In [27]:
```
#rank
A.rank()
```

Out[27]: 2

## Elementary row operations

**Syntax: elementary_row_op**

Performs the elementary row operations

1. n->kn (row n becomes kn)
2. n<->m (swap rows)
3. n->n+km (row n becomes row n + k* row m)

In [28]:
```
import numpy as np
from sympy import Matrix
A=np.mat([[4,3,2,5],
              [1,4,1,9],
              [3,10,4,1]])
A=Matrix(A)
display(A)
```

$$\begin{bmatrix} 4 & 3 & 2 & 5 \\ 1 & 4 & 1 & 9 \\ 3 & 10 & 4 & 1 \end{bmatrix}$$

In [29]:
```
## 1. Scalar multiplication of a row (positive)
A.elementary_row_op(op='n->kn', row=1,k=2)
```

Out[29]:
$$\begin{bmatrix} 4 & 3 & 2 & 5 \\ 2 & 8 & 2 & 18 \\ 3 & 10 & 4 & 1 \end{bmatrix}$$

In [30]:
```
## 1. Scalar multiplication of a row (negative)
A.elementary_row_op(op='n->kn', row=2,k=-3)
```

Out[30]:
$$\begin{bmatrix} 4 & 3 & 2 & 5 \\ 1 & 4 & 1 & 9 \\ -9 & -30 & -12 & -3 \end{bmatrix}$$

In [31]:
```
## 2. Row swap
A.elementary_row_op(op='n<->m', row1=1,row2=2)
```

Out[31]:
$$\begin{bmatrix} 4 & 3 & 2 & 5 \\ 3 & 10 & 4 & 1 \\ 1 & 4 & 1 & 9 \end{bmatrix}$$

In [32]:
```
## 3. Row change based on another row
A.elementary_row_op(op='n->n+km', k=3,row1=1,row2=2)
```

Out[32]:
$$\begin{bmatrix} 4 & 3 & 2 & 5 \\ 10 & 34 & 13 & 12 \\ 3 & 10 & 4 & 1 \end{bmatrix}$$

**Q1. Perform row operations on the following matrix**

In [33]:
```
B=np.mat([[1,2,3],
              [4,5,6],
              [7,8,9]])
B=Matrix(B)
display(B)
```

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

In [34]:
```
B=B.elementary_row_op(op='n->n+km', k=-4,row1=1,row2=0)
B=B.elementary_row_op(op='n->n+km', k=-7,row1=2,row2=0)
B
```

Out[34]:
$$\begin{bmatrix} 1 & 2 & 3 \\ 0 & -3 & -6 \\ 0 & -6 & -12 \end{bmatrix}$$

**Q2. Considering B, we use row operations to reduce it to Echleon form and find rank**

In [35]:
```
display(B)
```

$$\begin{bmatrix} 1 & 2 & 3 \\ 0 & -3 & -6 \\ 0 & -6 & -12 \end{bmatrix}$$

In [36]:
```
C=B.elementary_row_op(op='n->kn', k=-1/3,row=1)
D=C.elementary_row_op(op='n->kn', k=-1/6,row=2)
D
```

Out[36]:
$$\begin{bmatrix} 1 & 2 & 3 \\ 0 & 1.0 & 2.0 \\ 0 & 1.0 & 2.0 \end{bmatrix}$$

In [37]:
```
E=D.elementary_row_op(op='n->n+km', k=-1,row1=2,row2=1)
E
```

Out[37]:
$$\begin{bmatrix} 1 & 2 & 3 \\ 0 & 1.0 & 2.0 \\ 0 & 0 & 0 \end{bmatrix}$$

**Ans) The rank of the following matrix is 2**

## Elementary column operations

**Syntax: elementary_col_op**

Performs the elementary column operations

1. n->kn (col n becomes kn)
2. n<->m (swap cols)
3. n->n+km (col n becomes col n + k* col m)

In [38]:
```
S=np.mat([[4,2,3],
              [1,5,6],
              [-2,8,-8]])
S=Matrix(S)
display(S)
```

$$\begin{bmatrix} 4 & 2 & 3 \\ 1 & 5 & 6 \\ -2 & 8 & -8 \end{bmatrix}$$

**Perform the column operation C1<--> C2**

In [39]:
```
S1=S.elementary_col_op(op='n<->m', col1=0,col2=1)
S1
```

Out[39]:
$$\begin{bmatrix} 2 & 4 & 3 \\ 5 & 1 & 6 \\ 8 & -2 & -8 \end{bmatrix}$$

**Perform the column operation C3-->C3-5C2**

In [40]:
```
S2=S.elementary_col_op(op='n->n+km', k=-5,col1=2,col2=1)
S2
```

Out[40]:
$$\begin{bmatrix} 4 & 2 & -7 \\ 1 & 5 & -19 \\ -2 & 8 & -48 \end{bmatrix}$$

## Normal form

In [41]:
```
display(B)
```

$$\begin{bmatrix} 1 & 2 & 3 \\ 0 & -3 & -6 \\ 0 & -6 & -12 \end{bmatrix}$$

In [42]:
```
C=B.elementary_col_op(op='n->n+km', k=-2,col1=1,col2=0)
D=C.elementary_col_op(op='n->n+km', k=-3,col1=2,col2=0)
D
```

Out[42]:
$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -3 & -6 \\ 0 & -6 & -12 \end{bmatrix}$$

In [43]:
```
E=D.elementary_row_op(op='n->kn', k=-1/3,row=1)
E
```

Out[43]:
$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1.0 & 2.0 \\ 0 & -6 & -12 \end{bmatrix}$$

In [44]:
```
F=E.elementary_col_op(op='n->n+km', k=-2,col1=2,col2=1)
F
```

Out[44]:
$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1.0 & 0 \\ 0 & -6 & 0 \end{bmatrix}$$

In [45]:
```
G=F.elementary_row_op(op='n->n+km', k=6,row1=2,row2=1)
G
```

Out[45]:
$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1.0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

**By reducing to normal form, the rank is 2.**

**Conclusion: In this lab, we have learnt how to find the rank of matrices using the basic function. We also learnt to perform elementary row and column operations and used it to find ranks individually as well as using normal form.**

# Lab 4

## Topic: Reducing a matrix to Echelon form

## Date: 08/08/22 to 20/08/22

**Aim: The aim of this lab is to obtain matrices in row reduced echelon form using functions as well as loops.**

**Source codes and output:**

**sympy.Matrix(A).echelon_form() and sympy.Matrix(A).rref() converts matrix into row echelon form.**

In [46]:
```
import sympy
sympy.Matrix(A).echelon_form()
```

Out[46]:
$$\begin{bmatrix} 4 & 3 & 2 & 5 \\ 0 & 13 & 2 & 31 \\ 0 & 0 & 68 & -1104 \end{bmatrix}$$

In [47]:
```
sympy.Matrix(A).rref()
## output in the form of tuple
## last row gives index of pivot elements for each row
```

Out[47]:
```
(Matrix([
 [1, 0, 0,   97/17],
 [0, 1, 0,   83/17],
 [0, 0, 1, -276/17]]),
 (0, 1, 2))
```

In [48]:
```
A.rank()
```

Out[48]:
```
3
```

**Write a program to reduce the user defined matrix to Echelon form using the row operations.**

In [49]:
```
import numpy as np
from sympy import Matrix, pprint
A=np.mat(input("Enter matrix: "))
(r,c)=A.shape
A=Matrix(A)
pprint("The given matrix is:" )
pprint(A)
```

```
Enter matrix: 1,2,3;1,2,3;1,2,3
The given matrix is:
⎡1  2  3⎤
⎢       ⎥
⎢1  2  3⎥
⎢       ⎥
⎣1  2  3⎦
```

In [50]:
```
import numpy as np
from sympy import Matrix, pprint
A=np.mat(input("Enter a matrix:"))
(r,c)=A.shape
A=Matrix(A)
print("The given matrix is :")
pprint(A)
b=0
for j in range(c):
    for i in range(b+1,r):
        x=0
        Flag= False
        if (A[b,j]==0):
            for k in range(b+1,r):
                if (A[k,j] !=0):
                    A=A.elementary_row_op(op='n<->m', row1=k, row2=b)
                    x=1
                    break
        if (A[b,j]==0 and x==0):
            Flag= True
            break
        A=A.elementary_row_op(op='n->n+km', row=i, k=-(A[i,j]/A[b,j]), row1=
    if Flag:
        continue
    else:
        b+=1
print("The echelon form of the matrix is:")
pprint(A)
```

```
Enter a matrix:1,2,3;1,2,3;1,2,3
The given matrix is :
⎡1  2  3⎤
⎢       ⎥
⎢1  2  3⎥
⎢       ⎥
⎣1  2  3⎦
The echelon form of the matrix is:
⎡1  2  3⎤
⎢       ⎥
⎢0  0  0⎥
⎢       ⎥
⎣0  0  0⎦
```

## Normal form

In [53]:
```
import numpy as np
from sympy import Matrix, pprint
A=np.mat(input("Enter a matrix:"))
A=Matrix(A)
print("The given matrix is :")
pprint(A)
A=A.elementary_row_op(op='n->n+km', row=None, k=-5, row1=1, row2=0)
A=A.elementary_col_op(op='n->kn', col=None, k=-1/4, col1=1)
A=A.elementary_row_op(op='n->n+km', row=None, k=-9, row1=2, row2=0)
A=A.elementary_col_op(op='n->kn', col=None, k=-1/16, col1=2)
display(A)
A=A.elementary_row_op(op='n->n+km', row=None, k=-2, row1=2, row2=1)
A=A.elementary_col_op(op='n->n+km', col=None, k=12, col1=3,col2=1)
A=A.elementary_row_op(op='n->n+km', row=None, k=0.5, row1=0, row2=1)
A=A.elementary_col_op(op='n->n+km', col=None, k=-0.5, col1=2,col2=1)
display(A)
A=A.elementary_row_op(op='n->n+km', row=None, k=5, row1=0, row2=2)
A=A.elementary_col_op(op='n->n+km', col=None, k=-0.0625, col1=2,col2=0)
display(A)
A=A.elementary_row_op(op='n->kn', row=None, k=-10, row1=2)
display(A)
A=A.elementary_col_op(op='n->n+km', col=None, k=2, col1=3,col2=0)
display(A)
display(A)
```

```
Enter a matrix:1,2,3,4;5,6,7,8;9,10,11,12
The given matrix is :
⎡1  2   3   4 ⎤
⎢             ⎥
⎢5  6   7   8 ⎥
⎢             ⎥
⎣9  10  11  12⎦
```

$$\begin{bmatrix} 1 & -0.5 & -0.1875 & 4 \\ 0 & 1.0 & 0.5 & -12 \\ 0 & 2.0 & 1.0 & -24 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0.0625 & -2.0 \\ 0 & 1.0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & -2.0 \\ 0 & 1.0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & -2.0 \\ 0 & 1.0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1.0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1.0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

In [54]:

```
import numpy as np
from sympy import Matrix, pprint
A=np.mat(input("Enter a matrix:"))
(r,c)=A.shape
A=Matrix(A)
print("The given matrix is :")
pprint(A)
b=0
for j in range(c):
    for i in range(b+1,r):
        x=0
        Flag= False
        if (A[b,j]==0):
            for k in range(b+1,r):
                if (A[k,j] !=0):
                    A=A.elementary_row_op(op='n<->m', row1=k, row2=b)
                    x=1
                    break
        if (A[b,j]==0 and x==0):
            Flag= True
            break
        A=A.elementary_row_op(op='n->n+km', row=i, k=-(A[i,j]/A[b,j]), row1=
    if Flag:
        continue
    else:
        b+=1
b=0
for i in range(r):
    for j in range(b+1,c):
        x=0
        Flag= False
        if (A[i,b]==0):
            for k in range(b+1,c):
                if (A[i,k] !=0):
                    A=A.elementary_col_op(op='n<->m', col1=i, col2=b)
                    x=1
                    break
        if (A[i,b]==0 and x==0):
            Flag= True
            break
        A=A.elementary_col_op(op='n->n+km', col=j, k=-(A[i,j]/A[i,b]), col1=
    if Flag:
        continue
    else:
        b+=1
for i in range(min(r,c)):
    if A[i,i]!=0:
        A=A.elementary_row_op(op='n->kn',row=i,k=(1/A[i,i]),row1=i,row2=i)
print("The normal form of the matrix is:")
pprint(A)

rank=0
for i in range(min(r,c)):
    if A[i,i]!=0:
        rank+=1
print("The rank of the matrix is: ",rank)
```

```
Enter a matrix:1,2,3;4,5,6;7,8,9
The given matrix is :
⎡1  2  3⎤
⎢       ⎥
⎢4  5  6⎥
⎢       ⎥
⎣7  8  9⎦
The normal form of the matrix is:
⎡1  0  0⎤
⎢       ⎥
⎢0  1  0⎥
⎢       ⎥
⎣0  0  0⎦
The rank of the matrix is:  2
```

**Conclusion: We learnt how to obtain matrices in row reduced echelon form using functions as well as loops.**

# Lab 5

## Topic: Inverse of matrix by different methods

## Date: 22/08/22 to 03/09/22

**Aim: Through this lab, we aim to find the inverse of a matrix using different methods, specifically using the Gauss Jordan Method.**

**Source codes and output:**

## Gauss Jordan method of finding inverse of matrix

In [55]:
```
import numpy as np
from sympy import Matrix, pprint
A=np.mat(input("Enter a matrix:"))
A=Matrix(A)
print("The given matrix is :")
pprint(A)
```

```
Enter a matrix:1,2,3;5,6,7;1,2,9
The given matrix is :
⎡1  2  3⎤
⎢       ⎥
⎢5  6  7⎥
⎢       ⎥
⎣1  2  9⎦
```

In [57]:
```
from numpy import *
```

In [56]:
```
## Checking if matrix is singlular

if A.det()==0:
    print("Matrix is singular, thus, not possible")
else:
    print("Matrix is not singular")
```

```
Matrix is not singular
```

In [58]:
```
A.det()
```

Out[58]: $-24$

In [59]:
```
I=np.identity(3)
I
```

Out[59]:
```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

In [60]:
```
C=np.concatenate((A, I))
C
```

Out[60]:
```
array([[1, 2, 3],
       [5, 6, 7],
       [1, 2, 9],
       [1.0, 0.0, 0.0],
       [0.0, 1.0, 0.0],
       [0.0, 0.0, 1.0]], dtype=object)
```

In [61]:
```
## Augmented matrix
A=C.T
A
```

Out[61]:
```
array([[1, 5, 1, 1.0, 0.0, 0.0],
       [2, 6, 2, 0.0, 1.0, 0.0],
       [3, 7, 9, 0.0, 0.0, 1.0]], dtype=object)
```

**Conclusion: Through this practical, we were able to find the inverse of a matrix using the Gauss Jordan Method (provided inverse exists).**

# Lab 6

## Topic: Solving system of Equations using various methods

## Date: 05/09/22 to 17/09/22

**Aim: The aim of this practical is to solve the system of linear equations using different methods in the form of matrices and display the solutions of the same.**

**Source codes and output:**

### Q1. Solve the system of linear equations

In [62]:
```
#a.
A = np.array([[1,1], [3,2]])
B = np.array([[-2],[0]])
print(np.linalg.solve(A,B))
```

```
[[ 4.]
 [-6.]]
```

In [63]:
```
#b.
A = np.array([[2,-4, 6], [4,2,6], [4,2,-10]])
B = np.array([[10],[30], [50]])
print(np.linalg.solve(A,B))
```

```
[[ 9.25]
 [ 0.25]
 [-1.25]]
```

### Q2. Solve the given word problem

An amount of $Rs$.500 is put into three investments at the rate of interest of 6%,7%,8% per annum respectively. The total annual income from these investmenets is $Rs$.358 . If the combined income from the first investments is $Rs$.70 more than the income from the third, find the amount of each investment.

In [64]:
```
#corrected the question slightly
#changed investment from 500 to 5000
#since the answer for x = -30500 is not mathematically sound

#using sympy

from sympy import *
x,y,z = symbols('x,y,z')
a = Eq(x + y + z, 5000)
b = Eq(6*x/100 + 7*y/100 + 8*z/100, 358)
c = Eq(6*x/100 + 7*y/100, 70 + 8*z/100)
solve([a,b,c], (x,y,z))
```

Out[64]:
```
{x: 1000, y: 2200, z: 1800}
```

In [65]:
```
#using matrix operators in numpy
A = np.array([[1,1,1], [6/100, 7/100, 8/100], [6/100, 7/100, -8/100]])
B = np.array([[5000],[358], [70]])
print(np.linalg.solve(A,B))
```

```
[[1000.]
 [2200.]
 [1800.]]
```

**Conclusion: Through this practical, we were able to solve the system of linear equations using matrices and solve some real time equations with the help of the matrix methods.**

# Lab 7

## Topic: Finding Eigen values and Eigenvectors of a matrix

## Date: 19/09/22 to 24/09/22

**Aim: The aim of this lab is to find the eigen values and eigen vectors of a given matrix using sympy package. We also prove several properties relating to the eigen values and eigen vectors. Cayley Hamilton theorem is proved and discussed and the technique of diagonalization is also applied.**

**Source codes and output:**

In [66]:
```
import numpy as np
from sympy import *
from sympy.abc import *
from numpy.linalg import eig
```

**Find the eigen values and eigen vectors of the given matrix**

$$\begin{bmatrix} 1 & 2 & 3 \\ 0 & 2 & 3 \\ 0 & 0 & 3 \end{bmatrix}$$

In [67]:
```
A = np.array([[1,2,3],
              [0,2,3],
             [0,0,3]])
w,v=eig(A)

# printing eigen values
print("Printing the Eigen values of the given square array:\n",w)

# printing eigen vectors
print("Printing Right eigenvectors of the given square array:\n",v)
```

```
Printing the Eigen values of the given square array:
 [1. 2. 3.]
Printing Right eigenvectors of the given square array:
 [[1.         0.89442719 0.81818182]
 [0.         0.4472136  0.54545455]
 [0.         0.         0.18181818]]
```

In [68]:
```
np.linalg.eigvals(A)
```

Out[68]:
```
array([1., 2., 3.])
```

In [69]:
```
1 np.linalg,eig(A)
```

Out[69]:
```
(<module 'numpy.linalg' from 'C:\\Users\\12138\\anaconda3\\lib\\site-packages
\\numpy\\linalg\\__init__.py'>,
 (array([1., 2., 3.]),
  array([[1.        , 0.89442719, 0.81818182],
         [0.        , 0.4472136 , 0.54545455],
         [0.        , 0.        , 0.18181818]])))
```

**Example 2**

$$\begin{bmatrix} 4 & 3 & 2 \\ 1 & 4 & 1 \\ 3 & 10 & 4 \end{bmatrix}$$

In [70]:
```
1  B = np.array([[4,3,2],
2                [1,4,1],
3                [3,10,4]])
4  w,v=eig(B)
5  ## w gives eigen values and v gives eigen vectors
6
7  # printing eigen values
8  print("Printing the Eigen values of the given square array:\n",w)
9
10 # printing eigen vectors
11 print("Printing eigenvectors of the given square array:\n",v)
```

```
Printing the Eigen values of the given square array:
 [8.98205672 2.12891771 0.88902557]
Printing eigenvectors of the given square array:
 [[-0.49247712 -0.82039552 -0.42973429]
 [-0.26523242  0.14250681 -0.14817858]
 [-0.82892584  0.55375355  0.89071407]]
```

In [71]:
```
1 for i in range(0,3):
2     print("The eigen vector for eigen value ",w[i],"is ",v[:,i])
```

```
The eigen vector for eigen value  8.982056720677651 is  [-0.49247712 -0.2652324
2 -0.82892584]
The eigen vector for eigen value  2.12891770502734 is  [-0.82039552  0.14250681
0.55375355]
The eigen vector for eigen value  0.8890255742950103 is  [-0.42973429 -0.148178
58  0.89071407]
```

**Properties:**

1. For a nxn matrix, the number of eigen values is n.
2. The sum of the eigen values is equal to the sum of the diaginal elements of the matrix.
3. The product of the eigen values is equal to the determinant of the matrix.
4. The eigen value of identity matrix is equal to 1.
5. The eigen value of a triangular matrix is same as diagonal elements of the matrix.

6. For a skew symmetric matrix, the eigen values are imaginary.
7. For orthogonal matrix, the length of eigenvalues equals 1.
8. For idempotent matrix,the eigen values are 0 and 1.

**Property 1**

In [72]:
```
print("Number of eigen values is ",len(w))
```

```
Number of eigen values is  3
```

In [73]:
```
print("Order of matrix is ", np.shape(B))
```

```
Order of matrix is  (3, 3)
```

In [74]:
```
print("For a nxn matrix, the number of eigen values is n.")
```

```
For a nxn matrix, the number of eigen values is n.
```

**Property 2**

In [75]:
```
print("The sum of diagonal elements of matrix ",np.trace(B))
print("Sum of eigen values is ",sum(w))
print("The sum of the eigen values is equal to the sum of the diaginal eleme
```

```
The sum of diagonal elements of matrix  12
Sum of eigen values is  12.000000000000002
The sum of the eigen values is equal to the sum of the diaginal elements of the
matrix.
```

**Property 3**

In [76]:
```
B = Matrix([[4,3,2],
            [1,4,1],
            [3,10,4]])
print("The determinant of matrix is, ",det(B))

print("The product of eigen values is ",prod(w))

print("The product of the eigen values is equal to the determinant of the ma
```

```
The determinant of matrix is,  17
The product of eigen values is  17.000000000000007
The product of the eigen values is equal to the determinant of the matrix.
```

**Property 4**

In [77]:
```
C = np.array([[1,0,0],
              [0,1,0],
              [0,0,1]])
w,v=eig(C)
## w gives eigen values and v gives eigen vectors

# printing eigen values
print("Printing the Eigen values of the given square array:\n",w)
print("The eigen value of identity matrix is equal to 1")
```

```
Printing the Eigen values of the given square array:
 [1. 1. 1.]
The eigen value of identity matrix is equal to 1
```

**Property 5**

In [78]:
```
T=np.array([[4,0,0],[2,3,0],[1,2,3]])
print("The eigen values of T are ",np.linalg.eigvals(T))
print("The eigen value of a triangular matrix is same as diagonal elements o
```

```
The eigen values of T are  [3. 3. 4.]
The eigen value of a triangular matrix is same as diagonal elements of the matr
ix
```

**Property 6**

In [79]:
```
SS=np.array([[1,3,2],
             [-3,1,1],
             [-2,-1,1]])
w,v=eig(SS)
## w gives eigen values and v gives eigen vectors

# printing eigen values
print("Printing the Eigen values of the given square array:\n",w)
print("The eigen values of skew symmetric matrix are imaginary")
```

```
Printing the Eigen values of the given square array:
 [1.+3.74165739j 1.-3.74165739j 1.+0.j        ]
The eigen values of skew symmetric matrix are imaginary
```

**Property 7**

In [80]:
```
import numpy as np
from scipy.linalg import qr

n = 3
H = np.random.randn(n, n)
Q, R = qr(H)
AB=Q.dot(Q.T)
print (AB)
w,v=eig(AB)
print("Eigen vectors are ",w)
print("For orthogonal matrix, the length of eigenvalues equals 1")
```

```
[[ 1.00000000e+00 -4.84452883e-17 -5.09600086e-17]
 [-4.84452883e-17  1.00000000e+00  1.97296496e-16]
 [-5.09600086e-17  1.97296496e-16  1.00000000e+00]]
Eigen vectors are  [1. 1. 1.]
For orthogonal matrix, the length of eigenvalues equals 1
```

**Cayley Hamilton theorem**

In [81]:
```
import numpy as np
from sympy import *
```

**Method to prove CHT**

In [82]:
```
from math import *
import numpy as np

A=np.mat([[2,3],[4,5]])
X=np.poly(A)
print("Coefficients of characteristic matrix are ",X)
```

```
Coefficients of characteristic matrix are  [ 1. -7. -2.]
```

In [83]:
```
trace=np.trace(A)
print(trace)
```

```
7
```

In [84]:
```
det=np.linalg.det(A)
det
```

Out[84]:
```
-2.0
```

In [85]:
```
I=np.eye(2)
I
```

Out[85]:
```
array([[1., 0.],
       [0., 1.]])
```

In [86]:
```
Final=A*A-trace*A+det*I
print(Final)
```

```
[[0. 0.]
 [0. 0.]]
```

In [87]:
```
### Hence, proved.
```

In [88]:
```
from numpy import *

print("Enter elements of the matrix: ")
A = mat(input())
s = 0 ## empty sum
print() ## prints extra space
print("The matrix is : ")
print(A)
print()
I = eye(len(A), len(A))
print("The identity matrix is: ")
print(I) ## I
print()
ce = poly(A) ## coefficients of charactersitic eqn
ce = ce.round()
print("The coefficients of the characteristic equation = ", ce)
for i in range(len(ce)):
    eq = ce[i]*I*(A**(len(ce)-i)) ## repeated the number of times there are
    s = s+eq ## for getting aA^3+bA^2+cA+dI
print()
print("Replacing given matrix in characteristic equation: ")
print(s) ## sum of the characteristic eqn should be zero
print("Cayley Hamilton theorem is proved.")
```

```
Enter elements of the matrix:
1,2,3;5,6,7;1,9,10

The matrix is :
[[ 1  2  3]
 [ 5  6  7]
 [ 1  9 10]]

The identity matrix is:
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]

The coefficients of the characteristic equation =  [  1. -17.  -0. -28.]

Replacing given matrix in characteristic equation:
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
Cayley Hamilton theorem is proved.
```

**Diagonalization**

In [90]:
```
B = np.array([[1,2,3],
              [0,2,3],
             [0,0,3]])
w,v=eig(B)
## w gives eigen values and v gives eigen vectors

# printing eigen values
print("Printing the Eigen values of the given square array:\n",w)

# printing eigen vectors
print("Printing eigenvectors of the given square array:\n",v)
```

```
Printing the Eigen values of the given square array:
 [1. 2. 3.]
Printing eigenvectors of the given square array:
 [[1.         0.89442719 0.81818182]
 [0.         0.4472136  0.54545455]
 [0.         0.         0.18181818]]
```

In [91]:
```
## transpose of eigen vector matrix
P=v.T
P
```

Out[91]:
```
array([[1.        , 0.        , 0.        ],
       [0.89442719, 0.4472136 , 0.        ],
       [0.81818182, 0.54545455, 0.18181818]])
```

In [92]:
```
## finding inverse of P matrix
P_in=np.linalg.inv(P)
P_in
```

Out[92]:
```
array([[ 1.00000000e+00,  0.00000000e+00,  0.00000000e+00],
       [-2.00000000e+00,  2.23606798e+00, -6.70759744e-17],
       [ 1.50000000e+00, -6.70820393e+00,  5.50000000e+00]])
```

In [93]:
```
## Diagonalization
## Finding P inverse * B * P
D=P_in*B*P
D.astype(int)
```

Out[93]:
```
array([[1, 0, 0],
       [0, 2, 0],
       [0, 0, 3]])
```

The matrix B is diagonalized as we have found matrix P such that D = P inverse * B * P gives a matrix with all non-diagonal elements as zero and diagonal elements as the eigen valyes, 1,2 and 3.

In [94]:
```
B=Matrix(B)
x,y=B.diagonalize()
print(y)
```

```
Matrix([[1, 0, 0], [0, 2, 0], [0, 0, 3]])
```

**Conclusion: Through the same, we learned how to use the "eig" function in the Sympy package to extract the eigen values and eigen vectors for a matrix. We also learned about the diagonalization procedure and proved various properties connected to the same Cayley Hamilton theorem.**

# Lab 8

## Topic: Expressing a vector as a linear combination of given set of vectors

## Date: 10/10/22 to 15/10/22

**Aim: The aim of this lab is to explore the concept of linear combination of vectors.**

**Source codes and output:**

**Q1. Express < 2, 13, 6 > as linear combination of < 1, 5, − 1 > , < 1, 2, 1 > and < 1, 4, 3 >.**

In [95]:
```
x = np.array([[1, 5,-1],
              [1,2, 1],
              [1,4,3]])

y = ([2, 13, 6])

coeff = np.linalg.solve(x, y)
print("The values of constants are: ",coeff)
print()
print("The Linear Combination is ")
print(" ")
print(y,"=",np.round(coeff[0]),"",x[0],"+",np.round(coeff[1]),"",x[1],"+",np
```

```
The values of constants are:  [20.1 -3.6  0.1]

The Linear Combination is

[2, 13, 6] = 20.0  [ 1  5 -1] + -4.0  [1 2 1] + 0.0 * [1 4 3]
```

**Q2. Take inputs about vectors and express as linear combination**

In [97]:
```
import numpy as np
from sympy import Matrix, pprint

A=np.mat(input("Enter the coefficients matrix:"))
A=Matrix(A)
print("The given coefficient matrix is :")
pprint(A)
A_1=A.T
(r,c)=A_1.shape
B=np.mat(input("Enter the matrix of solutions:"))
(r1,c1)=B.shape
pprint(B)
if (c==r1):
    alpha = np.linalg.solve(A_1, B)
    pprint(alpha)
else:
    print("The solution is not possible")
```

```
Enter the coefficients matrix:1,2,3;4,5,6;7,8,9
The given coefficient matrix is :
⎡1  2  3⎤
⎢       ⎥
⎢4  5  6⎥
⎢       ⎥
⎣7  8  9⎦
Enter the matrix of solutions:5,3,2
[[5 3 2]]
The solution is not possible
```

**Q3. Determine whether the vector <2,1,3> is a linear combination of the vectors <1,2,3> , <3,2,1> and <2,3,1>.**

In [98]:
```
x = np.array([[1, 2, 3],
              [3,2, 1],
              [2, 3, 1]])

y = ([2, 1, 3])

coeff = np.linalg.solve(x.T, y)
print("The values of constants are: ",coeff)
print(" ")

print("The Linear Combination is ")
print(" ")
print(y,"=",np.round(coeff[0]),"",x[0],"+",np.round(coeff[1]),"",x[1],"+",np
```

```
The values of constants are:  [ 1.  1. -1.]

The Linear Combination is

[2, 1, 3] = 1.0  [1 2 3] + 1.0  [3 2 1] + -1.0 * [2 3 1]
```

In [99]:
```
## Verification

coeff=np.matrix(coeff)
x_t=x
x_t=np.matrix(x_t)
y=np.matrix(y)
ver=coeff*x_t
print(ver)
print(y)
```

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

**Conclusion: In this concept, we expressed a given vector as a linear combination of other other vectors and verified if a vector can be expressed as a linear combination of other given vectors and tallied the results obtained.**

# Lab 9

## Topic: Linear Span, Linear Independence and Linear dependence

## Date: 17/10/22 to 22/10/22

**Aim: Building on the idea of linear vector combinations, this lab explores the ideas of linear independence, linear dependency, and linear span.**

**Source codes and output:**

**1. Let V={(-1,4),(3,-5)}. Check whether or not the vector (19,3) form a linear span**

In [100]:
```
v = np.array([[-1,3],[4,-5]])
print("V:" , v.T)

c= np.array([[19],[3]])
print("C:",c.T)

sol=np.linalg.solve(v,c)
print("\nThe solution of given system of equations is:\n", sol)

y=0
for i in sol:
      if np.isreal(i) == False:
              y=y+1
print()
if y==0:
      print(c.T, "belongs to the span(v)")
else:
      print(c.T, "doesn't belong to the span(v)")
```

```
V: [[-1  4]
 [ 3 -5]]
C: [[19  3]]

The solution of given system of equations is:
 [[14.85714286]
 [11.28571429]]

[[19  3]] belongs to the span(v)
```

**2. Write a program to check for Linear independence for vectors in $R^4$ as input by the user.**

In [101]:
```
A=[]
B=[]
C=[]
D=[]

for i in range(4):
    A.append(int(input("Enter a number into the first tuple:")))
    B.append(int(input("Enter a number into the second tuple:")))
    C.append(int(input("Enter a number into the third tuple:")))
    D.append(int(input("Enter a number into the fourth tuple:")))

A1=np.asarray(A)
B1=np.asarray(B)
C1=np.asarray(C)
D1=np.asarray(D)
st=np.vstack((A1, B1, C1,D1))
print("The obtained vector stack is:\n",st)

if np.linalg.det(st)==0:
    print("The given vectors are linearly independent")
else:
    print("The given vectors are not linearly independent")
```

```
Enter a number into the first tuple:2
Enter a number into the second tuple:1
Enter a number into the third tuple:1
Enter a number into the fourth tuple:1
Enter a number into the first tuple:1
Enter a number into the second tuple:1
Enter a number into the third tuple:1
Enter a number into the fourth tuple:1
Enter a number into the first tuple:1
Enter a number into the second tuple:1
Enter a number into the third tuple:1
Enter a number into the fourth tuple:1
Enter a number into the first tuple:1
Enter a number into the second tuple:1
Enter a number into the third tuple:1
Enter a number into the fourth tuple:1
The obtained vector stack is:
 [[2 1 1 1]
 [1 1 1 1]
 [1 1 1 1]
 [1 1 1 1]]
The given vectors are linearly independent
```

**3. If $\upsilon_1$ = (1,2,-1), $\upsilon_2$ = (2,-1,1), $\upsilon_3$ = (8,1,1). Show that $\upsilon_1$,$\upsilon_2$ and $\upsilon_3$ are linearly dependent in $R^3$.**

In [102]:
```
V=np.matrix([[1,2,-1],[2,-1,1],[8,1,1]])
display(V)
print("Determinant of matrix of vectors = ",round(np.linalg.det(V)))
print("Since the determinant of the matrix is zero, the given vectors are li
```

```
matrix([[ 1,  2, -1],
        [ 2, -1,  1],
        [ 8,  1,  1]])

Determinant of matrix of vectors =  0
Since the determinant of the matrix is zero, the given vectors are linearly dep
endent in R^3
```

**Conclusion: Through this lab, we explored the concepts of linear span and learnt to assess the linear independence and dependence of vectors.**

# Lab 10

## Topic: Linear transformations and plotting of linear transformations

## Date: 24/10/22 to 05/11/22

**Aim: The aim of this lab is to explore the concept of linear transformation of vectors using python and plot them using matplotlib package to visualize the change brought by the transformations.**

**Source codes and output:**

**Q. Are the following functions linear transformations?**

In [103]:
```
def T(u,v):
    return(u+2*v,u-v)
a=1
b=2
c=3
d=-1
p,q=T(a,b)
m,n=T(c,d)
c1=4
c2=-1
c11=c1*a+c2*c
c22=c1*b+c2*d
print("RHS: T[c1*u+c2*v]")
print(T(c11,c22))
print("LHS: c1*T[u]+c2*T[v]")
print(c1*p+c2*m,c1*q+c2*n)
```

```
RHS: T[c1*u+c2*v]
(19, -8)
LHS: c1*T[u]+c2*T[v]
19 -8
```

T is a linear transformation

In [104]:
```
def T(u,v):
    return(u,u**2)
a=1
b=2
c=3
d=-1
p,q=T(a,b)
m,n=T(c,d)
c1=4
c2=-1
c11=c1*a+c2*c
c22=c1*b+c2*d
print("RHS: T[c1*u+c2*v]")
print(T(c11,c22))
print("LHS: c1*T[u]+c2*T[v]")
print(c1*p+c2*m,c1*q+c2*n)
```

```
RHS: T[c1*u+c2*v]
(1, 1)
LHS: c1*T[u]+c2*T[v]
1 -5
```

T is not a linear transformation.

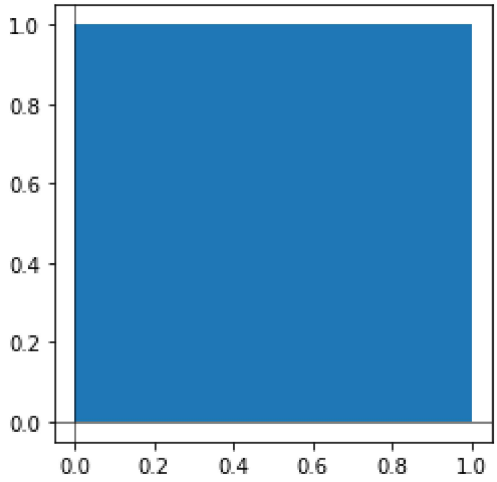
In [105]:

```
import numpy as np
u=np.mat(input("Enter first vector: "))
v=np.mat(input("Enter second vector: "))
c1=int(input("Enter first scalar: "))
c2=int(input("Enter second scalar: "))

t=c1*u+c2*v ##LHS

def T(x,y,z):
    X=np.mat([x+y,y+z,z+x])
    return X

lhs=T(t[0,0],t[0,1],t[0,2])
rhs=c1*T(u[0,0],u[0,1],u[0,2])+c2*T(v[0,0],v[0,1],v[0,2])

if lhs.all==rhs.all:
    print("T is a LT")
else:
    print("T is not a LT")
```

```
Enter first vector: 1,2,3
Enter second vector: 12,11,10
Enter first scalar: 1
Enter second scalar: 2
T is not a LT
```

**If T is a linear transformation such that T(1,0)=(1,1) and T(0,1)=(-1,2). Show that T maps the square with vertices (0,0),(1,0),(0,1),(1,1) into parallelogram**

In [106]:
```
import numpy as np
import matplotlib.pyplot as plt

x=np.linspace(-10,10)
Square= plt.Rectangle((0,0), 1, 1)
plt.gca().add_patch(Square)
plt.axis('scaled')
plt.axhline(0, lw=0.5, color='black')
plt.axvline(0, lw=0.5, color='black')

plt.show()
```

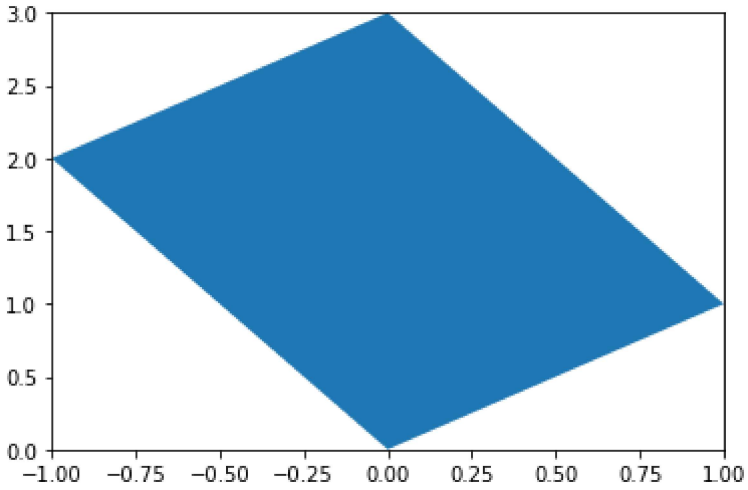



In [107]:
```
## Finding out T

def T(x,y):
    X=np.mat([x-y,x+2*y])
    return X
print(T(0,0))
print(T(1,0))
print(T(1,1))
print(T(0,1))
```

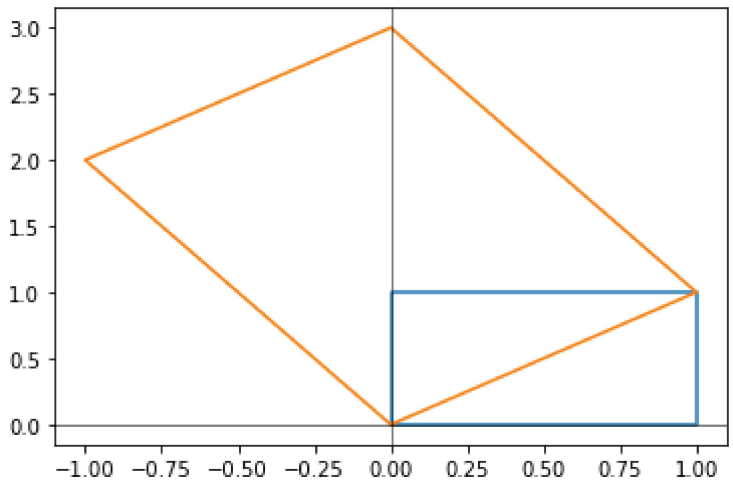
```
[[0 0]]
[[1 1]]
[[0 3]]
[[-1  2]]
```

In [108]:

```
## Plotting transformation

from matplotlib.patches import Polygon
pts = np.array([[0,0], [1,1], [0,3],[-1,2]])
p = Polygon(pts, closed=False)
ax = plt.gca()
ax.add_patch(p)
ax.set_xlim(-1,1)
ax.set_ylim(0,3)
plt.show()
```



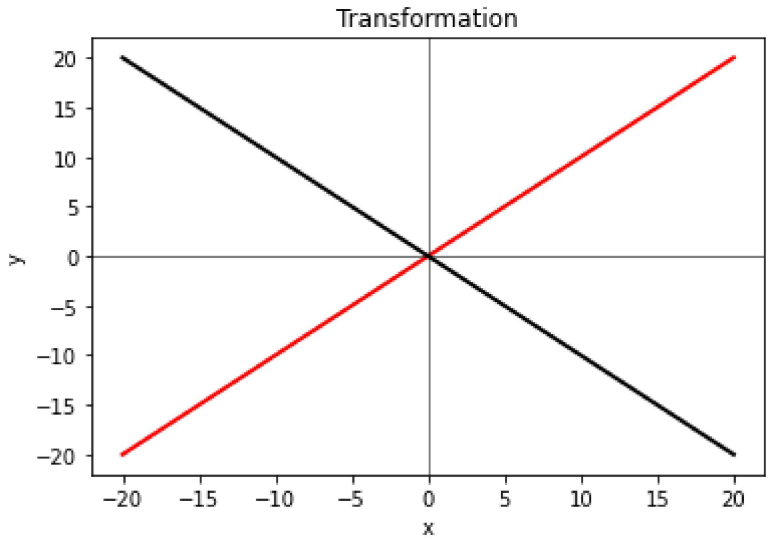
In [109]:
```
x=np.array([0,1,1,0,0])
y=np.array([0,0,1,1,0])
plt.plot(x,y)

x1=np.array([0,1,0,-1,0])
y1=np.array([0,1,3,2,0])

plt.plot(x1,y1)
plt.axhline(0, lw=0.5, color='black')
plt.axvline(0, lw=0.5, color='black')

plt.show()
```



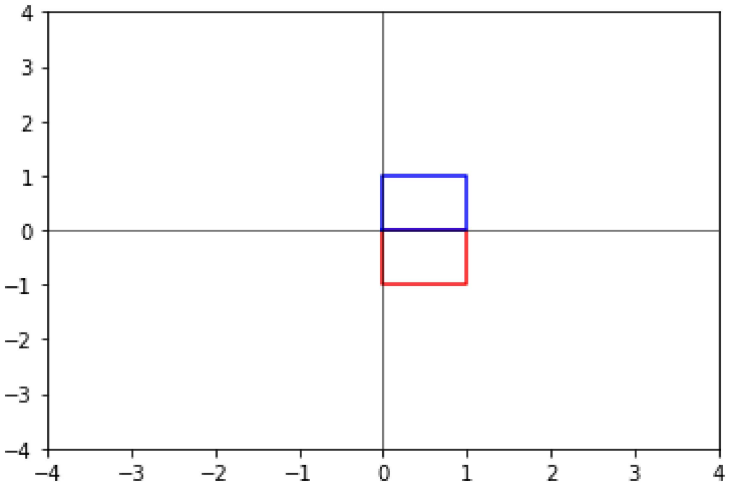

**Geometrically show the LT of T[x,y]=[x,-y]**

In [110]:

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

x=np.linspace(-20, 20)

plt.plot(x,x, lw=2, color='red', label='[x,y]')
plt.plot(x,-x, lw=2, color='black', label='[x,-y]')

plt.title('Transformation')
plt.xlabel('x')
plt.ylabel('y')
plt.axhline(0, lw=0.5, color='black')
plt.axvline(0, lw=0.5, color='black')
None
```

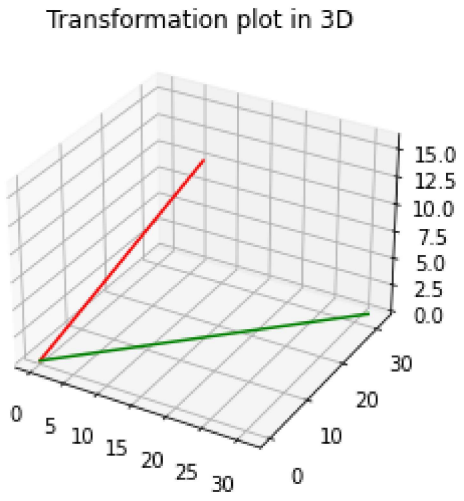


In [111]:
```
x=np.array([0,1,1,0,0])
y=np.array([0,0,-1,-1,0])
plt.xlim(-4,4)
plt.ylim(-4,4)
plt.plot(x,y,color='red')

x1=np.array([0,1,1,0,0])
x2=np.array([0,0,1,1,0])
plt.xlim(-4,4)
plt.ylim(-4,4)
plt.plot(x1,x2,color='blue')

plt.axhline(0, lw=0.5, color='black')
plt.axvline(0, lw=0.5, color='black')

plt.show()
```



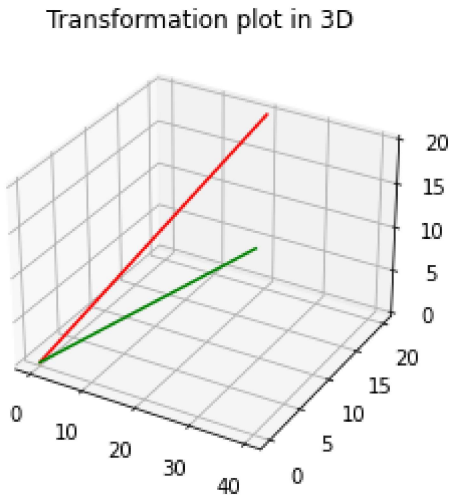

**Geometrically transform: T(x,y,z)=[x+y,y+z,y-z]**

In [112]:
```
fig=plt.figure()
ax=plt.axes(projection='3d')

z=np.linspace(0,16)
x=np.linspace(0,15)
y=np.linspace(0,16)

ax.plot3D(x,y,z,'red')
ax.plot3D(x+y,y+z,y-z, 'green')
ax.set_title('Transformation plot in 3D')
plt.show()
```




**Geometrically transform: T(x,y,z)=[x+y,x-y,z]**

In [113]:
```
fig=plt.figure()
ax=plt.axes(projection='3d')

z=np.linspace(0,20)
x=np.linspace(0,20)
y=np.linspace(0,20)

ax.plot3D(x,y,z,'red')
ax.plot3D(x+y,x-y,z, 'green')
ax.set_title('Transformation plot in 3D')
plt.show()
```




**Conclusion: In this lab, we learnt to check if a function is a linear transformation and using 2D and 3D plotting to visualize linear transformations.**

# Lab 11

## Topic: Applications of Rank and Nullity

## Date: 07/11/22 to 19/11/22

**Aim: The lab was designed to verify the Rank-Nullity theorem by diving into the concepts of Rank and Nullity of a Vector Space.**

**Source codes and output:**

**Q. Given the matrix A, find the spanning set for the null space of A.**

In [114]:
```
A=Matrix(np.matrix([[1,2,1,4],[0,2,8,10],[1,4,9,14],[-1,0,7,6]]))
display(A)
```

$$\begin{bmatrix} 1 & 2 & 1 & 4 \\ 0 & 2 & 8 & 10 \\ 1 & 4 & 9 & 14 \\ -1 & 0 & 7 & 6 \end{bmatrix}$$

In [115]:
```
A.nullspace()
```

Out[115]:
```
[Matrix([
 [ 7],
 [-4],
 [ 1],
 [ 0]]),
 Matrix([
 [ 6],
 [-5],
 [ 0],
 [ 1]])]
```

**Q. Find the nullity of the given matrix**

In [116]:
```
from sympy import Matrix
A = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
A = Matrix(A)
n = A.shape[1]
rank = A.rank()
nullity = n - rank
print("Nullity : ", nullity)
```

```
Nullity :  1
```

***Conclusion: From the lab above, we can verify the rank-nullity theorem and understand the concepts of Rank and Null matrix.***

# CONCLUSION OF THE LAB

***We used python to study various concepts of Linear Algebra such as eigen values, eigen vectors, CHT Theorem, Solving Equations using various methods, Transformations, Plotting, Rank-Nullity Theorem etc.***