In [2]:

```
from sympy.interactive import printing
printing.init_printing(use_latex=True)
from sympy import Eq,solve_linear_system,Matrix,det,transpose,Transpose,symbols,trace,Trace
from sympy.solvers import solve
from sympy import simplify,sqrt
from sympy.physics.quantum import Bra,Ket
from time import time
import numpy as np
import sympy as sp

print('________________________----Welcome to Linear Algebra Calculator Material----________________________')
print('\n\n\n')
print('1 -- UNIT-I   -- SOLVING SYSTEM OF LINEAR EQUATIONS')
print('2 -- UNIT-II  -- LINEAR DEPENDANCY AND BASIS FORMATION CHECKING')
print('3 -- UNIT-III -- FINDING EIGEN VALUE AND EIGEN VECTOR')
print('4 -- UNIT-IV  -- ORTHOGONALIZATION')
print('5 -- UNIT-V   -- QR DECOMPOSITION')

inum=int(input('Enter Your Choice of Chapter : '))
if inum==1:
    print("-------------Welcome! To Solving system Of Linear Equations---------------")


    print("2.2X2 Matrix")
    print("3.3X3 Matrix")
    print("4.4X4 Matrix")

    u=int(input("which nXn matrix you want : "))
    display("Using Sympy Module")
    if u==2:
        display("---------------------------------------------------------------------")
        eq1=sp.Function("eq1")
        eq2=sp.Function("eq2")
        x,y=sp.symbols("x y")
        x1=int(input("Enter your X1 coefficient : "))
        y1=int(input("Enter your Y1 coefficient : "))
        display("---------------------------------------------------------------------")
        x2=int(input("Enter your X2 coefficient : "))
        y2=int(input("Enter your Y2 coefficient : "))
        display("---------------------------------------------------------------------")
        c1=int(input("Enter your constant 1 : "))
        c2=int(input("Enter your constant 2 : "))
        display("---------------------------------------------------------------------")

        eq1=sp.Function("eq1")
```

```
    eq2=sp.Function("eq2")
    x,y=sp.symbols('x y')

    eq1=Eq(x1*x+y1*y,c1)
    eq2=Eq(x2*x+y2*y,c2)

    display(eq1)
    display(eq2)

    row1=[x1,y1,c1]
    row2=[x2,y2,c2]
    system=Matrix((row1,row2))
    display(system)
    start=time()
    if(solve_linear_system(system,x,y)==None):
        display("System is Inconsistent")
    else:
        display("System is Consistent")
        display(solve_linear_system(system,x,y))
    end=time()
    display("Execution Time : ",end-start)


if u==3:
    display("--------------------------------------------------------------------")

    x1=int(input("Enter your X1 coefficient : "))
    y1=int(input("Enter your Y1 coefficient : "))
    z1=int(input("Enter your z1 coefficient : "))

    display("--------------------------------------------------------------------")

    x2=int(input("Enter your X2 coefficient : "))
    y2=int(input("Enter your Y2 coefficient : "))
    z2=int(input("Enter your z2 coefficient : "))

    display("--------------------------------------------------------------------")

    x3=int(input("Enter your X3 coefficient : "))
    y3=int(input("Enter your Y3 coefficient : "))
    z3=int(input("Enter your z3 coefficient : "))

    display("--------------------------------------------------------------------")

    c1=int(input("Enter your constant 1 : "))
    c2=int(input("Enter your constant 2 : "))
    c3=int(input("Enter your constant 3 : "))
```

```
    display("--------------------------------------------------------------------")

    eq1=sp.Function("eq1")
    eq2=sp.Function("eq2")
    eq3=sp.Function("eq3")

    x,y,z=sp.symbols('x y z')

    eq1=Eq(x1*x+y1*y+z1*z,c1)
    eq2=Eq(x2*x+y2*y+z2*z,c2)
    eq3=Eq(x3*x+y3*y+z3*z,c3)

    display(eq1)
    display(eq2)
    display(eq3)

    row1=[x1,y1,z1,c1]
    row2=[x2,y2,z2,c2]
    row3=[x3,y3,z3,c3]

    system=Matrix((row1,row2,row3))
    display(system)
    start=time()
    if(solve_linear_system(system,x,y,z)==None):
        display("System is Inconsistent")
    else:
        display("System is Consistent")
        display(solve_linear_system(system,x,y,z))
    end=time()
    display("Execution Time : ",end-start)

if u==4:
    display("--------------------------------------------------------------------")
    w1=int(input("Enter your w1 coefficient : "))
    x1=int(input("Enter your X1 coefficient : "))
    y1=int(input("Enter your Y1 coefficient : "))
    z1=int(input("Enter your z1 coefficient : "))
    display("--------------------------------------------------------------------")
    w2=int(input("Enter your w2 coefficient : "))
    x2=int(input("Enter your X2 coefficient : "))
    y2=int(input("Enter your Y2 coefficient : "))
    z2=int(input("Enter your z2 coefficient : "))
    display("--------------------------------------------------------------------")
    w3=int(input("Enter your w3 coefficient : "))
    x3=int(input("Enter your X3 coefficient : "))
    y3=int(input("Enter your Y3 coefficient : "))
```

```
    z3=int(input("Enter your z3 coefficient : "))
    display("--------------------------------------------------------------------")

    w4=int(input("Enter your w4 coefficient : "))
    x4=int(input("Enter your X4 coefficient : "))
    y4=int(input("Enter your Y4 coefficient : "))
    z4=int(input("Enter your z4 coefficient : "))

    display("--------------------------------------------------------------------")

    c1=int(input("Enter your constant 1 : "))
    c2=int(input("Enter your constant 2 : "))
    c3=int(input("Enter your constant 3 : "))
    c4=int(input("Enter your constant 4 : "))

    display("--------------------------------------------------------------------")

    eq1=sp.Function("eq1")
    eq2=sp.Function("eq2")
    eq3=sp.Function("eq3")
    eq4=sp.Function("eq4")
    w,x,y,z=sp.symbols('w x y z')


    eq1=Eq(w1*w+x1*x+y1*y+z1*z,c1)
    eq2=Eq(w2*w+x2*x+y2*y+z2*z,c2)
    eq3=Eq(w3*w+x3*x+y3*y+z3*z,c3)
    eq4=Eq(w4*w+x4*x+y4*y+z4*z,c4)


    display(eq1)
    display(eq2)
    display(eq3)
    display(eq4)

    row1=[w1,x1,y1,z1,c1]
    row2=[w2,x2,y2,z2,c2]
    row3=[w3,x3,y3,z3,c3]
    row4=[w4,x4,y4,z4,c4]


    system=Matrix((row1,row2,row3,row4))
    display(system)
    start=time()
    if(solve_linear_system(system,w,x,y,z)==None):
        display("System is Inconsistent")
    else:
```

```
            display("System is Consistent")
            display(solve_linear_system(system,w,x,y,z))

        end=time()
        display("Execution Time : ",end-start)



    print("Using Numpy module")
    print("--------------------------------------------------------------------")
    if u==2:
        nrow1=[x1,y1]
        nrow2=[x2,y2]

        nmat=np.array([nrow1,nrow2])
        cons=np.array([c1,c2])
        display(nmat,cons)

        start1=time()
        answer=linalg.solve(nmat,cons)
        end1=time()
        xval=answer[0]
        yval=answer[1]
        print("--------------------------------------------------------------------")

        display("x= ",float(xval))
        display("y= ",float(yval))
        display("--------------------------------------------------------------------")
        display("Execution time : ",end1-start1)
        display("--------------------------------------------------------------------")

    if u==3:
        nrow1=[x1,y1,z1]
        nrow2=[x2,y2,z2]
        nrow3=[x3,y3,z3]

        nmat=np.array([nrow1,nrow2,nrow3])
        cons=np.array([c1,c2,c3])
        display(nmat,cons)

        start1=time()
        answer=linalg.solve(nmat,cons)
        end1=time()
        xval=answer[0]
        yval=answer[1]
        zval=answer[2]
        print("--------------------------------------------------------------------")
```

```
        display("x= ",float(xval))
        display("y= ",float(yval))
        display("z= ",float(zval))

        print("--------------------------------------------------------------------")
        display("Execution time : ",end1-start1)
        print("--------------------------------------------------------------------")

    if u==4:
        nrow1=[w1,x1,y1,z1]
        nrow2=[w2,x2,y2,z2]
        nrow3=[w3,x3,y3,z3]
        nrow4=[w4,x4,y4,z4]

        nmat=np.array([nrow1,nrow2,nrow3,nrow4])
        cons=np.array([c1,c2,c3,c4])
        display(nmat,cons)

        start1=time()
        answer=linalg.solve(nmat,cons)
        end1=time()
        wval=answer[0]
        xval=answer[1]
        yval=answer[2]
        zval=answer[3]
        print("--------------------------------------------------------------------")

        display("w= ",int(wval))
        display("x= ",int(xval))
        display("y= ",int(yval))
        display("z= ",int(zval))

        print("--------------------------------------------------------------------")
        display("Execution time : ",end1-start1)
        print("--------------------------------------------------------------------")
elif inum==2:
    print("------------- Welcome! To Find linear combination dependancy and basis formation checking ---------------")
    no=int(input("No of Times you going to execute the program : "))
    for j in range(0,no):
        n=int(input('Number of vectors : '))
        m,a,e,g=sp.symbols('* + = -->')
        if n==2:
                x,y,m,a,e,g=sp.symbols("x1 x2 * + = -->")
                a11,a12=map(int,input("Enter 1st row : ").split())
                a21,a22=map(int,input("Enter 2nd row : ").split())
                vect1,vect2=[a11,a12],[a21,a22]
```

```
        amat=Matrix((vect1,vect2))
        print("Given Matrix")
        display(amat)
        print('now')
        display(((x,m,vect1),a,(y,m,vect2),e,(0,0),g))
        r1,r2=[a11,a21,0],[a12,a22,0]

        eq1,eq2,eqn,eqm=sp.Function("eq1"),sp.Function("eq2"),sp.Function("eqn"),sp.Function("eqm")
        eqm=(a11*x,a12*x)
        eqn=(a21*y,a22*y)
        display((eqm,a,eqn,e,(0,0)))
        eq1,eq2=Eq(a11*x+a21*y,0),Eq(a12*x+a22*y,0)
        display(eq1,eq2)
        print("arranged matrix")
        display((transpose(amat),Matrix((x,y)),e,Matrix((0,0))))
        system=Matrix((r1,r2))
        print("augmented form")
        display(system)

        print("\n\n\nThe two vectors are ")
        if(det(amat)==0):
            print("linearly dependant")
            display(solve_linear_system(system,x,y))
            print("dim R2 != 2, The given vectors NOT form a basis of R^2")
        else:
            print("linearly independant")
            display(solve_linear_system(system,x,y))
            print("The given vectors does form a basis of R^2")
if n==3:
        x,y,z=sp.symbols("x1 x2 x3")
        a11,a12,a13=map(int,input("Enter 1st row : ").split())
        a21,a22,a23=map(int,input("Enter 2nd row : ").split())
        a31,a32,a33=map(int,input("Enter 3rd row : ").split())
        vect1,vect2,vect3=[a11,a12,a13],[a21,a22,a23],[a31,a32,a33]
        amat=Matrix((vect1,vect2,vect3))
        print("Given Matrix")
        display(amat)
        print('now')
        display(((x,m,vect1),a,(y,m,vect2),a,(z,m,vect3),e,(0,0,0),g))
        r1,r2,r3=[a11,a21,a31,0],[a12,a22,a32,0],[a13,a23,a33,0]
        eq1,eq2,eq3,eqn,eqm,eqo=sp.Function("eq1"),sp.Function("eq2"),sp.Function("eq3"),sp.Function("eqn"),sp.Function("eqm"),sp.Fur
        eqm=(a11*x,a12*x,a13*x)
        eqn=(a21*y,a22*y,a23*y)
        eqo=(a31*z,a32*z,a33*z)
        display((eqm,a,eqn,a,eqo,e,(0,0,0),g))
        eq1,eq2,eq3=Eq(a11*x+a21*y+a31*z,0),Eq(a12*x+a22*y+a32*z,0),Eq(a13*x+a23*y+a33*z,0)
        display(eq1,eq2,eq3)
```

```
        print("arranged matrix")
        display((transpose(amat),Matrix((x,y,z)),e,Matrix((0,0,0))))
        system=Matrix((r1,r2,r3))
        print("augmented form")
        display(system)

        print("\n\n\nThe two vectors are ")
        if(det(transpose(amat))==0):
            print("linearly dependant")
            display(solve_linear_system(system,x,y,z))
            print("dim R3 != 3, The given vectors NOT form a basis of R^3")
        else:
            print("linearly independant")
            display(solve_linear_system(system,x,y,z))
            print("The given vectors does form a basis of R^3")
if n==4:
        x,y,z,w=sp.symbols("x1 x2 x3 x4")
        a11,a12,a13,a14=map(int,input("Enter 1st row : ").split())
        a21,a22,a23,a24=map(int,input("Enter 2nd row : ").split())
        a31,a32,a33,a34=map(int,input("Enter 3rd row : ").split())
        a41,a42,a43,a44=map(int,input("Enter 4th row : ").split())
        vect1,vect2,vect3,vect4=[a11,a12,a13,a14],[a21,a22,a23,a24],[a31,a32,a33,a34],[a41,a42,a43,a44]
        amat=Matrix((vect1,vect2,vect3,vect4))
        print("Given Matrix")
        display(amat)
        print('now')
        display(((x,m,vect1),a,(y,m,vect2),a,(z,m,vect3),a,(w,m,vect4),e,(0,0,0,0),g))
        r1,r2,r3,r4=[a11,a21,a31,a41,0],[a12,a22,a32,a42,0],[a13,a23,a33,a43,0],[a14,a24,a34,a44,0]
        eq1,eq2,eq3,eq4=sp.Function("eq1"),sp.Function("eq2"),sp.Function("eq3"),sp.Function("eq4")
        eqm=(a11*x,a12*x,a13*x,a14*x)
        eqn=(a21*y,a22*y,a23*y,a24*y)
        eqo=(a31*z,a32*z,a33*z,a34*z)
        eqp=(a41*w,a42*w,a43*w,a44*w)
        display((eqm,a,eqn,a,eqo,a,eqp,e,(0,0,0,0),g))
        eq1,eq2,eq3,eq4=Eq(a11*x+a21*y+a31*z+a41*w,0),Eq(a12*x+a22*y+a32*z+a42*w,0),Eq(a13*x+a23*y+a33*z+a43*w,0),Eq(a14*x+a24*y+a34*
        display(eq1,eq2,eq3,eq4)
        system=Matrix((r1,r2,r3,r4))
        print('rearranged matrix')
        display((transpose(amat),Matrix((x,y,z,w)),e,Matrix((0,0,0,0))))
        print("augmented form")
        display(system)

        print("\n\n\nThe two vectors are ")
        if(det(amat)==0):
            print("linearly dependant")
            display(solve_linear_system(system,x,y,z,w))
            print("dim R4 != 4, The given vectors NOT form a basis of R^4")
```

```
                else:
                    print("linearly independant")
                    display(solve_linear_system(system,x,y,z,w))
                    print("The given vectors form a basis of R^4")
        print('_______________________________________________')
    elif inum==3:
        a11,a12,a13,a21,a22,a23,a31,a32,a33=map(int,input("Enter the matrix coefficients : ").split())
        r1=[a11,a12,a13]
        r2=[a21,a22,a23]
        r3=[a31,a32,a33]
        amat=sp.Matrix((r1,r2,r3))
        print("Input matrix")
        display(amat)
        s1=a11+a22+a33
        display('S1={0}+{1}+{2}'.format(a11,a22,a33))
        display('s1={0}'.format(s1))
        s2=(((a22*a33)-(a32*a23))+((a11*a33)-(a31*a13))+((a11*a22)-(a21*a12)))
        display('s2=((({0}*{1})-({2}*{3}))+(({4}*{5})-({6}*{7}))+(({8}*{9})-({10}*{11}))'.format(a22,a33,a32,a23,a11,a33,a31,a13,a11,a22,a21,a12)
        display('s2=({0}+{1}+{2})'.format(((a22*a33)-(a32*a23)),((a11*a33)-(a31*a13)),((a11*a22)-(a21*a12))))
        s3=det(amat)
        display('s3=det(amat)')
        display("s3={0}".format(s3))
        display('s1={0} s2={1} s3={2}'.format(s1,s2,s3))
        x=sp.symbols('λ')
        eq1=sp.Function("eq1")
        eq1=Eq((x**3 - (s1)*x**2 +(s2)*x- s3))
        display(eq1)
        k=sp.factor(eq1)
        display(k)
        K=solve(k)
        print("Eigen values are : ",K)
        a,b,c,d,e,f=sp.symbols('a b c * = ->')
        b1=[a11-x,a12,a13]
        b2=[a21,a22-x,a23]
        b3=[a31,a32,a33-x]
        bmat=sp.Matrix((b1,b2,b3))
        xmat=sp.Matrix((a,b,c))
        print("To find eigen vector")
        display(bmat,d,xmat)
        cmat=sp.Matrix((0,0,0))
        if(len(K)==2):
            print("If λ={0}".format(K[0]))
            b1=[a11-K[0],a12,a13]
            b2=[a21,a22-K[0],a23]
            b3=[a31,a32,a33-K[0]]
            bmat=sp.Matrix((b1,b2,b3))
            display((bmat,d,xmat,e,cmat,f))
```

```
        bm1=[a11-K[0],a12,a13,0]
        bm2=[a21,a22-K[0],a23,0]
        bm3=[a31,a32,a33-K[0],0]
        eq2=sp.Function("eq2")
        eq3=sp.Function("eq3")
        eq4=sp.Function("eq4")
        eq2=Eq((a11-K[0])*a+a12*b+a13*c)
        eq3=Eq(a21*a+(a22-K[0])*b+a23*c)
        eq4=Eq(a31*a+a32*b+(a33-K[0])*c)
        display(eq2,eq3,eq4)
        system=Matrix((bm1,bm2,bm3))
        n=sp.solve_linear_system(system,a,b,c)
        display(n)

        print("If λ={0}".format(K[1]))
        b1=[a11-K[1],a12,a13]
        b2=[a21,a22-K[1],a23]
        b3=[a31,a32,a33-K[1]]
        bmat=sp.Matrix((b1,b2,b3))
        display((bmat,d,xmat,e,cmat,f))
        bm1=[a11-K[1],a12,a13,0]
        bm2=[a21,a22-K[1],a23,0]
        bm3=[a31,a32,a33-K[1],0]
        eq2=sp.Function("eq2")
        eq3=sp.Function("eq3")
        eq4=sp.Function("eq4")
        eq2=Eq((a11-K[1])*a+a12*b+a13*c)
        eq3=Eq(a21*a+(a22-K[1])*b+a23*c)
        eq4=Eq(a31*a+a32*b+(a33-K[1])*c)
        display(eq2,eq3,eq4)
        system=Matrix((bm1,bm2,bm3))
        n=sp.solve_linear_system(system,a,b,c)
        display(n)

        print("Eigen vector")
        j=amat.eigenvects()
        display(j)

    if(len(K)==3):
        print("If λ={0}".format(K[0]))
        b1=[a11-K[0],a12,a13]
        b2=[a21,a22-K[0],a23]
        b3=[a31,a32,a33-K[0]]
        bmat=sp.Matrix((b1,b2,b3))
        display((bmat,d,xmat,e,cmat,f))
        bm1=[a11-K[0],a12,a13,0]
        bm2=[a21,a22-K[0],a23,0]
```

```
bm3=[a31,a32,a33-K[0],0]
eq2=sp.Function("eq2")
eq3=sp.Function("eq3")
eq4=sp.Function("eq4")
eq2=Eq((a11-K[0])*a+a12*b+a13*c)
eq3=Eq(a21*a+(a22-K[0])*b+a23*c)
eq4=Eq(a31*a+a32*b+(a33-K[0])*c)
display(eq2,eq3,eq4)
system=Matrix((bm1,bm2,bm3))
n=sp.solve_linear_system(system,a,b,c)
display(n)
print("If λ={0}".format(K[1]))
b1=[a11-K[1],a12,a13]
b2=[a21,a22-K[1],a23]
b3=[a31,a32,a33-K[1]]
bmat=sp.Matrix((b1,b2,b3))
display((bmat,d,xmat,e,cmat,f))
bm1=[a11-K[1],a12,a13,0]
bm2=[a21,a22-K[1],a23,0]
bm3=[a31,a32,a33-K[1],0]
eq2=sp.Function("eq2")
eq3=sp.Function("eq3")
eq4=sp.Function("eq4")
eq2=Eq((a11-K[1])*a+a12*b+a13*c)
eq3=Eq(a21*a+(a22-K[1])*b+a23*c)
eq4=Eq(a31*a+a32*b+(a33-K[1])*c)
display(eq2,eq3,eq4)
system=Matrix((bm1,bm2,bm3))
n=sp.solve_linear_system(system,a,b,c)
display(n)
print("If λ={0}".format(K[2]))
b1=[a11-K[2],a12,a13]
b2=[a21,a22-K[2],a23]
b3=[a31,a32,a33-K[2]]
bmat=sp.Matrix((b1,b2,b3))
display((bmat,d,xmat,e,cmat,f))
bm1=[a11-K[2],a12,a13,0]
bm2=[a21,a22-K[2],a23,0]
bm3=[a31,a32,a33-K[2],0]
eq2=sp.Function("eq2")
eq3=sp.Function("eq3")
eq4=sp.Function("eq4")
eq2=Eq((a11-K[2])*a+a12*b+a13*c)
eq3=Eq(a21*a+(a22-K[2])*b+a23*c)
eq4=Eq(a31*a+a32*b+(a33-K[2])*c)
display(eq2,eq3,eq4)
system=Matrix((bm1,bm2,bm3))
```

```
        n=sp.solve_linear_system(system,a,b,c)
        display(n)

        print("Eigen vector")

        j=amat.eigenvects()
        display(j)
elif inum==4:
    print('___________________----Welcome to Eigen value and Eigen vector Calculation----___________________')
    print('1. Vector method\n2. Matrix method')
    ornum=int(input('Enter method to do Orthogonalization'))
    if ornum==1:
        mv1,mv2,mv3,eq=sp.symbols('||V1|| ||V2|| ||V3|| =')
        su1,su2,su3,sv1,sv2,sv3=sp.symbols('U1 U2 U3 V1 V2 V3')
        bu1,bu2,bu3=Bra('U1'),Bra('U2'),Bra('U3')
        bv1,bv2,bv3=Bra('V1'),Bra('V2'),Bra('V3')
        ku1,ku2,ku3=Ket('U1'),Ket('U2'),Ket('U3')
        kv1,kv2,kv3=Ket('V1'),Ket('V2'),Ket('V3')

        u1=Matrix(list(map(int,input('Vector 1 :').split())))
        u2=Matrix(list(map(int,input('Vector 2 :').split())))
        u3=Matrix(list(map(int,input('Vector 3 :').split())))

        print('Let the given vectors be {u1,u2,u3}')
        display((u1,u2,u3))

        eq1=Eq(su1,sv1)
        display(eq1)
        display((sv1,eq,u1))
        v1=u1
        print('For our convinience we first find self inner products of v1')
        v1s=v1.dot(v1)
        ev1s=Bra(v1)*Ket(v1)
        display((mv1**2,eq,bv1*kv1,eq,ev1s,eq,v1s))
        eq2=(((su2-((bu2*kv1)/(mv1**2))*su1)))
        display((sv2,eq,eq2))
        u2v1=u2.dot(v1)
        display((bu2*kv1,eq,Bra(u2)*Ket(v1),eq,u2v1))
        div1=u2v1/v1s
        v2=u2-(div1*v1)
        display((sv2,eq,v2))
        v2s=v2.dot(v2)
        display((bv2*kv2,eq,Bra(v2)*Ket(v2),eq,v2s))
        eq3=((su3-(((bu3*kv1/(mv1**2))*sv1)-((bu3*kv2)/(mv2**2))*sv2)))
        display((sv3,eq,eq3))
        u3v1=u3.dot(v1)
```

```
        u3v2=u3.dot(v2)
        display((bu3*kv1,eq,Bra(u3)*Ket(v1),eq,u3v1))
        display((bu3*kv2,eq,Bra(u3)*Ket(v2),eq,u3v2))
        v2s=v2.dot(v2)
        ev2s=Bra(v2)*Ket(v2)
        display((mv2**2,eq,bv2*kv2,eq,ev2s,eq,v2s))
        div2=u3v1/v1s
        div3=u2v1/v2s
        v3=u3-(div2*v1)-(div3*v2)
        display((sv3,eq,v3))
        v3s=v3.dot(v3)
        ev3s=Bra(v3)*Ket(v3)
        display((mv3**2,eq,bv3*kv3,eq,ev3s,eq,v3s))
        print('Orthogonal Vectors We Got')
        display((v1,v2,v3))
        print('Orthonormal Basis')

        w1,w2,w3,mul,div=sp.symbols('w1 w2 w3 * /')
        display((w1,eq,(sv1/mv1)))
        display((w2,eq,(sv2/mv2)))
        display((w3,eq,(sv3/mv3)))
        W1=v1/sqrt(v1s)
        W2=v2/sqrt(v2s)
        W3=v3/sqrt(v3s)
        display((w1,eq,v1,div,sqrt(v1s)))
        display((w1,eq,W1))
        display((w2,eq,v2,div,sqrt(v2s)))
        display((w2,eq,W2))
        display((w3,eq,v3,div,sqrt(v3s)))
        display((w3,eq,W3))
        print('The obtained Orthonormal Basis is ')
        display((W1,W2,W3))
        quit()
    elif ornum==2:
        mv1,mv2,mv3,eq=sp.symbols('||V1|| ||V2|| ||V3|| =')
        su1,su2,su3,sv1,sv2,sv3=sp.symbols('U1 U2 U3 V1 V2 V3')
        bv1,bv2,bv3=Bra('V1'),Bra('V2'),Bra('V3')
        bu1,bu2,bu3=Bra('U1'),Bra('U2'),Bra('U3')
        ku1,ku2,ku3=Ket('U1'),Ket('U2'),Ket('U3')
        kv1,kv2,kv3=Ket('V1'),Ket('V2'),Ket('V3')
        w1,w2,w3,mul,div=sp.symbols('w1 w2 w3 * /')

        a11,a12,a21,a22=map(int,input('Enter the matrix elements of A : ').split())
        b11,b12,b21,b22=map(int,input('Enter the matrix elements of A : ').split())
        c11,c12,c21,c22=map(int,input('Enter the matrix elements of A : ').split())
        ar1,ar2=[a11,a12],[a21,a22]
        br1,br2=[b11,b12],[b21,b22]
```

```
cr1,cr2=[c11,c12],[c21,c22]
print('Let u1 , u2 and u3 be given basis')
matu1=Matrix((ar1,ar2))
display(matu1)
matu2=Matrix((br1,br2))
display(matu2)
matu3=Matrix((cr1,cr2))
display(matu3)
matv1=matu1
display((sv1,eq,su1))
display((sv1,eq,matv1))
mev1s=Bra(Transpose(matv1))*Ket(Transpose(Transpose(matv1)))
mv1s=trace(Transpose(matv1)*matv1)
display((mv1**2,eq,bv1*kv1,eq,mev1s,eq,Trace(Transpose(matv1)*matv1),eq,mv1s))
eq2=Eq(((su2-((bu2*kv1)/(mv1**2))*su1)),sv2)
display(eq2)
mu2mv1=trace(Transpose(matu2)*matv1)
display((bu2*kv1,eq,Bra(Transpose(matv1))*Ket(Transpose(Transpose(matu2))),eq,Trace(Transpose(matv1)*matu2),eq,mu2mv1))
mdiv1=mu2mv1/mv1s
matv2=matu2-(matu1*mdiv1)
display((sv2,eq,matv2))
mev2s=Bra(Transpose(matv2))*Ket(Transpose(Transpose(matv2)))
mv2s=trace(Transpose(matv2)*matv2)
display((mv2**2,eq,bv2*kv2,eq,mev2s,eq,Trace(Transpose(matv2)*matv2),eq,mv2s))
eq3=((su3-(((bu3*kv1)/(mv1**2))*sv1)-((bu3*kv2)/(mv2**2))*sv2))
display((eq3,eq,sv3))
mu3mv1=trace(Transpose(matv1)*matu3)
mu3mv2=trace(Transpose(matv2)*matu3)
display((bu3*kv1,eq,Bra(Transpose(matv1))*Ket(Transpose(Transpose(matu3))),eq,Trace(Transpose(matv1)*matu3),eq,mu3mv1))
display((bu3*kv2,eq,Bra(Transpose(matv2))*Ket(Transpose(Transpose(matu3))),eq,Trace(Transpose(matv2)*matu3),eq,mu3mv2))
mdiv2=mu3mv1/mv1s
mdiv3=mu3mv2/mv2s
matv3=matu3-(mdiv2*matv1)-(mdiv3*matv2)
display((sv3,eq,matv3))
mev3s=Bra(Transpose(matv3))*Ket(Transpose(Transpose(matv3)))
mv3s=trace(transpose(matv3)*matv3)
display((mv3**2,eq,bv3*kv3,eq,mev3s,eq,Trace(Transpose(matv3)*matv3),eq,mv3s))
print('Orthogonal basis We Got')
display((matv1,matv2,matv3))
print('Orthonormal Basis')
display((w1,eq,(sv1/mv1)))
display((w2,eq,(sv2/mv2)))
display((w3,eq,(sv3/mv3)))
matW1=matv1/sqrt(mv1s)
matW2=matv2/sqrt(mv2s)
matW3=matv3/sqrt(mv3s)
display((w1,eq,matv1,div,sqrt(mv1s)))
```

```
        display((w1,eq,matW1))
        display((w2,eq,matv2,div,sqrt(mv2s)))
        display((w2,eq,matW2))
        display((w3,eq,matv3,div,sqrt(mv3s)))
        display((w3,eq,matW3))
        print('The obtained Orthonormal Basis is ')
        display((matW1,matW2,matW3))
        quit()
elif inum==5:
    print('_______________________----Welcome To Calculate QR decomposition----_______________________')
    mv1,mv2,mv3,eq=sp.symbols('||u1|| ||u2|| ||u3|| =')
    su1,su2,su3,sv1,sv2,sv3=sp.symbols('a1 a2 a3 u1 u2 u3')
    bv1,bv2,bv3=Bra('u1'),Bra('u2'),Bra('u3')
    ku1,ku2,ku3=Ket('a1'),Ket('a2'),Ket('a3')
    kv1,kv2,kv3=Ket('u1'),Ket('u2'),Ket('u3')

    u1=list(map(int,input('coloumn 1 :').split()))
    u2=list(map(int,input('coloumn 2 :').split()))
    u3=list(map(int,input('coloumn 3 :').split()))
    amat=transpose(Matrix((u1,u2,u3)))
    display(amat)

    miu1,miu2,miu3=Matrix(u1),Matrix(u2),Matrix(u3)
    print('Let the given vectors be {a1,a2,a3}')
    display((u1,u2,u3))

    eq1=Eq(su1,sv1)
    display(eq1)
    display((sv1,eq,miu1))
    v1=miu1
    print('For our convinience we first find self inner products of v1')
    v1s=simplify(v1.dot(v1))
    ev1s=Bra(v1)*Ket(v1)
    display((mv1**2,eq,bv1*kv1,eq,ev1s,eq,v1s))
    eq2=Eq(((su2-((bv2*ku1)/(mv1**2))*su1)),sv2)
    display(eq2)
    u2v1=simplify(miu2.dot(v1))
    display((bv1*ku2,eq,Bra(u2)*Ket(v1),eq,u2v1))
    div1=simplify((u2v1/v1s))
    v2=simplify(miu2-(div1*v1))
    display((sv2,eq,v2))
    v2s=simplify(v2.dot(v2))
    display((bv2*kv2,eq,Bra(v2)*Ket(v2),eq,v2s))
    eq3=((su3-(((bv3*ku1)/(mv1**2))*sv1)-((bv3*ku2)/(mv2**2))*sv2))
    display((eq3,eq,sv3))
    u3v1=simplify(miu3.dot(v1))
    u3v2=simplify(miu3.dot(v2))
```

```
display((bv3*ku1,eq,Bra(u3)*Ket(v1),eq,u3v1))
display((bv3*ku2,eq,Bra(u3)*Ket(v2),eq,u3v2))
v2s=simplify(v2.dot(v2))
ev2s=Bra(v2)*Ket(v2)
display((mv2**2,eq,bv2*kv2,eq,ev2s,eq,v2s))
div2=simplify((u3v1/v1s))
div3=simplify((u3v2/v2s))
v3=simplify((miu3-(div2*v1)-(div3*v2)))
display((sv3,eq,v3))
v3s=simplify(v3.dot(v3))
ev3s=Bra(v3)*Ket(v3)
display((mv3**2,eq,bv3*kv3,eq,ev3s,eq,v3s))
print('Orthogonal Vectors We Got')
display((v1,v2,v3))
print('Orthonormal Basis')

w1,w2,w3,mul,div=sp.symbols('w1 w2 w3 * /')
display((w1,eq,(sv1/mv1)))
display((w2,eq,(sv2/mv2)))
display((w3,eq,(sv3/mv3)))
W1=simplify(v1/sqrt(v1s))
W2=simplify(v2/sqrt(v2s))
W3=simplify(v3/sqrt(v3s))
display((w1,eq,v1,div,simplify(sqrt(v1s))))
display((w1,eq,W1))
display((w2,eq,v2,div,simplify(sqrt(v2s))))
display((w2,eq,W2))
display((w3,eq,v3,div,simplify(sqrt(v3s))))
display((w3,eq,W3))
display((W1,W2,W3))
sq,sr,sa,st=symbols('Q R A T')
display((sq,eq,(((sv1/mv1)),((sv2/mv2)),((sv3/mv3)))))
display((sq,eq,transpose(Matrix((list(W1),list(W2),list(W3))))))
bw1,bw2,bw3=Bra('w1'),Bra('w2'),Bra('w3')
rmmat=Matrix(([bw1*ku1,bw1*ku2,bw1*ku3],[0,bw2*ku2,bw2*ku3],[0,0,bw3*ku3]))
display((sr,eq,rmmat))
a1w1=simplify(miu1.dot(W1))
display((bw1*ku1,eq,Bra(W1)*Ket(miu1),eq,a1w1))
a2w1=simplify(miu2.dot(W1))
display((bw1*ku2,eq,Bra(W1)*Ket(miu2),eq,a2w1))
a3w1=simplify(miu3.dot(W1))
display((bw1*ku3,eq,Bra(W1)*Ket(miu3),eq,a3w1))
a2w2=simplify(miu2.dot(W2))
display((bw2*ku2,eq,Bra(W2)*Ket(miu2),eq,a2w2))
a3w2=simplify(miu3.dot(W2))
display((bw2*ku3,eq,Bra(W2)*Ket(miu3),eq,a3w2))
a3w3=simplify(miu3.dot(W3))
```

```
    display((bw3*ku3,eq,Bra(miu3)*Ket(W3),eq,a3w3))
    rmat=Matrix(([a1w1,a2w1,a3w1],[0,a2w2,a3w2],[0,0,a3w3]))
    print('One Last Check')
    display((sa,eq,sq*sr,eq,transpose(Matrix((list(W1),list(W2),list(W3)))),mul,rmat,eq,transpose(Matrix((list(W1),list(W2),list(W3))))*rmat)
    quit()
```

```
________________________----Welcome to Linear Algebra Calculator Material----________________________



1 -- UNIT-I   -- SOLVING SYSTEM OF LINEAR EQUATIONS
2 -- UNIT-II  -- LINEAR DEPENDANCY AND BASIS FORMATION CHECKING
3 -- UNIT-III -- FINDING EIGEN VALUE AND EIGEN VECTOR
4 -- UNIT-IV  -- ORTHOGONALIZATION
5 -- UNIT-V   -- QR DECOMPOSITION
Enter Your Choice of Chapter : 2
------------- Welcome! To Find linear combination dependancy and basis formation checking ---------------
No of Times you going to execute the program : 1
Number of vectors : 4
Enter 1st row : 1 1 1 1
Enter 2nd row : 1 2 3 2
Enter 3rd row : 2 5 6 4
Enter 4th row : 2 6 8 5
Given Matrix
```

```
c:\users\elcot\appdata\local\programs\python\python39\lib\site-packages\IPython\lib\latextools.py:126: MatplotlibDeprecationWarning:
The to_png function was deprecated in Matplotlib 3.4 and will be removed two minor releases later. Use mathtext.math_to_image instead.
  mt.to_png(f, s, fontsize=12, dpi=dpi, color=color)
c:\users\elcot\appdata\local\programs\python\python39\lib\site-packages\IPython\lib\latextools.py:126: MatplotlibDeprecationWarning:
The to_rgba function was deprecated in Matplotlib 3.4 and will be removed two minor releases later. Use mathtext.math_to_image instead.
  mt.to_png(f, s, fontsize=12, dpi=dpi, color=color)
c:\users\elcot\appdata\local\programs\python\python39\lib\site-packages\IPython\lib\latextools.py:126: MatplotlibDeprecationWarning:
The to_mask function was deprecated in Matplotlib 3.4 and will be removed two minor releases later. Use mathtext.math_to_image instead.
  mt.to_png(f, s, fontsize=12, dpi=dpi, color=color)
c:\users\elcot\appdata\local\programs\python\python39\lib\site-packages\IPython\lib\latextools.py:126: MatplotlibDeprecationWarning:
The MathtextBackendBitmap class was deprecated in Matplotlib 3.4 and will be removed two minor releases later. Use mathtext.math_to_image ins
tead.
  mt.to_png(f, s, fontsize=12, dpi=dpi, color=color)
```

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

now

$$((x_1, *, [1, 1, 1, 1]), +, (x_2, *, [1, 2, 3, 2]), +, (x_3, *, [2, 5, 6, 4]), +, (x_4, *, [2, 6, 8, 5]), =, (0, 0, 0, 0), -->)$$

$$((x_1,\ x_1,\ x_1,\ x_1),\ +,\ (x_2,\ 2x_2,\ 3x_2,\ 2x_2),\ +,\ (2x_3,\ 5x_3,\ 6x_3,\ 4x_3),\ +,\ (2x_4,\ 6x_4,\ 8x_4,\ 5x_4),\ =,\ (0,\ 0,\ 0,\ 0),\ - - >)$$

$$x_1 + x_2 + 2x_3 + 2x_4 = 0$$

$$x_1 + 2x_2 + 5x_3 + 6x_4 = 0$$

$$x_1 + 3x_2 + 6x_3 + 8x_4 = 0$$

$$x_1 + 2x_2 + 4x_3 + 5x_4 = 0$$

rearranged matrix

$$\left(\begin{bmatrix}1 & 1 & 2 & 2\\1 & 2 & 5 & 6\\1 & 3 & 6 & 8\\1 & 2 & 4 & 5\end{bmatrix},\ \begin{bmatrix}x_1\\x_2\\x_3\\x_4\end{bmatrix},\ =,\ \begin{bmatrix}0\\0\\0\\0\end{bmatrix}\right)$$

augmented form

$$\begin{bmatrix}1 & 1 & 2 & 2 & 0\\1 & 2 & 5 & 6 & 0\\1 & 3 & 6 & 8 & 0\\1 & 2 & 4 & 5 & 0\end{bmatrix}$$

The two vectors are
linearly dependant

$$\{x_1 : x_4,\ x_2 : -x_4,\ x_3 : -x_4\}$$

dim R4 != 4, The given vectors NOT form a basis of R^4

```
In [ ]:
```