Experiment No.
Name of the Experiment

Date 12/4/22
Page No.

Experiment - 01
Vectors and Linear Combination

1.1 Plotting the vectors:

```
def plot_vectors(vectors, title=None):
    import numpy as np
    from matplotlib import pyplot as plt
    fig, ax = plt.subplots(figsize=(5,5))
    for x_cor, y_cor in vectors:
        ax.quiver(0,0, x_cor, y_cor
                  scale=1, angles='xy',
                  scale_units='xy', color='steelblue')
    limit = np.max(np.abs(vectors)) * 1.25
    ax.set_xlim([-limit, limit])
    ax.set_ylim([-limit, limit])
    ax.set_aspect('equal')
    ax.grid(True, linewidth=0.5, alpha=0.855)
    ax.spines['left'].set_position('center')
    ax.spines['bottom'].set_position('center')
    ax.spines['right'].set_color('none')
    ax.spines['top'].set_color('none')
    if title!=None:
        plt.title(title)
    plt.show()
```

chandra's

Experiment No.
Name of the Experiment

Date
Page No.

```
vectors = [(1,1), (1,2), (2,3), (1,3)]
plot_vectors(vectors, title = "plotting vectors using
                                quiver plot")
```

Basic operations on vectors:-

1) Plot the following vectors a=(1,3), b=(3,1), & a+b

```
import numpy as np
a = np.array((1,3))
b = np.array((3,1))
vectors = [a, b, a+b]
plot vectors (vectors, title = "Addition of a vector").
```

2) Plot the following vectors a=(1,2) and 2a

```
import numpy as np
a = np.array((1,2))
vectors = [a, 2*a]
plot_vectors (vectors = title = "Scalar multiplication")
```

chandra's

Experiment No. | Date
Name of the Experiment | Page No.

Program:-

```
import numpy as np
import sympy as sp
def is_in_span(vectors, s):
    variables = sp.symbols(f'a:{len(s)}')
    m = -vector
    for i, var in enumerate(variables):
        m = m + var * s[i]
    Scalars = sp.solve(m, variables, dict = True)
    if len(Scalars) == 0:
        Print("No! The given vectors is not in the
                Span of S")
        return False
    else:
        Print("Yes! The given vector is in the
                span s")
        Print("And the scalars are," Scalars[0])
        return Scalars
vector = np.array([2, -1, 1])
s = np.array([(1, 0, 2), (-1, 1, 1)])
is_in_span(vector, s)
```

2) $\begin{pmatrix} 1 & 2 \\ -3 & 4 \end{pmatrix}$ R, $S = \left\{ \begin{pmatrix} 1 & 0 \\ -1 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix} \right\}$

$\begin{pmatrix} 1 & 2 \\ -3 & 4 \end{pmatrix} = a_0 \begin{pmatrix} 1 & 0 \\ -1 & 0 \end{pmatrix} + a_1 \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix} + a_2 \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix}$

chandra's

SURYA Gold
Date Page

$S = \{e_1, e_2 \ldots e_n\}$ of ~~vectors~~ space standard basis. It is linearly independent
$e_1 = (1,0,0)$, $e_2 = (0,1,0)$, $e_3 = (0,0,1)$.

* Progeram:

```
import numpy as np
import Sympy as sp
def is independent ~~(f'a:{len(s)}')~~ (s):
    alphas = sp.symbols(f'a:{len(s)}')
    M = None
    for i, var in enumerate(alphas):
        if M is none:
            M = var * s[i]
        else:
            M = M + var * s[i]
    soln = np.solve(M, alphs, manual = True)
    if np.abs(soln[0]).sum() == 0:
        Print("Yes! The given set is linearly independent")
        return True
    else:
        Print("No! The given set is linearly dependent")
        Print("And the scalars are", soln)
        return False
```

1 a) s = np.array([[1,0,0,-1], [0,1,0,-1], [0,0,1,-1], [0,0,0,1]])
is independent(s).

2 a) ~~s = np.array~~ x = sp.Symbols('x')
s = np.array([[1+x+x**2], [x+x**2], [x**2]])
is independent(s)

SURYA Gold
Date Page

basis Program:

4.)
```
def is_basis(B, dim):
    if is_independent(B):
        if len(B) == int(dim):
            Print('And the dimension of V = {dim}
                  = number of elements of B.')
            Print('So, the given set is a basis.')
            return True
        else:
            Print("But, the dimension is not matched.
                  So, not a basis.")
            return False
    else:
        Print("So, the given set is not a basis")
        return False
B = np.array([[1,0,0,-1],[0,1,,0,-1],[0,0,1,-1],
              [0,0,0,1]])
dim = 4
is_basis(B, dim)
```

5.)

2.)
```
B = np.array([[1+x+x**2],[x+x**2],[x**2]])
dim = 3
is_basis(B, dim)
```

Output:

1.) Yes! the given set is linearly independent.
True

2.) Yes! the given set is linearly independent
True

9/14

SURYA Gold
Date Page

I * Program:-

1.)

```
from sympy import symbols, expand
import numpy as np

def T(x):
    x1 = x[0]
    x2 = x[1]
    return np.array([x1, 2*x1+x2])

def check_linear(T, no_coordinates_input=2):
    x = np.array(symbols(f'x:{no_coordinates_input}'))
    y = np.array(symbols(f'y:{no_coordinates_input}'))
    c = symbols('c')

    # compute the lhs and rhs
    lhs = T(c*x + y)
    rhs = c*T(x) + T(y)

    # expand each co-ordinate to compare
    lhs = [expand(i) for i in lhs]
    print(f'T(cx+y) is : {lhs}.')
    rhs = [expand(i) for i in rhs]
    print(f'cT(x) + T(y) is : {rhs}.')
    print('And, T(cx+y)')
    if lhs == rhs:
        print('And, T(cx+y) = cT(x) + T(y). So, the
              given T is linear.')
        return True
    else:
        print('And, T(cx+y) is not equal to cT(x)
```

2)

+ T(y). So, the given T is not linear.')
return False
check_linear(T)

2) def T(x)
x1 = x[0]
x2 = x[1]
x3 = x[2]
return np.array(
def check_linear(T, no_of_ordinates_input = 3):

II ~~Frankenality Th~~

II Rank Nullity Theorem:-

1) Range space:- Let T: V → W be the L.T, the Range space of T denoted by R(T) is defined to be,

$$R(T) = \{T(v) / v \in V\}$$

R(T) is the sub-space of W

If V is finite dimensional then, the rank of T r(T) denoted by r(T) will be the dimension of Range space

2) Null space:- Let T: V → W be the L.T. The null space denoted by N(T) is defined as,

$$N(T) = \{x \in V / T(x) = 0_w\}$$

It is also called Kernal of T.

The Nullity of T is dimension of Nullspace.

Nullity(L) + Rank(L) = dim(R^3)

$1 + 2 = 3$

$3 = 3$

1 *

Program:-

```
from sympy import Matrix
dim v = 3 # dim(R**3) = 3

A = Matrix([[1,-1,0], [2,0,1], [1,1,1]])
r = A.rank()
B = A.rref()
```

$R_3 \to 2R_3 - R_2$

$R_2 \to \frac{R_2}{2}$

$R_1 \to R_1 - 2R_2$

$x - z = 0$

$y + z = 0$

$x = z, \quad y = -z.$

2)

1)

SURYA Gold

Date Page

```
print(f'Range space of A is spanned by first {r} rows of :\n', B[0])

A_Nullspace = A.transpose().nullspace()
print('Nullspace of A is generated by the columns of:', A_Nullspace)

lhs = r + len(A_Nullspace)
rhs = dim V

print('Rank of A:', r)
print('Nullity of A:', len(A_Nullspace))
print('Dimension of the domain of T:', dim V)

if lhs == rhs:
    Print('Rank and Nullity theorem is verified!!')
else:
    Print('You only made a mistake!')
```

1.) Sol:

LT $\begin{pmatrix} 1 & 0 & 1 \\ 2 & 1 & 1 \\ -1 & 1 & -2 \end{pmatrix}$

I Program:

```
import numpy as np
from numpy import linalg as la

def T(x, y, z):
    return [x-y+z, 2*x+3*y, (1/2)*z, x+y-2*z]

B1 = np.array([[-1, 1, 0], [5, -1, 2], [1, 2, 1]])
Print ("The specified basis of the domain space:", B1)
```

II Program:

```
from sy
import

a, c1, c
A = Mat
u1 = np
u2 = np

v1 = np.
v2 = np

M = Ve
```

```
B2 = np.array([[1,1,0],[0,0,1],[1,5,2]])
Print("The specified basis for the co-domain space
        is", B2)

W1 = T(-1,1,0)
W2 = T(5,-1,2)
W3 = T(1,2,1)

Print("Evaluating T at the vectors of basis B1:", W1,
                                        W2, W3)

V1 = la.solve(B2, T, W1)
V2 = la.solve(B2, T, W2)
V3 = la.solve(B2. T, W3)

M = np.array([V1, V2, V3])

Print("The matrix of the linear transformation
        is:\n", M.T).
```

II Program:

```
from sympy import symbols, Matrix, solve
import numpy as np

a, c1, c2, x, y = symbols('a, c1, c2, x, y')
A = Matrix([[2,3],[4,-5]])
U1 = np.array([1,-1])
U2 = np.array([1,1])

V1 = np.array([1,0])
V2 = np.array([0,1])
```

SURYA Gold
Date _____ Page _____

```
T0 = [A[0,0]*v1, A[1,0]*v2]
T1 = [A[0,1]*v1, A[1,1]*v2]

sumT0 = T0[0] + T0[1]
sumT1 = T1[0] + T1[1]

Print("The images of the ordered basis vectors in
      B1 under the tranformation are", sumT0,
                                      sumT1)

eq = np.array([u1[0]*c1 + u2[0]*c2, u1[1]*
                                  c1 + u2[1]*c2])
p, q = eq[0], eq[1]
a = solve((p-x, q-y), c1, c2, dict=True)
c1, c2 = a[0][c1], a[0][c2]

Print("The co-ordinates of the vector (x,y) w.r.t
      B1 are", c1, "and", c2)

t = sumT0*c1 + SumT1*c2

Print("The linear tranformation of T is T(x,y)
                              =", tuple(t)).
```

$\frac{dy}{y} = \frac{dz}{z}$

$-\log y = \log z + \log c$

$\log z + \log y = \log c$

$yz = c_1 \rightarrow (1)$

$\frac{dx}{x^2+2y^2} = \frac{dy}{-xy} \Rightarrow \frac{dy}{dx} = \frac{-xy}{x^2+2y^2}$

$\frac{dx}{x^2} = \frac{dy}{y}$

→ Program:

~~import sympy as sp~~

~~P, Q, R~~

1)
```
from sympy import *
x, y, z = symbols('x, y, z')
P = y+z
Q = z+x
R = x+y
f = simplify(P*(diff(Q,z) - diff(R,y)) +
             Q*(diff(R,x) - diff(P,z)) +
             R*(diff(P,y) - diff(Q,x)))
if f == 0:
    Print("The eqn is integrable")
else:
    Print("The eqn is not integrable")
```

Output:-

The equation is integrable.

M = Var-X.S[i]

chandra's

Teacher's Signature :

$\Rightarrow p^2 - x = q^2 - y = a$

$p^2 = a + x, \quad q^2 = a + y$

$p = \sqrt{a+x}, \quad q = \sqrt{a+y}$

$du = p\,dx + q\,dy$

$du = \sqrt{a+x}\,dx + \sqrt{a+y}\,dy$

$\int du = \int \sqrt{a+x}\,dx + \int \sqrt{a+y}\,dy$

$u = \frac{2}{3}(a+x)^{3/2} + \frac{2}{3}(a+y)^{3/2} + \frac{2}{3}c$

$\frac{2}{3}z^{3/2} = \frac{2}{3}(a+x)^{3/2} + \frac{2}{3}(a+y)^{3/2} + \frac{2}{3}c$

$z^{3/2} = (a+x)^{3/2} + (a+y)^{3/2} + c$

* Program- from sympy.abc import x, y, a, b, c

Type-I

```
from sympy import *
x,y,z,a,b=symbols("x y z a b")
p=diff(z,x)
q=diff(z,y)
p=
f=Function('f')
u=f(x,y)
p=u.diff(x)
q=u.diff(y)
eq=p**2+q**2-1
Eq=eq.subs(p,a).subs(q,b)
Print("equation becomes", Eq)
b_val=solve(Eq,b)
```

```
print("b=", b_val)
z = a*x + b*y + c
ans = z.subs(b, b_val[0])
print("The required solution is", ans)
```

Type 2:-

```
from sympy import function, diff, Eq, pprint, solve, dsolve
from sympy.abc import x, y, a, b, c, p, q
z = Function("z")(u)
eqn = p*(1-q**2) - q*(1-z)
eqn1 = eqn.subs(p, diff(z,u)).subs(q, a*diff(z,u))
print(eqn1)
h1 = solve(eqn1, diff(z,u))
pprint(h1)
sol = dsolve(h1[0] - diff(z,u))
print(sol)
ans = sol.subs(u, x+a*y)
print("required answer is")
print(ans)
```

Type 3:-

```
from Sympy.abc import x, y, a, b, c, z, p, q
from Sympy import *
lhs = p+z
rhs = q+q
r1 = Eq(lhs, z)
r2 = Eq(rhs, z)
h1 = solve(r1, p)
h2 = solve(r2, q)
print("p=", h1, "and", "q=", h2)
z = integrate(h1[0], x) + integrate(h2[0], y)
print("The solution is", z)
```

```
if "u in one:
    M = var*x.s[i]
```

chandra's

SURYA Gold
Date Page

Typeh-2) from sympy.abc import x, y, a, b, p, q
from sympy import *
z = p*x+q*y+(p**2)-q**2)
sol = z.subs([(p,a),(q,b)])
print("general solution is", sol)

2) z = p*x+p*y+log(p*q)

3) z = ~~p.sqrt(x)~~ p*x+q*y+log(p*q)

→ Type 1 Output:

1) equation becomes a**2+b**2-1
b = [-Sqrt(-a**2+1), Sqrt(-a**2+1)]
the required solution is = a*x+(-y*Sqrt(-a

2) equation becomes a*b-1
b = [1/a]
the required solution is = a*x+c*y/a

3) Equation becomes a*b+a+b