Lab Component
of
Second Semester Engineering Mathematics

as prescribed by Visvesvaraya Technological University, Belagavi

Compiled by:

Dr. Ramananda H. S.
St Joseph Engineering College,
Mangaluru, INDIA.

Dr. K. Sushan Bairy
SOAS, REVA University,
Bengaluru, INDIA.

Dr. Smita S. Nagouda
CHRIST(Deemed to be University),
Central Campus, Bengaluru, INDIA.

Dr. Madhukar Krishnamurthy
BMS College of Engineering,
Bull Temple Road, Bengaluru, INDIA.

Dr. Chandra Shekara G.
BMS College of Engineering,
Bull Temple Road, Bengaluru, INDIA.

Mr. Sonam Kumar
AMC Engineering college,
Bannerghatta Road, Bengaluru, INDIA.

Message from BOS Chair

Dear Readers,

Welcome to the world of mathematics brought to life through the power of Python! In your hands, you hold a unique manual that combines the elegance of mathematics with the versatility of programming. Prepare to embark on a captivating journey where the realm of numbers, algorithms, and problem-solving converge.

This mathematics lab manual, infused with Python, is your gateway to experiencing mathematics in a dynamic and interactive way. By integrating programming into the study of mathematics, we aim to inspire you to explore, experiment, and develop a deep understanding of mathematical concepts through hands-on coding activities.

Python, a powerful and user-friendly programming language, serves as our trusty companion throughout this manual. It enables us to go beyond pen-and-paper calculations, unleashing the potential to solve complex problems, visualize mathematical concepts, and uncover patterns through the magic of coding. As you progress through the chapters, you will witness how Python becomes a bridge between abstract mathematical ideas and concrete computational implementations.

Inside these pages, you will embark on a variety of coding adventures that will challenge your logical thinking, enhance your problem-solving skills, and ignite your creativity. From building algorithms to solve equations, to simulating mathematical models, to analyzing data sets, each activity has been carefully crafted to reinforce fundamental mathematical principles while simultaneously developing your proficiency in Python.

Remember, programming is a skill that grows with practice. Don't be discouraged by the occasional hurdle or setback. Embrace the challenges as opportunities to learn, adapt, and improve. The exercises and examples provided in this manual will guide you through the intricacies of Python, gradually expanding your knowledge and confidence as you progress.

We extend our heartfelt appreciation to the authors, educators, and programmers who have contributed their expertise and passion to create this invaluable resource. Their dedication ensures that you have in your hands a manual that will equip you with the skills and knowledge to unravel the mysteries of mathematics using Python.

So, dear readers, let this mathematics lab manual with Python be your guide as you embark on a thrilling voyage of exploration and discovery. May it inspire you to develop a deep appreciation for the beauty of mathematics, the artistry of programming, and the infinite possibilities that arise when these two worlds intertwine.

Take hold of your imagination, harness the power of Python, and delve into the captivating world of mathematics like never before. Prepare to witness the magic of algorithms, to unravel the secrets of mathematical patterns, and to develop a lifelong love for the boundless synergy of mathematics and programming.

Wishing you a remarkable journey filled with mathematical enlightenment and Pythonic adventures!

I am very much thankful to the authors of this lab manual, Dr. K. Sushan Bairy, REVA University, Bengaluru, Dr. Ramananda H. S., St Joseph Engineering College, Mangaluru, Dr. Smita S. Nagouda, Christ(Deemed to be University), Bengaluru, Dr. Chandra Shekara G., BMS College of Enginering, Bengaluru, Dr. Madhukar Krishnamurthy, BMS College of Enginering, Bengaluru, Mr. Sonam Kumar, AMC Engineering

College, Bengaluru for their continuous effort to prepare this lab manual.

I am very much thankful to Dr. Vidyashankar S. Vice Chancellor VTU, Dr. Rangaswamy B E, Registrar VTU, Dr. T Srinivasa, Registrar Evaluation VTU, Dr. Sadashive Gowda, Dean Academic VTU, Dr. Sadashiv Halbhavi, Special officer VTU, All the BOS members, Basic Science & Humanities for their guidance, support and encouragement for bringing out this manual.

Dr Suresha M
Chairman Board of Studies in Basic Sciences & Humanities

Instructions and method of evaluation

1. In each Lab student have to show the record of previous Lab.
2. Each Lab will be evaluated for 15 marks and finally average will be taken for 15 marks.
3. Viva questions shall be asked in labs and attendance also can be considered for everyday Lab evaluation.
4. Tests shall be considered for 5 marks and final Lab assessment is for 20 marks.
5. Student has to score minimum 8 marks out of 20 to pass Lab component.

Contents: Electrical & Electronics Engineering Stream

LAB 1: Finding gradient, divergent, curl and their geometrical interpretation and Verification of Green's theorem

1.1 Objectives:

Use python

1. to find the gradient of a given scalar function.
2. to find find divergence and curl of a vector function.
3. to evaluate integrals using Green's theorem.

1.2 Method I:

1. To find gradient of $\phi = x^2y + 2xz - 4$.

```
#To find gradient of scalar point function.
from sympy.vector import *
from sympy import symbols
N=CoordSys3D('N') #Setting the coordinate system
x,y,z=symbols('x y z')
A=N.x**2*N.y+2*N.x*N.z-4 #Variables x,y,z to be used with coordinate
                                system N
delop=Del() #Del operator
display(delop(A)) #Del operator applied to A
gradA=gradient(A) #Gradient function is used
print(f"\n Gradient of {A} is \n")
display(gradA)
```

$$\left(\frac{\partial}{\partial \mathbf{x_N}}(\mathbf{x_N}^2\mathbf{y_N} + 2\mathbf{x_N}\mathbf{z_N} - 4)\right)\hat{\mathbf{i}}_\mathbf{N} + \left(\frac{\partial}{\partial \mathbf{y_N}}(\mathbf{x_N}^2\mathbf{y_N} + 2\mathbf{x_N}\mathbf{z_N} - 4)\right)\hat{\mathbf{j}}_\mathbf{N} + \left(\frac{\partial}{\partial \mathbf{z_N}}(\mathbf{x_N}^2\mathbf{y_N} + 2\mathbf{x_N}\mathbf{z_N} - 4)\right)\hat{\mathbf{k}}_\mathbf{N}$$

```
Gradient of N.x**2*N.y + 2*N.x*N.z - 4 is
```

$$(2\mathbf{x_N}\mathbf{y_N} + 2\mathbf{z_N})\hat{\mathbf{i}}_\mathbf{N} + (\mathbf{x_N}^2)\hat{\mathbf{j}}_\mathbf{N} + (2\mathbf{x_N})\hat{\mathbf{k}}_\mathbf{N}$$

2. To find divergence of $\vec{F} = x^2yz\hat{i} + y^2zx\hat{j} + z^2xy\hat{k}$

```
#To find divergence of a vector point function
from sympy.vector import *
from sympy import symbols
N=CoordSys3D('N')
x,y,z=symbols('x y z')
A=N.x**2*N.y*N.z*N.i+N.y**2*N.z*N.x*N.j+N.z**2*N.x*N.y*N.k
delop=Del()
divA=delop.dot(A)
display(divA)

print(f"\n Divergence of {A} is \n")
```

```
display(divergence(A))
```

$$\frac{\partial}{\partial \mathbf{z_N}} \mathbf{x_N y_N z_N}^2 + \frac{\partial}{\partial \mathbf{y_N}} \mathbf{x_N y_N}^2 \mathbf{z_N} + \frac{\partial}{\partial \mathbf{x_N}} \mathbf{x_N}^2 \mathbf{y_N z_N}$$

```
Divergence of N.x**2*N.y*N.z*N.i + N.x*N.y**2*N.z*N.j + N.x*N.y*N.z**2*N.k is
```

$$6\mathbf{x_N y_N z_N}$$

3. To find curl of $\vec{F} = x^2yz\hat{i} + y^2zx\hat{j} + z^2xy\hat{k}$

```
#To find curl of a vector point function
from sympy.vector import *
from sympy import symbols
N=CoordSys3D('N')
x,y,z=symbols('x y z')
A=N.x**2*N.y*N.z*N.i+N.y**2*N.z*N.x*N.j+N.z**2*N.x*N.y*N.k
delop=Del()
curlA=delop.cross(A)
display(curlA)

print(f"\n Curl of {A} is \n")
display(curl(A))
```

$$\left(\frac{\partial}{\partial \mathbf{y_N}} \mathbf{x_N y_N z_N}^2 - \frac{\partial}{\partial \mathbf{z_N}} \mathbf{x_N y_N}^2 \mathbf{z_N}\right) \mathbf{\hat{i}_N} + \left(-\frac{\partial}{\partial \mathbf{x_N}} \mathbf{x_N y_N z_N}^2 + \frac{\partial}{\partial \mathbf{z_N}} \mathbf{x_N}^2 \mathbf{y_N z_N}\right) \mathbf{\hat{j}_N} + \left(\frac{\partial}{\partial \mathbf{x_N}} \mathbf{x_N y_N}^2 \mathbf{z_N} - \frac{\partial}{\partial \mathbf{y_N}} \mathbf{x_N}^2 \mathbf{y_N z_N}\right) \mathbf{\hat{k}_N}$$

```
Curl of N.x**2*N.y*N.z*N.i + N.x*N.y**2*N.z*N.j + N.x*N.y*N.z**2*N.k is
```

$$\left(-\mathbf{x_N y_N}^2 + \mathbf{x_N z_N}^2\right) \mathbf{\hat{i}_N} + \left(\mathbf{x_N}^2 \mathbf{y_N} - \mathbf{y_N z_N}^2\right) \mathbf{\hat{j}_N} + \left(-\mathbf{x_N}^2 \mathbf{z_N} + \mathbf{y_N}^2 \mathbf{z_N}\right) \mathbf{\hat{k}_N}$$

1.3 Method II:

1. To find gradient of $\phi = x^2yz$.

```
#To find gradient of a scalar point function x^2yz
from sympy.physics.vector import *
from sympy import var,pprint
var('x,y,z')
v=ReferenceFrame('v')
F=v[0]**2*v[1]*v[2]
G=gradient(F,v)
F=F.subs([(v[0],x),(v[1],y),(v[2],z)])
print("Given scalar function F=")
display(F)
G=G.subs([(v[0],x),(v[1],y),(v[2],z)])
print("\n Gradient of F=")
display(G)
```

```
Given scalar function F=
```

x^2yz

```
Gradient of F=
```

$2xyz\hat{\mathbf{v}}_{\mathbf{x}} + x^2z\hat{\mathbf{v}}_{\mathbf{y}} + x^2y\hat{\mathbf{v}}_{\mathbf{z}}$

2. To find divergence of $\vec{F} = x^2y\hat{i} + yz^2\hat{j} + x^2z\hat{k}$.

```
#To find divergence of F=x^2yi+yz^2j+x^2zk
from sympy.physics.vector import *
from sympy import var
var('x,y,z')
v=ReferenceFrame('v')
F=v[0]**2*v[1]*v.x+v[1]*v[2]**2*v.y+v[0]**2*v[2]*v.z
G=divergence(F,v)
F=F.subs([(v[0],x),(v[1],y),(v[2],z)])
print("Given vector point function is ")
display(F)

G=G.subs([(v[0],x),(v[1],y),(v[2],z)])
print("Divergence of F=")
display(G)
```

```
Given vector point function is
```

$x^2y\hat{\mathbf{v}}_{\mathbf{x}} + yz^2\hat{\mathbf{v}}_{\mathbf{y}} + x^2z\hat{\mathbf{v}}_{\mathbf{z}}$

```
Divergence of F=
```

$x^2 + 2xy + z^2$

3. To find curl of $\vec{F} = xy^2\hat{i} + 2x^2yz\hat{j} - 3yz^2\hat{k}$

```
#To find curl of F=xy^2i+2x^2yzj-3yz^2k
from sympy.physics.vector import *
from sympy import var
var('x,y,z')
v=ReferenceFrame('v')
F=v[0]*v[1]**2*v.x+2*v[0]**2*v[1]*v[2]*v.y-3*v[1]*v[2]**2*v.z
G=curl(F,v)
F=F.subs([(v[0],x),(v[1],y),(v[2],z)])
print("Given vector point function is ")
display(F)

G=G.subs([(v[0],x),(v[1],y),(v[2],z)])
print("curl of F=")
display(G)
```

```
Given vector point function is
```

$xy^2\hat{\mathbf{v}}_{\mathbf{x}} + 2x^2yz\hat{\mathbf{v}}_{\mathbf{y}} - 3yz^2\hat{\mathbf{v}}_{\mathbf{z}}$

```
curl of F=
```

$(-2x^2y - 3z^2)\hat{\mathbf{v}}_{\mathbf{x}} + (4xyz - 2xy)\hat{\mathbf{v}}_{\mathbf{z}}$

1.4 Green's theorem

Statement of Green's theorem in the plane: If $P(x, y)$ and $Q(x, y)$ be two continuous functions having continuous partial derivatives in a region R of the xy-plane, bounded by a simple closed curve C, then

$$\oint (Pdx + Qdy) = \int\int_{R} \left(\frac{\partial Q}{\partial x} - \frac{\partial P}{\partial y} \right) dxdy.$$

1. Using Green's theorem, evaluate $\oint_c [(x + 2y)dx + (x - 2y)dy]$, where c is the region bounded by coordinate axes and the line $x = 1$ and $y = 1$.

```
from sympy import *
var('x,y')
p=x+2*y
q=x-2*y
f=diff(q,x)-diff(p,y)
soln=integrate(f,[x,0,1],[y,0,1])
print("I=",soln)
```

```
I= -1
```

2. Using Green's theorem, evaluate $\oint_c [(xy + y^2)dx + x^2dy]$, where c is the closed curve bounded by $y = x$ and $y = x^2$.

```
from sympy import *
var('x,y')
p=x*y+y**2
q=x**2
f=diff(q,x)-diff(p,y)
soln=integrate(f,[y,x**2,x],[x,0,1])
print("I=",soln)
```

```
I= -1/20
```

1.5 Exercise:

1. If $u = x + y + z$, $v = x^2 + y^2 + z^2$, $w = yz + zx + xy$, find grad$u$, grad$v$ and grad$w$.
   Ans: $\hat{i} + \hat{j} + \hat{k}$, $2(x\hat{i} + y\hat{j} + z\hat{k})$, $(y + z)\hat{i} + (z + x)\hat{j} + (z + x)\hat{k}$.

2. Evaluate div$F$ and curl$F$ at the point (1,2,3), given that $\vec{F} = x^2yz\hat{i} + xy^2z\hat{j} + xyz^2\hat{k}$.
   Ans: $6xyz$, $x(z^2 - y^2)\hat{i} + y(x^2 - z^2)\hat{j} + z(y^2 - x^2)\hat{k}$.

3. Prove that the vector $(yz - x^2)\hat{i} + (4y - z^2x)\hat{j} + (2xz - 4z)\hat{k}$ is solenoidal.

4. Find the vector normal to the surface $xy^3z^2 = 4$ at the point $(-1, -1, 2)$.
   Ans: $-4\hat{i} - 12\hat{j} + 4\hat{k}$.

5. If $\vec{R} = x\hat{\imath} + y\hat{\jmath} + z\hat{k}$, show that (i) $\nabla \cdot \vec{R} = 3$, (ii) $\nabla \times \vec{R} = 0$.

6. Using Green's theorem, evaluate $\oint_c [(3x + 4y)dx + (2x - 3y)dy]$, where c is the boundary of the circle $x^2 + y^2 = 4$.

   Ans: -8π

LAB 2: Computation of basis and dimension for a vector space and graphical representation of linear transformation

2.1 Objectives:

Use python

1. to verify the Rank nullity theorem of given linear transformation.
2. to compute the dimension of vector space.
3. to represent linear transformations graphically.

2.2 Rank Nullity Theorem

Verify the rank-nullity theorem for the linear transformation $T : \mathbb{R}^3 \to \mathbb{R}^3$ defined by $T(x, y, z) = (x + 4y + 7z, 2x + 5y + 8z, 3x + 6y + 9z)$.

```
import numpy as np
from scipy.linalg import null_space

# Define a linear transformation interms of matrix
A = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

# Find the rank of the matrix A
rank = np.linalg.matrix_rank(A)
print("Rank of the matrix",rank)

# Find the null space of the matrix A
ns = null_space(A)
print("Null space of the matrix",ns)
# Find the dimension of the null space
nullity = ns.shape[1]
print("Null space of the matrix",nullity)
# Verify the rank-nullity theorem
if rank + nullity == A.shape[1]:
    print("Rank-nullity theorem holds.")
else:
    print("Rank-nullity theorem does not hold.")
```

```
Rank of the matrix 2
Null space of the matrix [[-0.40824829]
 [ 0.81649658]
 [-0.40824829]]
Null space of the matrix 1
Rank-nullity theorem holds.
```

2.3 Dimension of Vector Space

Find the dimension of subspace spanned by the vectors $(1, 2, 3), (2, 3, 1)$ and $(3, 1, 2)$.

```
import numpy as np

# Define the vector space V
V = np.array([
    [1, 2, 3],
    [2, 3, 1],
    [3, 1, 2]])
# Find the dimension and basis of V
basis = np.linalg.matrix_rank(V)
dimension = V.shape[0]
print("Basis of the matrix",basis)
print("Dimension of the matrix",dimension)
```

```
Basis of the matrix 3
Dimension of the matrix 3
```

Extract the linearly independent rows in given matrix : Basis of Row space

```
from numpy import *
import sympy as sp
A=[[1,-1,1,1],[2,-5,2,2],[3,-3,5,3],[4,-4,4,4]]
AB=array(A)
S=shape(A)
n=len(A)
for i in range(n):
    if AB[i,i]==0:
        ab=copy(AB)
        for k in range(i+1,S[0]):
            if ab[k,i]!=0:
                ab[i,:]=AB[k,:]
                ab[k,:]=AB[i,:]
                AB=copy(ab)
    for j in range(i+1,n):
        Fact=AB[j,i]/AB[i,i]
        for k in range(i,n):
            AB[j,k]=AB[j,k]-Fact*AB[i,k]
display("REF of given matrix: ",sp.Matrix(AB))
temp = {(0, 0, 0,0)}
result = []
for idx, row in enumerate(map(tuple, AB)):
    if row not in temp:
        result.append(idx)
print("\n Basis are non-zero rows of A:")
display(sp.Matrix(AB[result]))
```

'REF of given matrix: '

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

Basis are non-zero rows of A:

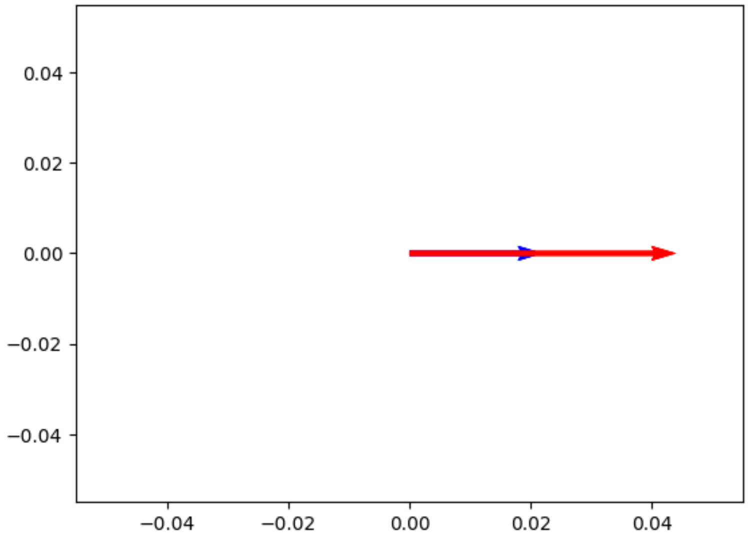
$$\begin{bmatrix} 1 & -1 & 1 & 1 \\ 0 & -3 & 0 & 0 \\ 0 & 0 & 2 & 0 \end{bmatrix}$$

2.4 Graphical representation of a transformation

2.4.1 Horizontal stretch:

Represent the horizontal stretch transformation $T : R^2 ßR^2$ geometrically
Find the image of vector $(10, 0)$ when it is stretched horizontally by 2 units.

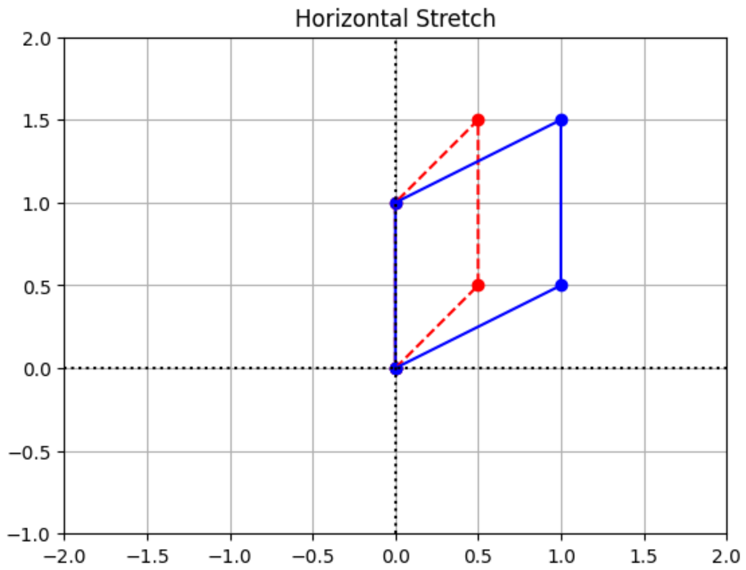
```
import numpy as np
import matplotlib.pyplot as plt
V = np.array([[10,0]])
origin = np.array([[0, 0, 0],[0, 0, 0]]) # origin point
A=np.matrix([[2,0],[0,1]])
V1=np.matrix(V)
V2=A*np.transpose(V1)
V2=np.array(V2)
plt.quiver(*origin, V[:,0], V[:,1], color=['b'], scale=50)
plt.quiver(*origin, V2[0,:], V2[1,:], color=['r'], scale=50)
plt.show()
```



Another example.

```
from math import pi, sin, cos
```

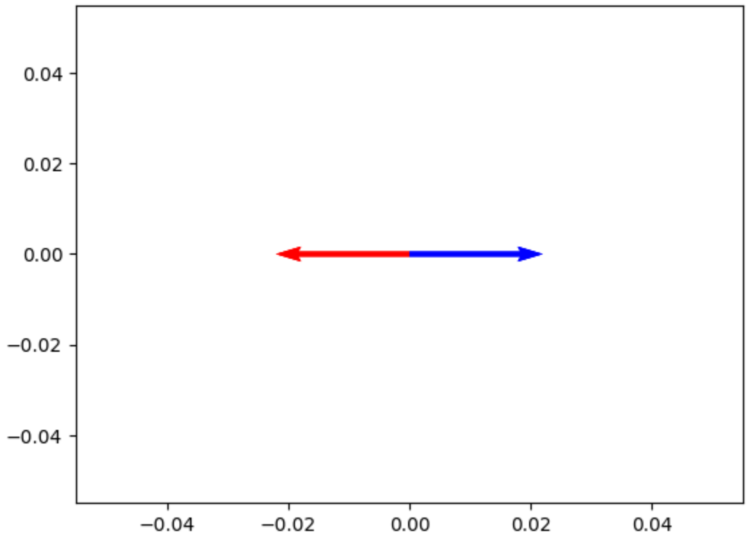
```
import matplotlib.pyplot as plt
import numpy as np

coords = np.array([[0,0],[0.5,0.5],[0.5,1.5],[0,1],[0,0]])
coords = coords.transpose()
coords
x = coords[0,:]
y = coords[1,:]

A = np.array([[2,0],[0,1]])
A_coords = A@coords
x_LT1 = A_coords[0,:]
y_LT1 = A_coords[1,:]

# Create the figure and axes objects
fig, ax = plt.subplots()

# Plot the points.  x and y are original vectors, x_LT1 and y_LT1 are
                                             images
ax.plot(x,y,'ro')
ax.plot(x_LT1,y_LT1,'bo')

# Connect the points by lines
ax.plot(x,y,'r',ls="--")
ax.plot(x_LT1,y_LT1,'b')

# Edit some settings
ax.axvline(x=0,color="k",ls=":")
ax.axhline(y=0,color="k",ls=":")
ax.grid(True)
ax.axis([-2,2,-1,2])
ax.set_aspect('equal')
ax.set_title("Horizontal Stretch");
```



2.4.2 Reflection:

Represent the reflection transformation $T : \mathbb{R}^2 \to \mathbb{R}^2$ geometrically.
Find the image of vector $(10, 0)$ when it is reflected about y axis.

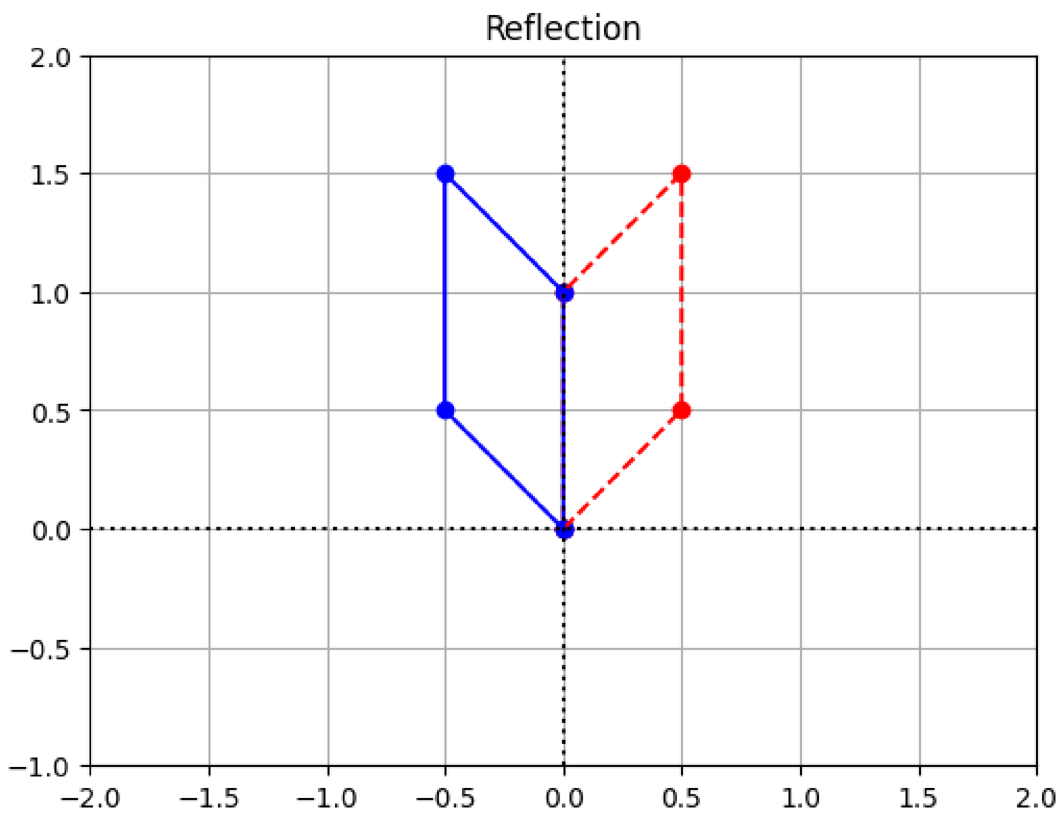
```
import numpy as np
import matplotlib.pyplot as plt
V = np.array([[10,0]])
origin = np.array([[0, 0, 0],[0, 0, 0]]) # origin point
A=np.matrix([[-1,0],[0,1]])
V1=np.matrix(V)
V2=A*np.transpose(V1)
V2=np.array(V2)
plt.quiver(*origin, V[:,0], V[:,1], color=['b'], scale=50)
plt.quiver(*origin, V2[0,:], V2[1,:], color=['r'], scale=50)
plt.show()
```



Another example.

```
B = np.array([[-1,0],[0,1]])
B_coords = B@coords

x_LT2 = B_coords[0,:]
y_LT2 = B_coords[1,:]

# Create the figure and axes objects
fig, ax = plt.subplots()

# Plot the points.  x and y are original vectors, x_LT1 and y_LT1 are
                                    images
ax.plot(x,y,'ro')
ax.plot(x_LT2,y_LT2,'bo')

# Connect the points by lines
ax.plot(x,y,'r',ls="--")
ax.plot(x_LT2,y_LT2,'b')

# Edit some settings
ax.axvline(x=0,color="k",ls=":")
```

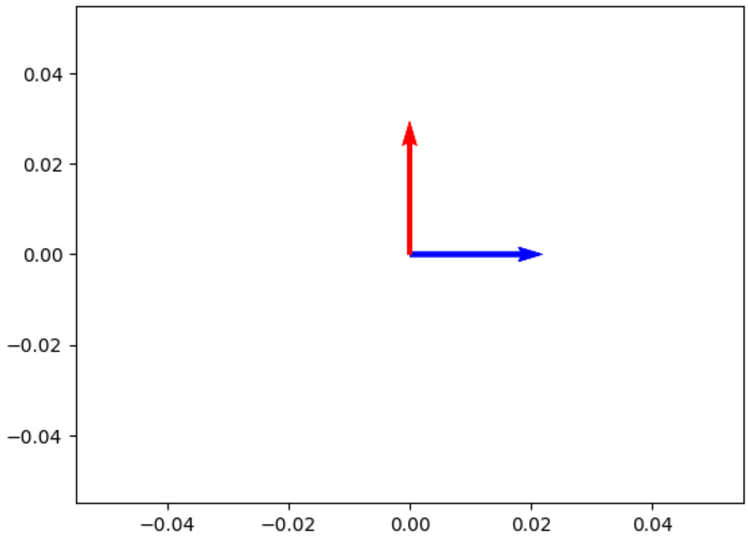
```
ax.axhline(y=0,color="k",ls=":")
ax.grid(True)
ax.axis([-2,2,-1,2])
ax.set_aspect('equal')
ax.set_title("Reflection");
```



2.4.3 Rotation:

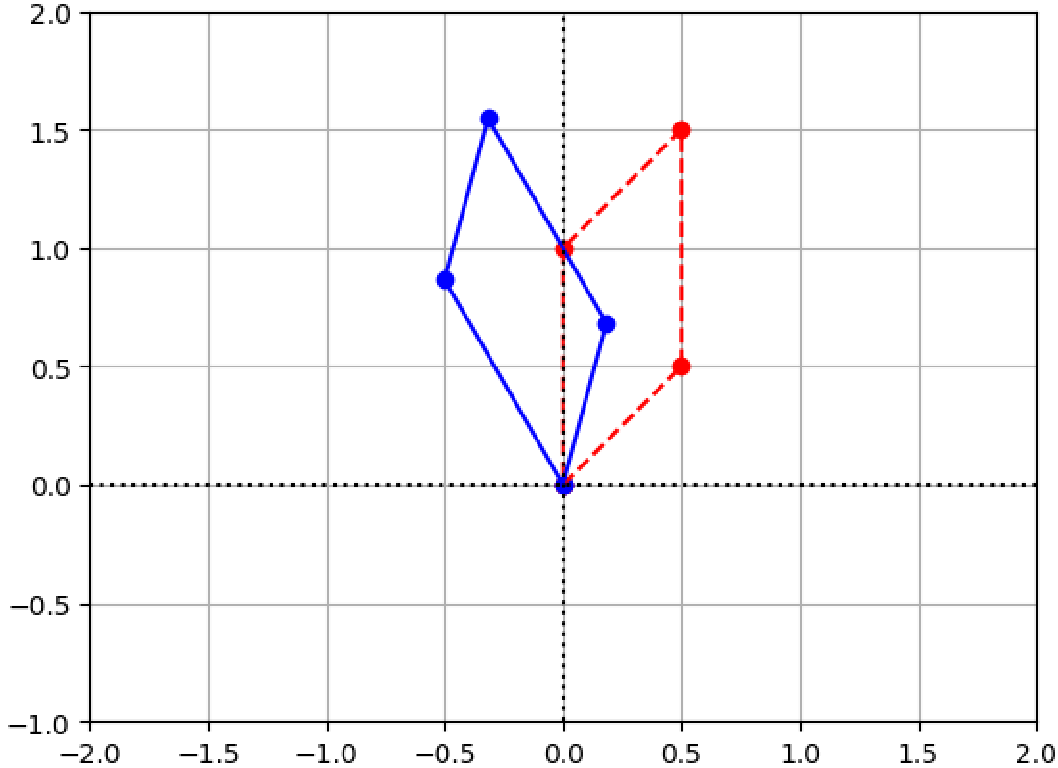
Represent the rotation transformation $T : \mathbb{R}^2 \to \mathbb{R}^2$ geometrically.
Find the image of vector $(10, 0)$ when it is rotated by $\pi/2$ radians.

```
import numpy as np
import matplotlib.pyplot as plt
V = np.array([[10,0]])
origin = np.array([[0, 0, 0],[0, 0, 0]]) # origin point
A=np.matrix([[0,-1],[1,1]])
V1=np.matrix(V)
V2=A*np.transpose(V1)
V2=np.array(V2)
plt.quiver(*origin, V[:,0], V[:,1], color=['b'], scale=50)
plt.quiver(*origin, V2[0,:], V2[1,:], color=['r'], scale=50)
plt.show()
```



Another example.

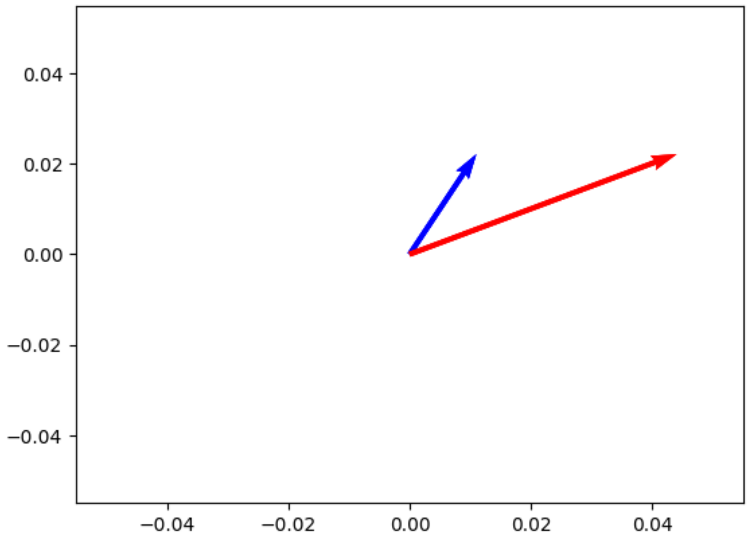
```
theta = pi/6
R = np.array([[cos(theta),-sin(theta)],[sin(theta),cos(theta)]])
R_coords = R@coords

x_LT3 = R_coords[0,:]
y_LT3 = R_coords[1,:]

# Create the figure and axes objects
fig, ax = plt.subplots()

# Plot the points.  x and y are original vectors, x_LT1 and y_LT1 are
                                    images
ax.plot(x,y,'ro')
ax.plot(x_LT3,y_LT3,'bo')

# Connect the points by lines
ax.plot(x,y,'r',ls="--")
ax.plot(x_LT3,y_LT3,'b')

# Edit some settings
ax.axvline(x=0,color="k",ls=":")
ax.axhline(y=0,color="k",ls=":")
ax.grid(True)
ax.axis([-2,2,-1,2])
ax.set_aspect('equal')
```



2.4.4 Shear Transformation

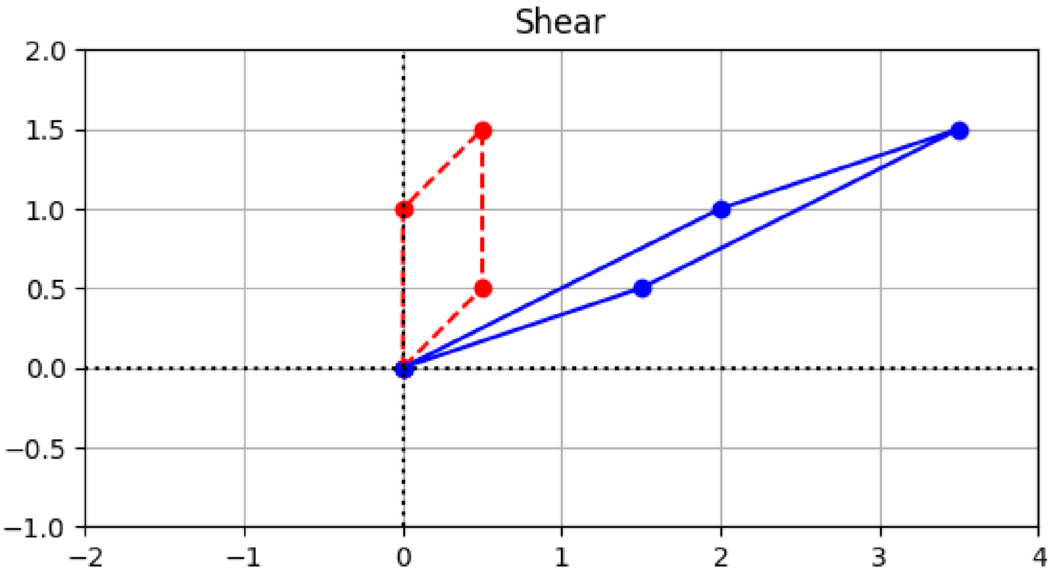
Represent the Shear transformation $T : \mathbb{R}^2 \to \mathbb{R}^2$ geometrically.
Find the image of $(2, 3)$ under shear transformation.

```
import numpy as np
import matplotlib.pyplot as plt
V = np.array([[2,3]])
origin = np.array([[0, 0, 0],[0, 0, 0]]) # origin point
A=np.matrix([[1,2],[0,1]])
V1=np.matrix(V)
V2=A*np.transpose(V1)
V2=np.array(V2)
print("Image of given vectors is:", V2)
plt.quiver(*origin, V[:,0], V[:,1], color=['b'], scale=20)
plt.quiver(*origin, V2[0,:], V2[1,:], color=['r'], scale=20)
plt.show()
```



Another example.

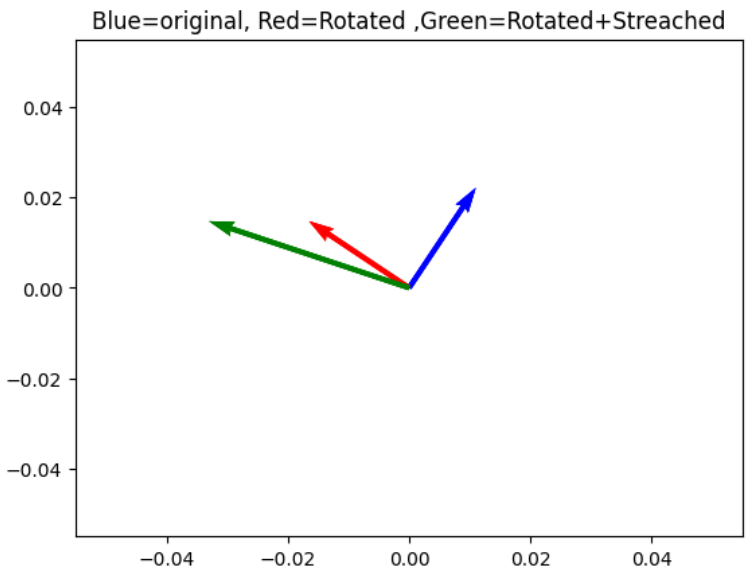
```
S = np.array([[1,2],[0,1]])
S_coords = S@coords

x_LT4 = S_coords[0,:]
y_LT4 = S_coords[1,:]


# Create the figure and axes objects
fig, ax = plt.subplots()

# Plot the points.  x and y are original vectors, x_LT1 and y_LT1 are
                                    images
ax.plot(x,y,'ro')
ax.plot(x_LT4,y_LT4,'bo')

# Connect the points by lines
ax.plot(x,y,'r',ls="--")
ax.plot(x_LT4,y_LT4,'b')

# Edit some settings
ax.axvline(x=0,color="k",ls=":")
ax.axhline(y=0,color="k",ls=":")
ax.grid(True)
ax.axis([-2,4,-1,2])
ax.set_aspect('equal')
ax.set_title("Shear");
```



2.4.5 Composition

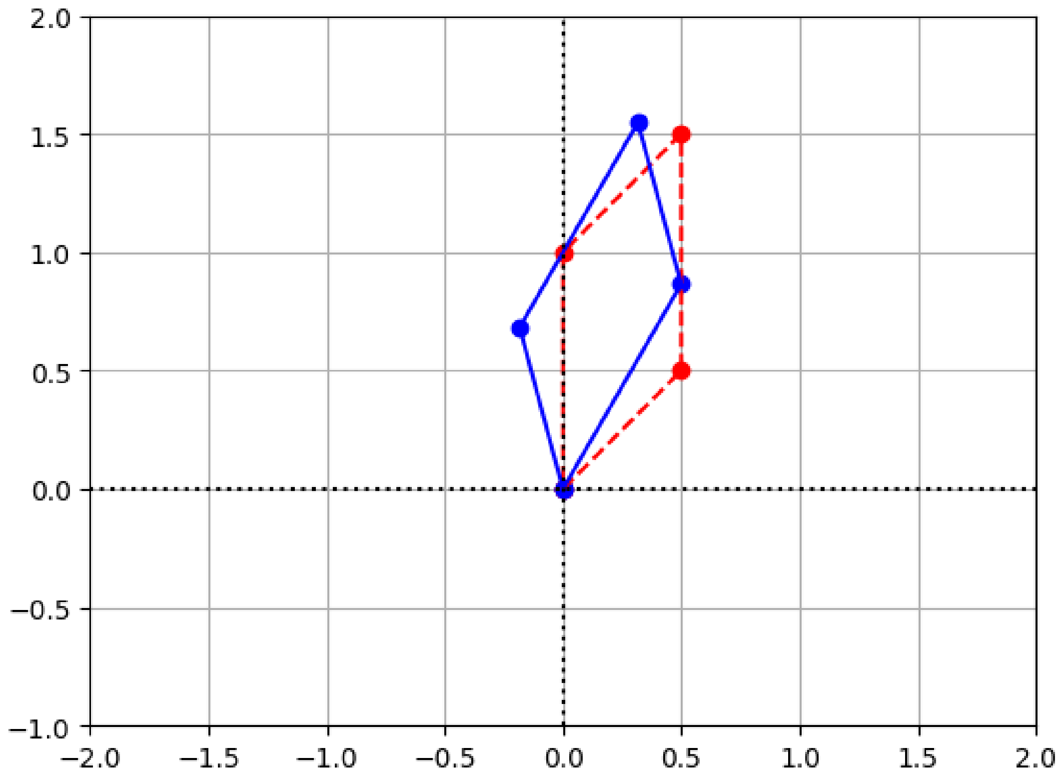
Represent the composition of two 2D transformations.
Find the image of vector $(10, 0)$ when it is rotated by $\pi/2$ radians then stretched horizontally 2 units.

```
import numpy as np
import matplotlib.pyplot as plt
V = np.array([[2,3]])
origin = np.array([[0, 0, 0],[0, 0, 0]]) # origin point
A=np.matrix([[0,-1],[1,0]])
B=np.matrix([[2,0],[0,1]])
V1=np.matrix(V)
V2=A*np.transpose(V1)
V3=B*V2
V2=np.array(V2)
V3=np.array(V3)
print("Image of given vectors is:", V3)
plt.quiver(*origin, V[:,0], V[:,1], color=['b'], scale=20)
plt.quiver(*origin, V2[0,:], V2[1,:], color=['r'], scale=20)
plt.quiver(*origin, V3[0,:], V3[1,:], color=['g'], scale=20)
plt.title('Blue=original, Red=Rotated ,Green=Rotated+Streached')
plt.show()
```



Another example.

```
C = np.array([[-cos(theta),sin(theta)],[sin(theta),cos(theta)]])
C_coords = C@coords

x_LT5 = C_coords[0,:]
y_LT5 = C_coords[1,:]


# Create the figure and axes objects
fig, ax = plt.subplots()

# Plot the points.  x and y are original vectors, x_LT1 and y_LT1 are
                                   images
ax.plot(x,y,'ro')
ax.plot(x_LT5,y_LT5,'bo')

# Connect the points by lines
ax.plot(x,y,'r',ls="--")
ax.plot(x_LT5,y_LT5,'b')

# Edit some settings
ax.axvline(x=0,color="k",ls=":")
ax.axhline(y=0,color="k",ls=":")
ax.grid(True)
ax.axis([-2,2,-1,2])
ax.set_aspect('equal')
```



2.5 Exercise:

1. Verify the rank nullity theorem for the following linear transformation
   a) $T : \mathbb{R}^2 \to \mathbb{R}^3$ defined by $T(x, y) = (x + 4y, 2x + 5y, 3x + 6y)$.
   Ans: Rank=2, Nullity=1, RNT verified
   b) $T : \mathbb{R}^3 \to \mathbb{R}^4$ defined by $T(x, y, z) = (x+4y-z, 2x+5y+8z, 3x+y+2z, x+y+z)$.
   Ans: Rank=3, Nullity=1, RNT verified
2. Find the dimension of the subspace spanned following set of vectors
   a) $S = (1, 2, 3, 4), (2, 4, 6, 8), (1, 1, 1, 1)$
   Ans: Dimension of subspace is 2
   b) $S = (1, -1, 3, 4), (2, 1, 6, 8), (1, 1, 1, 1), (3, 3, 3, 3)$
   Ans: Dimension of subspace is 3
3. Find the image of $(1, 3)$ under following $2D$ transformations
   a) Horizontal stretch
   b) Reflection
   c) Shear
   d) Rotation

LAB 3: Visualization in time and frequency domain of standard functions

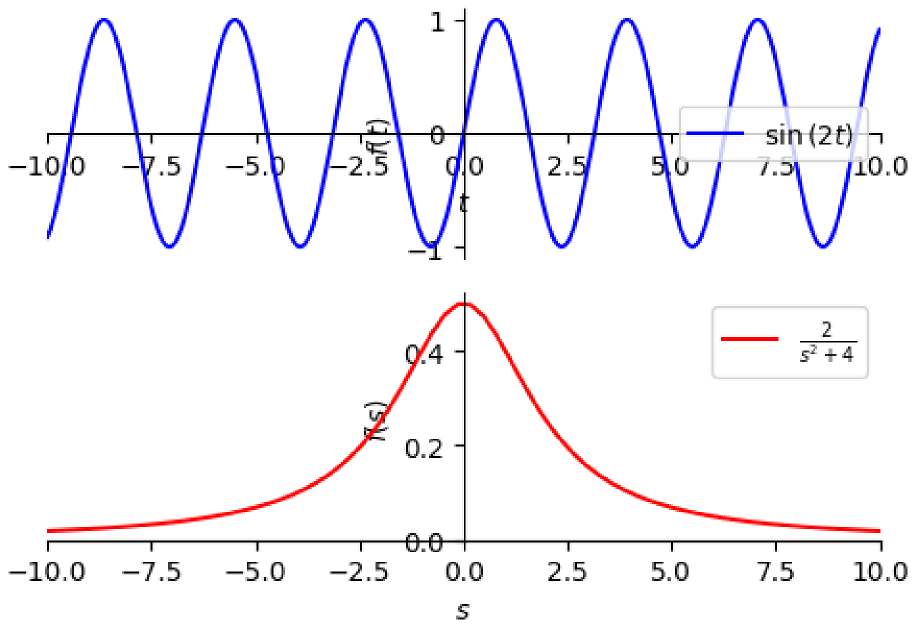
3.1 Objectives:

Use python

1. to use the standard in-built function of Laplace transform.
2. to graphically plot time and frequency domain of standard functions.

Represent the Laplace transform of $f(t) = \sin 2t$, both in time and frequency domains

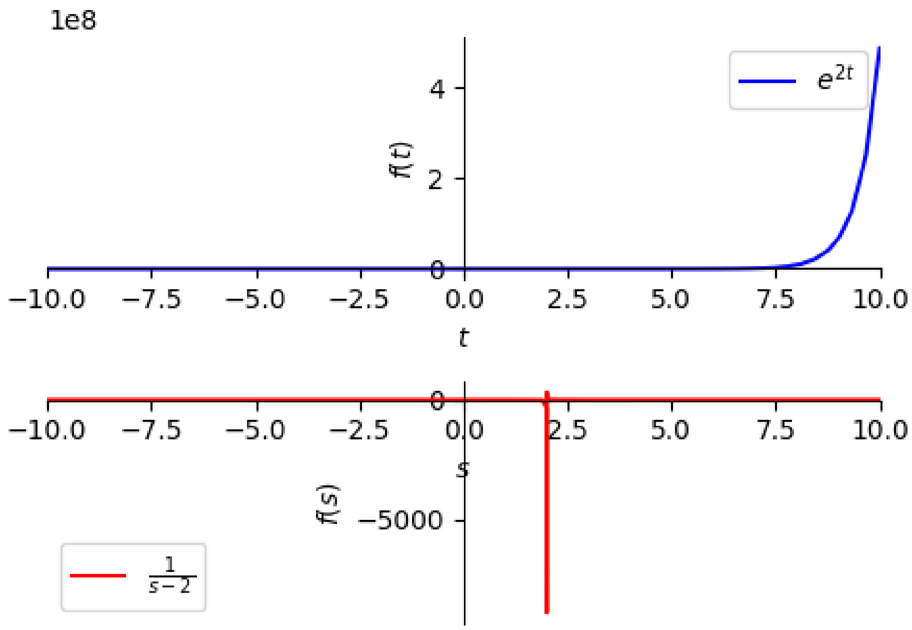
```
from sympy import *
import matplotlib.pyplot as plt
s,t=symbols('s t',positive=True)
f=sin(2*t)
F=laplace_transform(f,t,s)
print('The Laplace Transform of f is',F[0])
print(F[0].expand())
p1=plot(f, show=False,xlim=(-10, 10), line_color='blue', legend=True)
p2=plot(F[0],show=False,xlim=(-10, 10), line_color='red', legend=True)

plotgrid = plotting.PlotGrid(2, 1, p1, p2, show=False, size=(5., 3.5))
plotgrid.show()
```



Represent the Laplace transform of $f(t) = e^{2t}$, both in time and frequency domains

```
from sympy import *
import matplotlib.pyplot as plt
s,t=symbols('s t',positive=True)
f=exp(2*t)
F=laplace_transform(f,t,s)
print('The Laplace Transform of f is',F[0])
print(F[0].expand())
```

```
p1=plot(f, show=False,xlim=(-10, 10), line_color='blue', legend=True)
p2=plot(F[0],show=False,xlim=(-10, 10), line_color='red', legend=True)

plotgrid = plotting.PlotGrid(2, 1, p1, p2, show=False, size=(5., 3.5))
plotgrid.show()
```



3.2 Exercise:

1. Represent the following functions in time and frequency domains

   a) $\cos t$

   b) $\sinh t$

   c) $\cosh t$

   d) e^{-2t}

LAB 4: Computing Laplace transform and inverse Laplace transform of standard functions

4.1 Objectives:

Use python

1. to compute Laplace transform using in-built function.
2. to compute inverse Laplace transform using in-built function.

4.2 Laplace Transform

```
from sympy import *
t, s = symbols('t, s')
a = symbols('a', real=True, positive=True)
init_printing()
f=sin(a*t)
integrate(f*exp(-s*t), (t, 0, oo))
```

$$\begin{cases} \frac{1}{a\left(1+\frac{s^2}{a^2}\right)} & \text{for } |\arg(s)| < \frac{\pi}{2} \\ \int\limits_0^\infty e^{-st} \sin(at)\, dt & \text{otherwise} \end{cases}$$

```
from sympy import *
t, s = symbols('t, s')
a = symbols('a', real=True, positive=True)
init_printing()
f1=sin(a*t)
print('the Laplace transform of',f1 ,'is')
display(laplace_transform(f1, t, s, noconds=True))
f2=cos(a*t)
print('the Laplace transform of',f2 ,'is')
display(laplace_transform(f2, t, s, noconds=True))
f3=cosh(a*t)
print('the Laplace transform of',f3 ,'is')
display(laplace_transform(f3, t, s, noconds=True))
f4=exp(a*t)
print('the Laplace transform of',f4 ,'is')
display(laplace_transform(f4, t, s, noconds=True))
f5=sinh(a*t)
print('the Laplace transform of',f5 ,'is')
display(laplace_transform(f5, t, s, noconds=True))
f6=t**3
print('the Laplace transform of',f6 ,'is')
display(laplace_transform(f6, t, s, noconds=True))
```

```
the Laplace transform of sin(a*t) is
```

$$\frac{a}{a^2 + s^2}$$

```
the Laplace transform of cos(a*t) is
```

$$\frac{s}{a^2 + s^2}$$

```
the Laplace transform of cosh(a*t) is
```

$$\frac{s}{-a^2 + s^2}$$

```
the Laplace transform of exp(a*t) is
```

$$\frac{1}{-a + s}$$

```
the Laplace transform of sinh(a*t) is
```

$$\frac{a}{-a^2 + s^2}$$

```
the Laplace transform of t**3 is
```

$$\frac{6}{s^4}$$

4.3 Inverse Laplace Transform

```
# import inverse_laplace_transform
import sympy as sp
s= sp.symbols('s')

t = sp.Symbol('t', positive=True)
a = sp.symbols('a', real = True)

# Using inverse_laplace_transform() method
gfg = sp.inverse_laplace_transform(a/(s**2+a**2), s, t)
print(gfg)
gfg =sp. inverse_laplace_transform(s/(s**2+a**2), s, t)
print(gfg)
gfg = sp.inverse_laplace_transform(1/(s**4), s, t)
print(gfg)
gfg = sp.inverse_laplace_transform(1/(s**2-a**2), s, t)
print(gfg)
gfg =sp. inverse_laplace_transform(s/(s**2-a**2), s, t)
print(gfg)
gfg = sp.inverse_laplace_transform(1/(s**2-a**2), s, t)
print(gfg)
gfg = sp.inverse_laplace_transform((2)/(s-4), s, t)
print(gfg)
```

```
sin(a*t)
cos(a*t)
t**3/6
sinh(a*t)/a
cosh(a*t)
sinh(a*t)/a
2*exp(4*t)
```

4.4 Exercise:

1. Find the Laplace transform of $t\cos 4t$

   Ans: $\frac{s^2-16}{s^4+32s^2+256}$

2. Find the Laplace transform of $\frac{\sin 2t}{t}$

   Ans: $\tan^{-1}(2/s)$

3. Find the inverse Laplace transform of $\frac{s^2+s-2}{s(s-2)(s+3)}$

   Ans: $\frac{2}{5}e^{2t} + \frac{1}{3} + \frac{4}{15}e^{-3t}$

4. Find the inverse Laplace transform of $\frac{s}{s^4+4a^4}$

   Ans: $\frac{\sin at \sinh at}{2a^2}$

LAB 5: Laplace transform of convolution of two functions

5.1 Objectives:

Use python

1. to calculate Laplace Transform for convolution of two functions.
2. to verify Convolution Theorem for two given functions.

Let $f(t), g(t)$ be two functions, then the convolution $f \circledast g$ is given by

$$(f \circledast g)(t) = \int_0^t f(\tau)g(t-\tau)d\tau$$

The Convolution Theorem states that $\mathcal{L}(f \circledast g) = \mathcal{L}(f)\mathcal{L}(g)$.

1. Find the Laplace transform of the convolution of the functions $f(t) = t$ and $g(t) = e^t$.

```
from sympy import *
t, s, tau = symbols('t,s,tau')

# Given functions
f = t
g = exp(t)

# Convolution
fog = integrate(
    f.subs({t:tau})*g.subs({t:t-tau}),
    (tau,0,t)
)

# Alternately, convolution can also be calculated using
# the second function and a step reduced first function.
# gof = integrate(
#     g.subs({t:tau})*f.subs({t:t-tau}),
#     (tau,0,t)
# )

# Function to find laplace transform
def LT(func):
  return laplace_transform(func,t,s,noconds=True)

# Calculation
FOG = LT(fog)

print ('Laplace transform of the convolution of given functions is:')
display(FOG)
```

```
Laplace transform of the convolution of given functions is:
```

$$\frac{1}{s^2(s-1)}$$

2. Verify the Convolution Theorem for Laplace transform of the functions $f(t) = t$ and $g(t) = e^t$.

```
from sympy import *
t, s, tau = symbols('t,s,tau')

# Given functions
f = t
g = exp(t)

# Convolution
fog = integrate(
    f.subs({t:tau})*g.subs({t:t-tau}),
    (tau,0,t)
)

# Function to find laplace transform
def LT(func):
  return laplace_transform(func,t,s,noconds=True)

# Calculation
F = LT(f)
G = LT(g)
FOG = LT(fog)

print ('Convolution of given functions is:')
display (fog)
print ('Laplace transform of convolution is:')
display(FOG)
print ('Laplace transforms of given functions is:')
display(F,G)

# Checking Convolution Theorem
print ('\nChecking Convolution Theorem ...')
if FOG == F*G:
  print(' verified!')
```

```
Convolution of given functions is:
```

$$-t + e^t - 1$$

```
Laplace transform of convolution is:
```

$$\frac{1}{s^2 (s-1)}$$

```
Laplace transforms of given functions is:
```

$$\frac{1}{s^2}$$

$$\frac{1}{s-1}$$

```
Checking Convolution Theorem ...
 verified!
```

5.2 Exercise:

1. Find the Laplace transformation of the convolution of $f(t) = \sin t$ and $g(t) = \dfrac{\sin 3t}{3}$. Verify the convolution theorem for Laplace transformation.
$\left[\mathcal{L}\left(\sin t \circledast \dfrac{\sin 3t}{3}\right) = \dfrac{1}{s^2+1} \times \dfrac{1}{s^2+9} = \mathcal{L}(\sin t) \times \mathcal{L}\left(\dfrac{\sin 3t}{3}\right)\right]$.

2. Find the Laplace transformation of the convolution of $f(t) = t\cos t$ and $g(t) = t^2 \sin t$. Verify the convolution theorem for Laplace transform.
$\Big[\mathcal{L}\left(t\cos t \circledast t^2 \sin t\right) = \dfrac{11s^8 + 76s^6 + 1058s^4 - 1904s^2 + 171}{8(s^{10} + 5s^8 + 10s^6 + 10s^4 + 5s^2 + 1)}$
$= \dfrac{s^2-1}{s^4+2s^2+1} \times \dfrac{2(3s^2-1)}{s^6+3s^4+3s^2+1} = \mathcal{L}(t\cos t) \times \mathcal{L}\left(t^2 \sin t\right)\Big]$.

3. Find the Laplace transformation of the convolution of $f(t) = \cosh 3t$ and $g(t) = t^5$. Verify the convolution theorem for Laplace transform.
$\left[\mathcal{L}\left(\cosh 3t \circledast t^5\right) = \dfrac{120}{s^5(s^2-9)} = \dfrac{s}{s^2-9} \times \dfrac{120}{s^6} = \mathcal{L}(\cosh 3t) \times \mathcal{L}\left(t^5\right)\right]$.

4. Find the Laplace transformation of the convolution of $f(t) = \dfrac{4t+5}{e^{2t}}$ and $g(t) = \dfrac{4t-5}{e^{-2t}}$. Verify the convolution theorem for Laplace transform.
$\Big[\mathcal{L}\left(\dfrac{4t+5}{e^{2t}} \circledast \dfrac{4t-5}{e^{-2t}}\right) = \dfrac{196-25s^2}{s^4-8s^2+16} = \dfrac{5s+14}{s^2+4s+4} \times \dfrac{14-5s}{s^2-4s+4}$
$= \mathcal{L}\left(\dfrac{4t+5}{e^{2t}}\right) \times \mathcal{L}\left(\dfrac{4t-5}{e^{-2t}}\right)\Big]$

LAB 6: Solution of algebraic and transcendental equation by Regula-Falsi and Newton-Raphson method

6.1 Objectives:

Use python

1. to solve algebraic and transcendental equation by Regula-Falsi method.
2. to solve algebraic and transcendental equation by Newton-Raphson method.

6.2 Regula-Falsi method to solve a transcendental equation

Obtain a root of the equation $x^3 - 2x - 5 = 0$ between 2 and 3 by regula-falsi method. Perform 5 iterations.

```
# Regula Falsi method
from sympy import *
x=Symbol('x')
g =input('Enter the function  ') #%x^3-2*x-5;    %function
f=lambdify(x,g)
a=float(input('Enter a valus :')) #2
b=float(input('Enter b valus :')) # 3
N=int(input('Enter number of iterations :')) #5

for i in range(1,N+1):
    c=(a*f(b)-b*f(a))/(f(b)-f(a))
    if((f(a)*f(c)<0)):
        b=c
    else:
        a=c
    print('itration %d  \t  the root %0.3f \t function value %0.3f \n'%
                                    (i,c,f(c)));
```

```
Enter the function  x**3-2*x-5
Enter a valus :2
Enter b valus :3
Enter number of iterations :5
itration 1        the root 2.059        function value -0.391

itration 2        the root 2.081        function value -0.147

itration 3        the root 2.090        function value -0.055

itration 4        the root 2.093        function value -0.020

itration 5        the root 2.094        function value -0.007
```

Using tolerance value we can write the same program as follows:
Obtain a root of the equation $x^3 - 2x - 5 = 0$ between 2 and 3 by regula-falsi method. Correct to 3 decimal places.

```
# Regula Falsi method while loop2
from sympy import *
x=Symbol('x')
g =input('Enter the function  ')  #%x^3-2*x-5;    %function
f=lambdify(x,g)
a=float(input('Enter a valus :')) # 2
b=float(input('Enter b valus :')) # 3
N=float(input('Enter tolarence  :')) # 0.001
x=a;
c=b;
i=0
while (abs(x-c)>=N):
    x=c
    c=((a*f(b)-b*f(a))/(f(b)-f(a)));
    if((f(a)*f(c)<0)):
        b=c
    else:
        a=c
        i=i+1
    print('itration %d  \t  the root %0.3f \t function value %0.3f \n'%
                                    (i,c,f(c)));
print('final value of the root is %0.5f'%c)
```

```
Enter the function  x**3-2*x-5
Enter a valus :2
Enter b valus :3
Enter tolarence  :0.001
itration 1        the root 2.059        function value -0.391

itration 2        the root 2.081        function value -0.147

itration 3        the root 2.090        function value -0.055

itration 4        the root 2.093        function value -0.020

itration 5        the root 2.094        function value -0.007

itration 6        the root 2.094        function value -0.003

final value of the root is 2.09431
```

6.3 Newton-Raphson method to solve a transcendental equation

Find a root of the equation $3x = \cos x + 1$, near 1, by Newton Raphson method. Perform 5 iterations

```
from sympy import *
x=Symbol('x')
g =input('Enter the function  ')  #%3x-cos(x)-1;    %function
f=lambdify(x,g)
dg = diff(g);
```

```
df=lambdify(x,dg)
x0= float(input('Enter the intial approximation  ')); # x0=1
n= int(input('Enter the number of iterations  '));    #n=5;
for i in range(1,n+1):
    x1 =(x0 - (f(x0)/df(x0)))
    print('itration %d  \t  the root %0.3f \t function value %0.3f \n'%
                                        (i, x1,f(x1))); #print all
                                        iteration value
    x0 = x1
```

```
Enter the function  3*x-cos(x)-1
Enter the intial approximation  1
Enter the number of iterations  5
itration 1         the root 0.620          function value 0.046

itration 2         the root 0.607          function value 0.000

itration 3         the root 0.607          function value 0.000

itration 4         the root 0.607          function value 0.000

itration 5         the root 0.607          function value 0.000
```

6.4 Exercise:

1. Find a root of the equation $3x = \cos x + 1$, between 0 and 1, by Regula-falsi method. Perform 5 iterations.

   Ans: 0.607

2. Find a root of the equation $xe^x = 2$, between 0 and 1, by Regula-falsi method. Correct to 3 decimal places.

   Ans: 0.853

3. Obtain a real positive root of $x^4 - x = 0$, near 1, by Newton-Raphson method. Perform 4 iterations.

   Ans: 1.856

4. Obtain a real positive root of $x^4 + x^3 - 7x^2 - x + 5 = 0$, near 3, by Newton-Raphson method. Perform 7 iterations.

   Ans: 2.061

LAB 7: Interpolation /Extrapolation using Newton's forward and backward difference formula

7.1 Objectives:

Use python

1. to interpolate using Newton's Forward interpolation method.
2. to interpolate using Newton's backward interpolation method.
3. to extrapolate using Newton's backward interpolation method.

1. Use Newtons forward interpolation to obtain the interpolating polynomial and hence calculate y(2) for the following:

x:	1	3	5	7	9
y:	6	10	62	210	502

```
from sympy import *
import numpy as np
n = int(input('Enter number of data points: '))
210
x = np.zeros((n))
y = np.zeros((n,n))


# Reading data points
print('Enter data for x and y: ')
for i in range(n):
    x[i] = float(input( 'x['+str(i)+']='))
    y[i][0] = float(input( 'y['+str(i)+']='))

# Generating forward difference table
for i in range(1,n):
    for j in range(0,n-i):
        y[j][i] = y[j+1][i-1] - y[j][i-1]

print('\nFORWARD DIFFERENCE TABLE\n');

for i in range(0,n):
    print('%0.2f' %(x[i]), end='')
    for j in range(0, n-i):
        print('\t\t%0.2f' %(y[i][j]), end='')
    print()
 # obtaining the polynomial
t=symbols('t')
f=[] # f is a list type data


p=(t-x[0])/(x[1]-x[0])
f.append(p)
for i in range(1,n-1):
    f.append(f[i-1]*(p-i)/(i+1))
    poly=y[0][0]
for i in range(n-1):
    poly=poly+y[0][i+1]*f[i]
```

```
simp_poly=simplify(poly)
print('\nTHE INTERPOLATING POLYNOMIAL IS\n');
pprint(simp_poly)
# if you want to interpolate at some point the next session will help
inter=input('Do you want to interpolate at a point(y/n)? ') # y
if inter=='y':
    a=float(input('enter the point ')) #2
    interpol=lambdify(t,simp_poly)
    result=interpol(a)
    print('\nThe  value of the function at' ,a,'is\n',result);
```

```
Enter number of data points: 5
Enter data for x and y:
x[0]=1
y[0]=6
x[1]=3
y[1]=10
x[2]=5
y[2]=62
x[3]=7
y[3]=210
x[4]=9
y[4]=502

FORWARD DIFFERENCE TABLE

1.00            6.00            4.00            48.00           48.00           0.00
3.00            10.00           52.00           96.00           48.00
5.00            62.00           148.00          144.00
7.00            210.00          292.00
9.00            502.00

THE INTERPOLATING POLYNOMIAL IS

     3        2
1.0⋅t  - 3.0⋅t  + 1.0⋅t + 7.0
Do you want to interpolate at a point(y/n)? y
enter the point 2

The  value of the function at 2.0 is
 5.0
```

2. Use Newtons backward interpolation to obtain the interpolating polynomial and hence calculate y(8) for the following data:

x:	1	3	5	7	9
y:	6	10	62	210	502

```
from sympy import *
import numpy as np
import sys
print("This will use Newton's backword intepolation formula ")
# Reading number of unknowns
n = int(input('Enter number of data points: '))

# Making numpy array of n & n x n size and initializing
# to zero for storing x and y value along with differences of y
x = np.zeros((n))
y = np.zeros((n,n))


# Reading data points
```

```python
print('Enter data for x and y: ')
for i in range(n):
    x[i] = float(input( 'x['+str(i)+']='))
    y[i][0] = float(input( 'y['+str(i)+']='))

# Generating backward difference table
for i in range(1,n):
    for j in range(n-1,i-2,-1):
        y[j][i] = y[j][i-1] - y[j-1][i-1]


print('\nBACKWARD DIFFERENCE TABLE\n');

for i in range(0,n):
    print('%0.2f' %(x[i]), end='')
    for j in range(0, i+1):
        print('\t%0.2f' %(y[i][j]), end='')
    print()


# obtaining the polynomial
t=symbols('t')
f=[]


p=(t-x[n-1])/(x[1]-x[0])
f.append(p)
for i in range(1,n-1):
       f.append(f[i-1]*(p+i)/(i+1))

poly=y[n-1][0]
print(poly)
for i in range(n-1):
       poly=poly+y[n-1][i+1]*f[i]
       simp_poly=simplify(poly)
print('\nTHE INTERPOLATING POLYNOMIAL IS\n');
pprint(simp_poly)
# if you want to interpolate at some point the next session will help
inter=input('Do you want to interpolate at a point(y/n)? ')
if inter=='y':
       a=float(input('enter the point '))
       interpol=lambdify(t,simp_poly)
       result=interpol(a)
       print('\nThe  value of the function at' ,a,'is\n',result);
```

```
This will use Newton's backword intepolation formula
Enter number of data points: 5
Enter data for x and y:
x[0]=1
y[0]=6
x[1]=3
y[1]=10
x[2]=5
y[2]=62
x[3]=7
y[3]=210
x[4]=9
y[4]=502

BACKWARD DIFFERENCE TABLE

1.00    6.00
3.00    10.00   4.00
5.00    62.00   52.00   48.00
7.00    210.00  148.00  96.00   48.00
9.00    502.00  292.00  144.00  48.00   0.00
502.0

THE INTERPOLATING POLYNOMIAL IS

     3        2
1.0·t  - 3.0·t  + 1.0·t + 7.0
Do you want to interpolate at a point(y/n)? y
enter the point 8

The  value of the function at 8.0 is
 335.0
```

7.2 Exercise:

1. Obtain the interpolating polynomial for the following data

x:	0	1	2	3
y:	1	2	1	10

   Ans: $2x^3 - 7x^2 + 6x + 1$

2. Find the number of men getting wage Rs. 100 from the following table:

wage:	50	150	250	350
No. of men:	9	30	35	42

   Ans: 23 men

3. Using Newton's backward interpolation method obtain y(160) for the following data

x :	100	150	200	250	300
y :	10	13	15	17	18

   Ans: 13.42

4. Using Newtons forward interpolation polynomial and calculate y(1) and y(10).

x :	3	4	5	6	7	8	9
y :	4.8	8.4	14.5	23.6	36.2	52.8	73.9

   Ans: 3.1 and 100

LAB 8: Computation of area under the curve using Trapezoidal, Simpson's $\left(\frac{1}{3}\right)^{\text{rd}}$ and Simpsons $\left(\frac{3}{8}\right)^{\text{th}}$ rule

8.1 Objectives:

Use python

1. to find area under the curve represented by a given function using Trapezoidal rule.
2. to find area under the curve represented by a given function using Simpson's $\left(\frac{1}{3}\right)^{\text{rd}}$ rule.
3. to find area under the curve represented by a given function using Simpson's $\left(\frac{3}{8}\right)^{\text{th}}$ rule.
4. to find the area below the curve when discrete points on the curve are given.

8.2 Trapezoidal Rule

Evaluate $\int_0^5 \frac{1}{1+x^2}$.

```python
# Definition of the function to integrate
def my_func(x):
    return 1 / (1 + x ** 2)
```

```python
# Function to implement trapezoidal method
def trapezoidal(x0, xn, n):
  h = (xn - x0) / n                               # Calculating step size
  # Finding sum
  integration = my_func(x0) + my_func(xn)         # Adding first and last terms
  for i in range(1, n):
    k = x0 + i * h                                # i-th step value
    integration = integration + 2 * my_func(k)    # Adding areas of the trapezoids
  # Proportioning sum of trapezoid areas
  integration = integration * h / 2
  return integration
```

```python
# Input section
lower_limit = float(input("Enter lower limit of integration: "))
upper_limit = float(input("Enter upper limit of integration: "))
sub_interval = int(input("Enter number of sub intervals: "))

# Call trapezoidal() method and get result
result = trapezoidal(lower_limit, upper_limit, sub_interval)

# Print result
print("Integration result by Trapezoidal method is: " , result)
```

```
Enter lower limit of integration: 0
Enter upper limit of integration: 5
Enter number of sub intervals: 10
Integration result by Trapezoidal method is:  1.3731040812301099
```

8.3 Simpson's $\left(\frac{1}{3}\right)^{\text{rd}}$ Rule

Evaluate $\int_0^5 \frac{1}{1+x^2}$.

```
# Definition of the function to integrate
def my_func(x):
    return 1 / (1 + x ** 2)
```

```
# Function to implement the Simpson's one-third rule

def simpson13(x0,xn,n):
  h = (xn - x0) / n                    # calculating step size
  # Finding sum
  integration = (my_func(x0) + my_func(xn))
  k = x0
  for i in range(1,n):
    if i%2 == 0:
      integration = integration + 4 * my_func(k)
    else:
      integration = integration + 2 * my_func(k)
    k += h
  # Finding final integration value
  integration = integration * h * (1/3)
  return integration

# Input section
lower_limit = float(input("Enter lower limit of integration: "))
upper_limit = float(input("Enter upper limit of integration: "))
sub_interval = int(input("Enter number of sub intervals: "))

# Call trapezoidal() method and get result
result = simpson13(lower_limit, upper_limit, sub_interval)
print("Integration result by Simpson's 1/3 method is: %0.6f" % (result)
                                      )
```

```
Enter lower limit of integration: 0
Enter upper limit of integration: 5
Enter number of sub intervals: 100
Integration result by Simpson's 1/3 method is: 1.404120
```

8.4 Simpson's 3/8th rule

Evaluate $\int_0^6 \frac{1}{1+x^2}dx$ using Simpson's 3/8 th rule, taking 6 sub intervals

```
def simpsons_3_8_rule(f, a, b, n):
```

```python
    h = (b - a) / n
    s = f(a) + f(b)
    for i in range(1, n, 3):
        s += 3 * f(a + i * h)
    for i in range(3, n-1, 3):
        s += 3 * f(a + i * h)
    for i in range(2, n-2, 3):
        s += 2 * f(a + i * h)
    return s * 3 * h / 8

def f(x):
    return 1/(1+x**2) # function here

a = 0    # lower limit
b = 6 #   upper limit
n = 6 # number of sub intervals

result = simpsons_3_8_rule(f, a, b, n)
print('%3.5f'%result)
```

```
1.27631
```

8.5 Exercise:

1. Evaluate the integral $\int_0^1 \frac{x^2}{1+x^3}dx$ using Simpson's $\frac{1}{3}$ rule.

   Ans: 0.23108

2. Use Simpson's $\frac{3}{8}$ rule to find $\int_0^{0.6} e^{-x^2}dx$ by taking seven ordinates.

   Ans: 0.5351

3. Evaluate using trapezoidal rule $\int_0^{\pi} sin^2 x dx$. Take $n = 6$.

   Ans: $\pi/2$

4. A solid of revolution is formed by rotating about the x-axis, the area between the x-axis, the lines $x = 0$ and $x = 1$, and a curve through the points with the following co-ordinates:

x	y
0.00	1.0000
0.25	0.9896
0.50	0.9589
0.75	0.9089
1.00	0.8415

Estimate the volume of the solid formed using Simpson's $\frac{1}{3}$rd rule. Hint: Required volume is $\int_0^1 y^2 * \pi dx$. **[Ans: 2.8192]**

5. The velocity v(km/min) of a moped which starts from rest, is given at fixed intervals of time t(min) as follows:

t:	2	4	6	8	10	12	14	16	18	20
v:	10	18	25	29	32	20	11	5	2	0

Estimate approximately the distance covered in twenty minutes.

Answer for 5.

We know that ds/dt=v. So to get distance (s) we have to integrate.
Here $h = 2.2$, $v_0 = 0$, $v_1 = 10$, $v_2 = 18$, $v_3 = 25$ etc.

```
# we shall use simpson's 1/3 rule directly to estimate

h=2
y= [0, 10 ,18, 25, 29,32 ,20, 11 ,5 ,2 , 0]
result=(h/3)*((y[0]+y[10])+4*(y[1]+y[3]+y[5]+y[7]+y[9])+2*(y[2]+y[4]+y[
                                                    6]+y[8]))

print('%3.5f'%result,'km.')
```

```
309.33333 km.
```

LAB 9: Solution of ODE of first order and first degree by Taylor's series and Modified Euler's method

9.1 Objectives:

Use python

1. to solve ODE by Taylor series method.
2. to solve ODE by Modified Euler method.
3. to trace the solution curves.

9.2 Taylor series method to solve ODE

Solve: $\frac{dy}{dx} - 2y = 3e^x$ with $y(0) = 0$ using Taylor series method at $x = 0.1(0.1)0.3$.

```
## module taylor
'''X,Y = taylor(deriv,x,y,xStop,h).
4th-order Taylor series method for solving the initial value problem {y
                                    }' = {F(x,{y})}, where
{y} = {y[0],y[1],...y[n-1]}.
x,y = initial conditions
xStop = terminal value of x
h = increment of x
'''
from numpy import array
def taylor(deriv,x,y,xStop,h):
    X = []
    Y = []
    X.append(x)
    Y.append(y)
    while x < xStop:                    # Loop over integration steps
        D = deriv(x,y)                 # Derivatives of y
        H = 1.0
        for j in range(3):            # Build Taylor series
            H = H*h/(j + 1)
            y = y + D[j]*H            # H = h^j/j!
        x = x + h
        X.append(x) # Append results to
        Y.append(y) # lists X and Y

    return array(X),array(Y) # Convert lists into arrays

# deriv = user-supplied function that returns derivatives in the 4 x n
                                    array
'''
[y'[0] y'[1] y'[2] ... y'[n-1]
y''[0] y''[1] y''[2] ... y''[n-1]
y'''[0] y'''[1] y'''[2] ... y'''[n-1]
y''''[0] y''''[1] y''''[2] ... y''''[n-1]]
'''
def deriv(x,y):
    D = zeros((4,1))
```

```
    D[0] = [2*y[0] + 3*exp(x)]
    D[1] = [4*y[0]+ 9*exp(x)]
    D[2] = [8*y[0]+ 21*exp(x)]
    D[3] = [16*y[0]+ 45*exp(x)]
    return D

x = 0.0         # Initial value of x
xStop = 0.3       # last value
y = array([0.0])          # Initial values of y
h = 0.1              # Step size
X,Y = taylor(deriv,x,y,xStop,h)

print("The required values are :at x= %0.2f, y=%0.5f, x=%0.2f, y=%0.5f,
                                    x = %0.2f, y=%0.5f,x = %0.2f, y=%0
                                    .5f"%(X[0],Y[0],X[1],Y[1],X[2],Y[2]
                                    ,X[3],Y[3] ))
```

```
The required values are :at x= 0.00, y=0.00000, x=0.10, y=0.34850,
x = 0.20, y=0.81079,x = 0.30, y=1.41590
```

Solve $y' + 4y = x^2$ with initial conditions $y(0) = 1$ using Taylor series method at $x = 0.1, 0.2$.

```
from numpy import array
def taylor(deriv,x,y,xStop,h):
    X = []
    Y = []
    X.append(x)
    Y.append(y)
    while x < xStop:                    # Loop over integration steps
        D = deriv(x,y)                  # Derivatives of y
        H = 1.0
        for j in range(3):             # Build Taylor series
            H = H*h/(j + 1)
            y = y + D[j]*H             # H = h^j/j!
        x = x + h
        X.append(x) # Append results to
        Y.append(y) # lists X and Y

    return array(X),array(Y) # Convert lists into arrays

# deriv = user-supplied function that returns derivatives in the 4 x n
                                    array
'''
[y'[0] y'[1] y'[2] ... y'[n-1]
y"[0] y"[1] y"[2] ... y"[n-1]
y'''[0] y'''[1] y'''[2] ... y'''[n-1]
y""[0] y""[1] y""[2] ... y""[n-1]]
'''
def deriv(x,y):
    D = zeros((4,1))
    D[0] = [x**2-4*y[0]]
    D[1] = [2*x-4*x**2+16*y[0]]
    D[2] = [2-8*x+16*x**2-64*y[0]]
    D[3] = [-8+32*x-64*x**2+256*y[0]]
```

```
    return D

x = 0.0          # Initial value of x
xStop = 0.2       # last value
y = array([1.0])           # Initial values of y
h = 0.1                # Step size
X,Y = taylor(deriv,x,y,xStop,h)

print("The required values are :at x= %0.2f, y=%0.5f, x=%0.2f, y=%0.5f,
                                    x = %0.2f, y=%0.5f"%(X[0],Y[0],X[1
                                   ],Y[1],X[2],Y[2]))
```

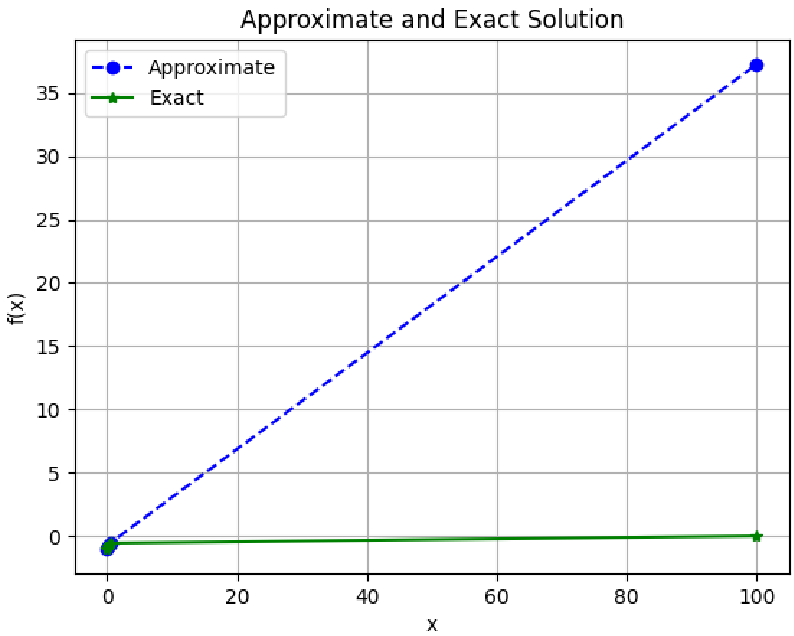
```
The required values are :at x= 0.00, y=1.00000, x=0.10, y=0.66967,
x = 0.20, y=0.45026
```

9.3 Euler's method to solve ODE:

To solve the ODE of the form $\frac{dy}{dx} = f(x, y)$ with initial conditions $y(x_0) = y_0$. The iterative formula is given by : $y(x_{(i+1)}) = y(x_i) + hf(x_i, y(x_i))$.
Solve: $y' = e^{-x}$ with $y(0) = -1$ using Euler's method at $x = 0.2(0.2)0.6$.

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

# Define parameters
f = lambda x, y: np.exp(-x) # ODE
h = 0.2 # Step size
y0 = -1 # Initial Condition
n=3
# Explicit Euler Method

y[0] = y0
x[0]=0

for i in range(0, n):
    x[i+1]=x[i]+h
    y[i + 1] = y[i] + h*f(x[i], y[i])

print("The required values are at x= %0.2f, y=%0.5f, x=%0.2f, y=%0.5f,
                                    x = %0.2f, y=%0.5f,x = %0.2f, y=%0.
                                    5f"%(x[0],y[0],x[1],y[1],x[2],y[2],
                                     x[3],y[3]))
print("\n\n")

plt.plot(x, y, 'bo--', label='Approximate')
plt.plot(x, -np.exp(-x), 'g*-', label='Exact')
plt.title("Approximate and Exact Solution" )
plt.xlabel('x')
plt.ylabel('f(x)')
plt.grid()
plt.legend(loc='best')
plt.show()
```

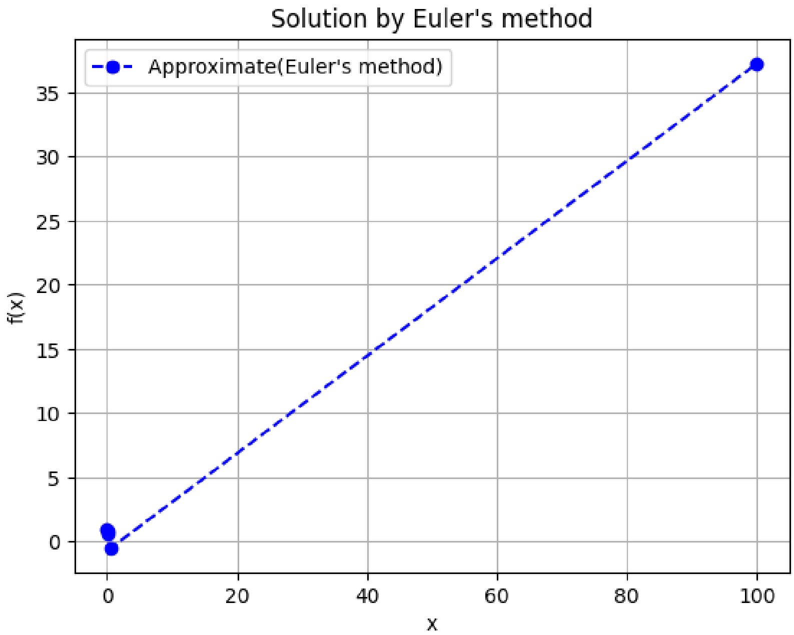
```
The required values are at x= 0.00, y=-1.00000, x=0.20, y=-0.80000,
x = 0.40, y=-0.63625,x = 0.60, y=-0.50219
```



Solve: $y' = -2y + x^3 e^{-2x}$ with $y(0) = 1$ using Euler's method at $x = 0.1, 0.2$.

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

# Define parameters
f = lambda x, y: -2*y+(x**3)*np.exp(-2*x) # ODE
h = 0.1 # Step size
y0 = 1 # Initial Condition
n=2
# Explicit Euler Method

y[0] = y0
x[0]=0

for i in range(0, n):
    x[i+1]=x[i]+h
    y[i + 1] = y[i] + h*f(x[i], y[i])


print("The required values are at x= %0.2f, y=%0.5f, x=%0.2f, y=%0.5f,x
                                      =%0.2f, y=%0.5f\n\n"%(x[0],y[0],x[1
                                      ],y[1],x[2],y[2]))

plt.plot(x, y, 'bo--', label="Approximate(Euler's method)")

plt.title("Solution by Euler's method")
plt.xlabel('x')
plt.ylabel('f(x)')
plt.grid()
plt.legend(loc='best')
plt.show()
```

```
The required values are at x= 0.00, y=1.00000, x=0.10, y=0.80000,
x=0.20, y=0.64008
```



9.4 Modified Euler's method

The iterative formula is:

$$y_1^{(n+1)} = y_0 + \frac{h}{2}[f(x_0, y_0) + f(x_1, y_1^{(n)})], \qquad n = 0, 1, 2, 3, \ldots,$$

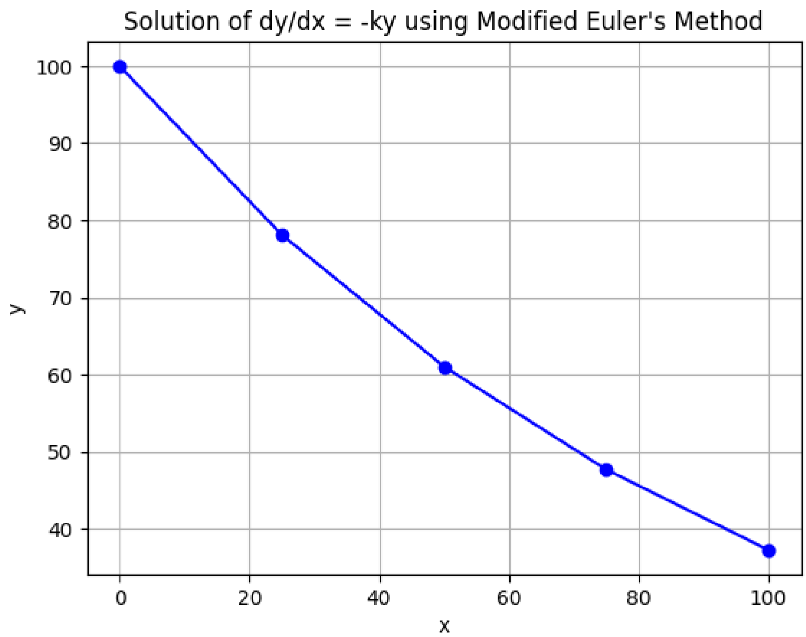
where, $y_1^{(n)}$ is the n^{th} approximation to y_1.

The first iteration will use Euler's method: $y_1^{(0)} = y_0 + hf(x_0, y_0)$.
Solve $y' = -ky$ with $y(0) = 100$ using modified Euler's method at $x = 100$, by taking $h = 25$.

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

def modified_euler(f, x0, y0, h, n):
    x = np.zeros(n+1)
    y = np.zeros(n+1)

    x[0] = x0
    y[0] = y0

    for i in range(n):
        x[i+1] = x[i] + h
        k1 = h * f(x[i], y[i])
        k2 = h * f(x[i+1], y[i] + k1)
        y[i+1] = y[i] + 0.5 * (k1 + k2)

    return x, y
```

```
def f(x, y):
    return -0.01 * y          # ODE dy/dx = -ky

x0 = 0.0
y0 = 100.0
h = 25
n = 4

x, y = modified_euler(f, x0, y0, h, n)

print("The required value at x= %0.2f, y=%0.5f"%(x[4],y[4]))
print("\n\n")

# Plotting the results
plt.plot(x, y, 'bo-')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Solution of dy/dx = -ky using Modified Euler\'s Method')
plt.grid(True)
plt.show()
```

```
The required value at x= 100.00, y=37.25290
```



9.5 Exercise:

1. Find $y(0.1)$ by Taylor Series exapnsion when $y' = x - y^2, y(0) = 1$.

   Ans: $y(0.1) = 0.9138$

2. Find $y(0.2)$ by Taylor Series exapnsion when $y' = x^2y - 1, y(0) = 1, h = 0.1$.

   Ans: $y(0.2) = 0.80227$

3. Evaluate by modified Euler's method: $y' = ln(x + y), y(0) = 2$ at $x = 0(0.2)0.8$.
   Ans: $2.0656, 2.1416, 2.2272, 2.3217$
4. Solve by modified Euler's method: $y' = x + y, y(0) = 1, h = 0.1, x = 0(0.1)0.3$.
   Ans: $1.1105, 1.2432, 1.4004$

LAB 10: Solution of ODE of first order and first degree by Runge-Kutta 4th order method and Milne's predictor and corrector method

10.1 Objectives:

1. To write a python program to solve first order differential equation using 4th order Runge Kutta method.

2. To write a python program to solve first order differential equation using Milne's predictor and corrector method.

10.2 Runge-Kutta method

Apply the Runge Kutta method to find the solution of $dy/dx = 1 + (y/x)$ at $y(2)$ taking $h = 0.2$. Given that $y(1) = 2$.

```
from sympy import *
import numpy  as np
def RungeKutta(g,x0,h,y0,xn):

  x,y=symbols('x,y')
  f=lambdify([x,y],g)
  xt=x0+h
  Y=[y0]
  while xt<=xn:
      k1=h*f(x0,y0)
      k2=h*f(x0+h/2, y0+k1/2)
      k3=h*f(x0+h/2, y0+k2/2)
      k4=h*f(x0+h, y0+k3)
      y1=y0+(1/6)*(k1+2*k2+2*k3+k4)
      Y.append(y1)
      #print('y(%3.3f'%xt,') is %3.3f'%y1)
      x0=xt
      y0=y1
      xt=xt+h
  return np.round(Y,2)
RungeKutta('1+(y/x)',1,0.2,2,2)
```

```
array([2.  , 2.62, 3.27, 3.95, 4.66, 5.39])
```

10.3 Milne's predictor and corrector method

Apply Milne's predictor and corrector method to solve $dy/dx = x^2 + (y/2)$ at y(1.4). Given that y(1)=2, y(1.1)=2.2156, y(1.2)=2.4649, y(1.3)=2.7514. Use corrector formula thrice.

```
# Milne's method to solve first order DE
# Use corrector formula thrice
x0=1
y0=2
```

```
y1=2.2156
y2=2.4649
y3=2.7514
h=0.1
x1=x0+h
x2=x1+h
x3=x2+h
x4=x3+h
def f(x,y):
  return x**2+(y/2)

y10=f(x0, y0)
y11=f(x1,y1)
y12=f(x2,y2)
y13=f(x3,y3)
y4p=y0+(4*h/3)*(2*y11-y12+2*y13)
print('predicted value of y4 is %3.3f'%y4p)
y14=f(x4,y4p);
for i in range(1,4):
  y4=y2+(h/3)*(y14+4*y13+y12);
  print('corrected value of y4 after \t iteration  %d  is \t %3.5f\t '%
                                    (i,y4))
  y14=f(x4,y4);
```

```
predicted value of y4 is 3.079
corrected value of y4 after          iteration  1  is           3.07940
corrected value of y4 after          iteration  2  is           3.07940
corrected value of y4 after          iteration  3  is           3.07940
```

In the next program, function will take all the inputs from the user and display the answer.

Apply Milne's predictor and corrector method to solve $dy/dx = x^2 + (y/2)$ at y(1.4). Given that y(1)=2, y(1.1)=2.2156, y(1.2)=2.4649, y(1.3)=2.7514. Use corrector formula thrice.

```
from sympy import *
def Milne(g,x0,h,y0,y1,y2,y3):
    x,y=symbols('x,y')
    f=lambdify([x,y],g)
    x1=x0+h
    x2=x1+h
    x3=x2+h
    x4=x3+h

    y10=f(x0, y0)
    y11=f(x1,y1)
    y12=f(x2,y2)
    y13=f(x3,y3)
    y4p=y0+(4*h/3)*(2*y11-y12+2*y13)
    print('predicted value of y4',y4p)
    y14=f(x4,y4p)
    for i in range(1,4):
        y4=y2+(h/3)*(y14+4*y13+y12)
        print('corrected value of y4 , iteration  %d '%i,y4)
```

```
        y14=f(x4,y4)
Milne('x**2+y/2',1,0.1,2,2.2156,2.4649,2.7514)
```

```
predicted value of y4 3.0792733333333335
corrected value of y4 , iteration  1  3.0793962222222224
corrected value of y4 , iteration  2  3.079398270370371
corrected value of y4 , iteration  3  3.079398304506173
```

Apply Milne's predictor and corrector method to solve $dy/dx = x - y^2$, y(0)=2 obtain y(0.8). Take h=0.2. Use Runge-Kutta method to calculate required initial values.

```
Y=RungeKutta('x-y**2',0,0.2,0,0.8)
print('y values from Runge -Kutta method:',Y)
Milne('x-y**2',0,0.2,Y[0],Y[1],Y[2],Y[3])
```

```
y values from Runge -Kutta method: [0.   0.02 0.08 0.18 0.3 ]
predicted value of y4 0.3042133333333334
corrected value of y4 , iteration  1  0.3047636165214815
corrected value of y4 , iteration  2  0.3047412758696499
corrected value of y4 , iteration  3  0.3047421836520892
```

10.4 Exercise:

1. Find $y(0.1)$ by Runge Kutta method when $y' = x - y^2, y(0) = 1$.

   Ans: $y(0.1) = 0.91379$

2. Evaluate by Runge Kutta method : $y' = log(x + y), y(0) = 2$ at $x = 0(0.2)0.8$.

   Ans: $2.155, 2.3418, 2.557, 2.801$

3. Solve by Milnes method: $y' = x + y$, y(0)=1, h=0.1, Calculate $y(0.4)$. Calculate required initial values from Runge Kutta method.

   Ans: 1.583649219

Contents: Computer Science and Engineering Stream

LAB 1: Programme to compute area, volume and center of gravity

1.1 Objectives:

Use python

1. to evaluate double integration.
2. to compute area and volume.
3. to calculate center of gravity of 2D object.

Syntax for the commands used:

1. Data pretty printer in Python:

```
pprint()
```

2. integrate:

```
integrate(function,(variable, min_limit, max_limit))
```

1.2 Double and triple integration

Example 1:

Evaluate the integral $\int_0^1 \int_0^x (x^2+y^2)dydx$

```
from sympy import *
x,y,z=symbols('x y z')
w1=integrate(x**2+y**2,(y,0,x),(x,0,1))
print(w1)
```

```
1/3
```

Example 2:

Evaluate the integral $\int_0^3 \int_0^{3-x} \int_0^{3-x-y} (xyz)dzdydx$

```
from sympy import *
x=Symbol('x')
y=Symbol('y')
z=Symbol('z')
w2=integrate((x*y*z),(z,0,3-x-y),(y,0,3-x),(x,0,3))
print(w2)
```

```
81/80
```

Example 3:

Prove that $\int\int(x^2+y^2)dydx = \int\int(x^2+y^2)dxdy$

```
from sympy import *
x=Symbol('x')
y=Symbol('y')
z=Symbol('z')
w3=integrate(x**2+y**2,y,x)
pprint(w3)
w4=integrate(x**2+y**2,x,y)
pprint(w4)
```

1.3 Area and Volume

Area of the region R in the cartesian form is $\int_R\int dxdy$

Example 4:

Find the area of an ellipse by double integration. A=4 $\int_0^a \int_0^{(b/a)\sqrt{a^2-x^2}} dydx$

```
from sympy import *
x=Symbol('x')
y=Symbol('y')
#a=Symbol('a')
#b=Symbol('b')
a=4
b=6
w3=4*integrate(1,(y,0,(b/a)*sqrt(a**2-x**2)),(x,0,a))
print(w3)
```

```
24.0*pi
```

1.4 Area of the region R in the polar form is $\int_R\int rdrd\theta$

Example 5:

Find the area of the cardioid $r = a(1 + cos\theta)$ by double integration

```
from sympy import *
r=Symbol('r')
t=Symbol('t')
a=Symbol('a')
#a=4

w3=2*integrate(r,(r,0,a*(1+cos(t))),(t,0,pi))
pprint(w3)
```

1.5 Volume of a solid is given by $\int_V \int \int dxdydz$

Example 6:

Find the volume of the tetrahedron bounded by the planes x=0,y=0 and z=0, $\frac{x}{a}+\frac{y}{b}+\frac{z}{c}=1$

```
from sympy import *
x=Symbol('x')
y=Symbol('y')
z=Symbol('z')
a=Symbol('a')
b=Symbol('b')
c=Symbol('c')
w2=integrate(1,(z,0,c*(1-x/a-y/b)),(y,0,b*(1-x/a)),(x,0,a))
print(w2)
```

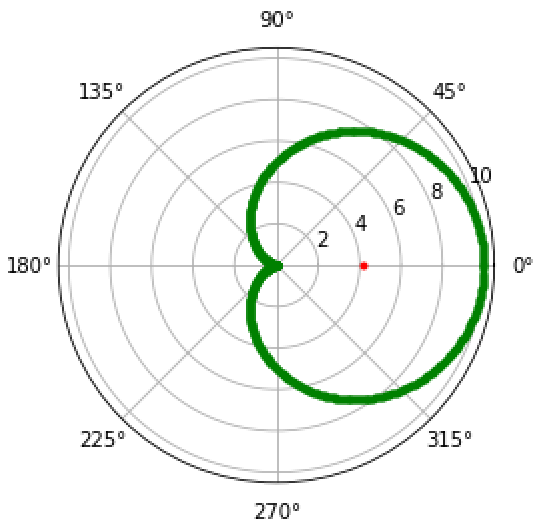
```
a*b*c/6
```

1.6 Center of Gravity

Find the center of gravity of cardioid . Plot the graph of cardioid and mark the center of gravity.

```
import numpy as np
import matplotlib.pyplot as plt
import math
from sympy import *
r=Symbol('r')
t=Symbol('t')
a=Symbol('a')
I1=integrate(cos(t)*r**2,(r,0,a*(1+cos(t))),(t,-pi,pi))
I2=integrate(r,(r,0,a*(1+cos(t))),(t,-pi,pi))
I=I1/I2
print(I)
I=I.subs(a,5)
plt.axes(projection = 'polar')
a=5


rad = np.arange(0, (2 * np.pi), 0.01)

# plotting the cardioid
for i in rad:
    r = a + (a*np.cos(i))
    plt.polar(i,r,'g.')

plt.polar(0,I,'r.')
plt.show()
```

```
5*a/6
```



1.7 Exercise:

1. Evaluate $\int_0^1 \int_0^x (x+y)dydx$
   Ans: 0.5

2. Find the $\int_0^{log(2)} \int_0^x \int_0^{x+log(y)} (e^{x+y+z})dzdydx$
   Ans: -0.2627

3. Find the area of positive quadrant of the circle $x^2 + y^2 = 16$
   Ans: 4π

4. Find the volume of the tetrahedron bounded by the planes x=0,y=0 and z=0, $\frac{x}{2} + \frac{y}{3} + \frac{z}{4} = 1$
   Ans: 4

LAB 2: Evaluation of improper integrals, Beta and Gamma functions

2.1 Objectives:

Use python

1. to evaluate improper integrals using Beta function.
2. to evaluate improper integrals using Gamma function.

Syntax for the commands used:

1. gamma

```
math.gamma(x)
```

Parameters :

x : The number whose gamma value needs to be computed.

2. beta

```
math.beta(x,y)
```

Parameters :

x ,y: The numbers whose beta value needs to be computed.

3. **Note:** We can evaluate improper integral involving infinity by using `inf`.

Example 1:

Evaluate $\int\limits_0^\infty e^{-x} dx$.

```
from sympy import *
x=symbols('x')
w1=integrate(exp(-x),(x,0,float('inf')))
print(simplify(w1))
```

```
1
```

Gamma function is $x(n) = \int_0^\infty e^{-x} x^{n-1} dx$

Example 2:

Evaluate $\Gamma(5)$ by using definition

```
from sympy import *
x=symbols('x')
w1=integrate(exp(-x)*x**4,(x,0,float('inf')))
print(simplify(w1))
```

```
24
```

Example 3:

Evaluate $\int_0^\infty e^{-st}\cos(4t)dt$. That is Laplace transform of $\cos(4t)$

```
from sympy import *
t,s=symbols('t,s')
# for infinity in sympy we use oo
w1=integrate(exp(-s*t)*cos(4*t),(t,0,oo))
display(simplify(w1))
```

$$\begin{cases} \frac{s}{s^2+16} & \text{for } 2|\arg(s)| < \pi \\ \int_0^\infty e^{-st}\cos(4t)\,dt & \text{otherwise} \end{cases}$$

Example 4:

Find Beta(3,5), Gamma(5)

```
#beta and gamma functions
from sympy import beta, gamma
m=input('m :');
n=input('n :');
m=float(m);
n=float(n);
s=beta(m,n);
t=gamma(n)
print('gamma (',n,') is %3.3f'%t)
print('Beta (',m,n,') is  %3.3f'%s)
```

```
m :3
n :5
gamma ( 5.0 ) is 24.000
Beta ( 3.0 5.0 ) is  0.010
```

Example 5:

Calculate Beta(5/2,7/2) and Gamma(5/2).

```
#beta and gamma functions
# If the number is a fraction give it in decimals. Eg 5/2=2.5
from sympy import beta, gamma
m=float(input('m : '));
n=float(input('n :'));

s=beta(m,n);
t=gamma(n)
print('gamma (',n,') is %3.3f'%t)
print('Beta (',m,n,') is %3.3f '%s)
```

```
m : 2.5
n :3.5
gamma ( 3.5 ) is 3.323
Beta ( 2.5 3.5 ) is 0.037
```

Example 6:

Verify that $Beta(m,n) = Gamma(m)Gamma(n)/Gamma(m+n)$ for m=5 and n=7

```
from sympy import beta, gamma
m=5;
n=7;
m=float(m);
n=float(n);
s=beta(m,n);
t=(gamma(m)*gamma(n))/gamma(m+n);
print(s,t)
if (abs(s-t)<=0.00001):
    print('beta and gamma are related')
else:
    print('given values are wrong')
```

```
0.0004329004329004329 0.0004329004329004329
beta and gamma are related
```

2.2 Exercise:

1. Evaluate $\int_0^\infty e^{-t}cos(2t)dt$
   Ans: 1/5

2. Find the value of Beta(5/2,9/2)
   Ans: 0.0214

3. Find the value of Gamma(13)
   Ans: 479001600

4. Verify that $Beta(m,n) = Gamma(m)Gamma(n)/Gamma(m+n)$ for m=7/2 and n=11/2
   Ans: True

LAB 3: Finding gradient, divergent, curl and their geometrical interpretation

1.1 Objectives:

Use python

1. to find the gradient of a given scalar function.
2. to find find divergence and curl of a vector function.

1.2 Method I:

To find gradient of $\phi = x^2y + 2xz - 4$.

```
#To find gradient of scalar point function.
from sympy.vector import *
from sympy import symbols
N=CoordSys3D('N') #Setting the coordinate system
x,y,z=symbols('x y z')
A=N.x**2*N.y+2*N.x*N.z-4 #Variables x,y,z to be used with coordinate
                                    system N
delop=Del() #Del operator
display(delop(A)) #Del operator applied to A
gradA=gradient(A) #Gradient function is used
print(f"\n Gradient of {A} is \n")
display(gradA)
```

$$\left(\frac{\partial}{\partial \mathbf{x_N}}\left(\mathbf{x_N}^2\mathbf{y_N} + 2\mathbf{x_N}\mathbf{z_N} - 4\right)\right)\hat{\mathbf{i}}_\mathbf{N} + \left(\frac{\partial}{\partial \mathbf{y_N}}\left(\mathbf{x_N}^2\mathbf{y_N} + 2\mathbf{x_N}\mathbf{z_N} - 4\right)\right)\hat{\mathbf{j}}_\mathbf{N} + \left(\frac{\partial}{\partial \mathbf{z_N}}\left(\mathbf{x_N}^2\mathbf{y_N} + 2\mathbf{x_N}\mathbf{z_N} - 4\right)\right)\hat{\mathbf{k}}_\mathbf{N}$$

```
Gradient of N.x**2*N.y + 2*N.x*N.z - 4 is
```

$$\left(2\mathbf{x_N}\mathbf{y_N} + 2\mathbf{z_N}\right)\hat{\mathbf{i}}_\mathbf{N} + \left(\mathbf{x_N}^2\right)\hat{\mathbf{j}}_\mathbf{N} + \left(2\mathbf{x_N}\right)\hat{\mathbf{k}}_\mathbf{N}$$

To find divergence of $\vec{F} = x^2yz\hat{i} + y^2zx\hat{j} + z^2xy\hat{k}$

```
#To find divergence of a vector point function
from sympy.vector import *
from sympy import symbols
N=CoordSys3D('N')
x,y,z=symbols('x y z')
A=N.x**2*N.y*N.z*N.i+N.y**2*N.z*N.x*N.j+N.z**2*N.x*N.y*N.k
delop=Del()
divA=delop.dot(A)
display(divA)

print(f"\n Divergence of {A} is \n")
display(divergence(A))
```

$$\frac{\partial}{\partial \mathbf{z_N}} \mathbf{x_N y_N z_N}^2 + \frac{\partial}{\partial \mathbf{y_N}} \mathbf{x_N y_N}^2 \mathbf{z_N} + \frac{\partial}{\partial \mathbf{x_N}} \mathbf{x_N}^2 \mathbf{y_N z_N}$$

```
Divergence of N.x**2*N.y*N.z*N.i + N.x*N.y**2*N.z*N.j + N.x*N.y*N.z**2*N.k is
```

$$6\mathbf{x_N y_N z_N}$$

To find curl of $\vec{F} = x^2yz\hat{i} + y^2zx\hat{j} + z^2xy\hat{k}$

```
#To find curl of a vector point function
from sympy.vector import *
from sympy import symbols
N=CoordSys3D('N')
x,y,z=symbols('x y z')
A=N.x**2*N.y*N.z*N.i+N.y**2*N.z*N.x*N.j+N.z**2*N.x*N.y*N.k
delop=Del()
curlA=delop.cross(A)
display(curlA)

print(f"\n Curl of {A} is \n")
display(curl(A))
```

$$\left(\frac{\partial}{\partial \mathbf{y_N}} \mathbf{x_N y_N z_N}^2 - \frac{\partial}{\partial \mathbf{z_N}} \mathbf{x_N y_N}^2 \mathbf{z_N}\right) \hat{\mathbf{i}}_\mathbf{N} + \left(-\frac{\partial}{\partial \mathbf{x_N}} \mathbf{x_N y_N z_N}^2 + \frac{\partial}{\partial \mathbf{z_N}} \mathbf{x_N}^2 \mathbf{y_N z_N}\right) \hat{\mathbf{j}}_\mathbf{N} + \left(\frac{\partial}{\partial \mathbf{x_N}} \mathbf{x_N y_N}^2 \mathbf{z_N} - \frac{\partial}{\partial \mathbf{y_N}} \mathbf{x_N}^2 \mathbf{y_N z_N}\right) \hat{\mathbf{k}}_\mathbf{N}$$

```
Curl of N.x**2*N.y*N.z*N.i + N.x*N.y**2*N.z*N.j + N.x*N.y*N.z**2*N.k is
```

$$\left(-\mathbf{x_N y_N}^2 + \mathbf{x_N z_N}^2\right) \hat{\mathbf{i}}_\mathbf{N} + \left(\mathbf{x_N}^2 \mathbf{y_N} - \mathbf{y_N z_N}^2\right) \hat{\mathbf{j}}_\mathbf{N} + \left(-\mathbf{x_N}^2 \mathbf{z_N} + \mathbf{y_N}^2 \mathbf{z_N}\right) \hat{\mathbf{k}}_\mathbf{N}$$

1.3 Method II:

To find gradient of $\phi = x^2yz$.

```
#To find gradient of a scalar point function x^2yz
from sympy.physics.vector import *
from sympy import var,pprint
var('x,y,z')
v=ReferenceFrame('v')
F=v[0]**2*v[1]*v[2]
G=gradient(F,v)
F=F.subs([(v[0],x),(v[1],y),(v[2],z)])
print("Given scalar function F=")
display(F)
G=G.subs([(v[0],x),(v[1],y),(v[2],z)])
print("\n Gradient of F=")
display(G)
```

Given scalar function F=

$$x^2yz$$

Gradient of F=

$$2xyz\hat{\mathbf{v}}_{\mathbf{x}} + x^2z\hat{\mathbf{v}}_{\mathbf{y}} + x^2y\hat{\mathbf{v}}_{\mathbf{z}}$$

To find divergence of $\vec{F} = x^2y\hat{i} + yz^2\hat{j} + x^2z\hat{k}$.

```
#To find divergence of F=x^2yi+yz^2j+x^2zk
from sympy.physics.vector import *
from sympy import var
var('x,y,z')
v=ReferenceFrame('v')
F=v[0]**2*v[1]*v.x+v[1]*v[2]**2*v.y+v[0]**2*v[2]*v.z
G=divergence(F,v)
F=F.subs([(v[0],x),(v[1],y),(v[2],z)])
print("Given vector point function is ")
display(F)

G=G.subs([(v[0],x),(v[1],y),(v[2],z)])
print("Divergence of F=")
display(G)
```

Given vector point function is

$$x^2y\hat{\mathbf{v}}_{\mathbf{x}} + yz^2\hat{\mathbf{v}}_{\mathbf{y}} + x^2z\hat{\mathbf{v}}_{\mathbf{z}}$$

Divergence of F=

$$x^2 + 2xy + z^2$$

To find curl of $\vec{F} = xy^2\hat{i} + 2x^2yz\hat{j} - 3yz^2\hat{k}$

```
#To find curl of F=xy^2i+2x^2yzj-3yz^2k
from sympy.physics.vector import *
from sympy import var
var('x,y,z')
v=ReferenceFrame('v')
F=v[0]*v[1]**2*v.x+2*v[0]**2*v[1]*v[2]*v.y-3*v[1]*v[2]**2*v.z
G=curl(F,v)
F=F.subs([(v[0],x),(v[1],y),(v[2],z)])
print("Given vector point function is ")
display(F)

G=G.subs([(v[0],x),(v[1],y),(v[2],z)])
print("curl of F=")
display(G)
```

Given vector point function is

$$xy^2\,\hat{\mathbf{v}}_{\mathbf{x}} + 2x^2yz\hat{\mathbf{v}}_{\mathbf{y}} - 3yz^2\,\hat{\mathbf{v}}_{\mathbf{z}}$$

curl of F=

$$(-2x^2y - 3z^2)\hat{\mathbf{v}}_{\mathbf{x}} + (4xyz - 2xy)\hat{\mathbf{v}}_{\mathbf{z}}$$

1.4 Exercise:

1. If $u = x + y + z$, $v = x^2 + y^2 + z^2$, $w = yz + zx + xy$, find grad$u$, grad$v$ and grad$w$.
   Ans: $\hat{i} + \hat{j} + \hat{k}$, $2(x\hat{i} + y\hat{j} + z\hat{k})$, $(y + z)\hat{i} + (z + x)\hat{j} + (z + x)\hat{k}$.
2. Evaluate div$F$ and curl$F$ at the point (1,2,3), given that $\vec{F} = x^2yz\hat{i} + xy^2z\hat{j} + xyz^2\hat{k}$.
   Ans: $6xyz$, $x(z^2 - y^2)\hat{i} + y(x^2 - z^2)\hat{j} + z(y^2 - x^2)\hat{k}$.
3. Prove that the vector $(yz - x^2)\hat{i} + (4y - z^2x)\hat{j} + (2xz - 4z)\hat{k}$ is solenoidal.
4. Find the vector normal to the surface $xy^3z^2 = 4$ at the point $(-1, -1, 2)$.
   Ans: $-4\hat{i} - 12\hat{j} + 4\hat{k}$.
5. If $\vec{R} = x\hat{i} + y\hat{j} + z\hat{k}$, show that (i) $\nabla \cdot \vec{R} = 3$, (ii) $\nabla \times \vec{R} = 0$.

LAB 4: Computation of basis and dimension for a vector space and graphical representation of linear transformation

4.1 Objectives:

Use python

1. to verify the Rank nullity theorem of given linear transformation
2. to compute the dimension of vector space
3. to represent linear transformations graphically

4.2 Rank Nullity Theorem

Verify the rank-nullity theorem for the linear transformation $T : \mathbb{R}^3 \to \mathbb{R}^3$ defined by $T(x, y, z) = (x + 4y + 7z, 2x + 5y + 8z, 3x + 6y + 9z)$.

```
import numpy as np
from scipy.linalg import null_space

# Define a linear transformation interms of matrix
A = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

# Find the rank of the matrix A
rank = np.linalg.matrix_rank(A)
print("Rank of the matrix",rank)

# Find the null space of the matrix A
ns = null_space(A)
print("Null space of the matrix",ns)
# Find the dimension of the null space
nullity = ns.shape[1]
print("Null space of the matrix",nullity)
# Verify the rank-nullity theorem
if rank + nullity == A.shape[1]:
    print("Rank-nullity theorem holds.")
else:
    print("Rank-nullity theorem does not hold.")
```

```
Rank of the matrix 2
Null space of the matrix [[-0.40824829]
 [ 0.81649658]
 [-0.40824829]]
Null space of the matrix 1
Rank-nullity theorem holds.
```

4.3 Dimension of Vector Space

Find the dimension of subspace spanned by the vectors $(1, 2, 3), (2, 3, 1)$ and $(3, 1, 2)$.

```
import numpy as np

# Define the vector space V
V = np.array([
    [1, 2, 3],
    [2, 3, 1],
    [3, 1, 2]])
# Find the dimension and basis of V
basis = np.linalg.matrix_rank(V)
dimension = V.shape[0]
print("Basis of the matrix",basis)
print("Dimension of the matrix",dimension)
```

```
Basis of the matrix 3
Dimension of the matrix 3
```

Extract the linearly independent rows in given matrix : Basis of Row space

```
from numpy import *
import sympy as sp
A=[[1,-1,1,1],[2,-5,2,2],[3,-3,5,3],[4,-4,4,4]]
AB=array(A)
S=shape(A)
n=len(A)
for i in range(n):
    if AB[i,i]==0:
        ab=copy(AB)
        for k in range(i+1,S[0]):
            if ab[k,i]!=0:
                ab[i,:]=AB[k,:]
                ab[k,:]=AB[i,:]
                AB=copy(ab)
    for j in range(i+1,n):
        Fact=AB[j,i]/AB[i,i]
        for k in range(i,n):
            AB[j,k]=AB[j,k]-Fact*AB[i,k]
display("REF of given matrix: ",sp.Matrix(AB))
temp = {(0, 0, 0,0)}
result = []
for idx, row in enumerate(map(tuple, AB)):
    if row not in temp:
        result.append(idx)
print("\n Basis are non-zero rows of A:")
display(sp.Matrix(AB[result]))
```

```
'REF of given matrix: '
```

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

```
Basis are non-zero rows of A:
```

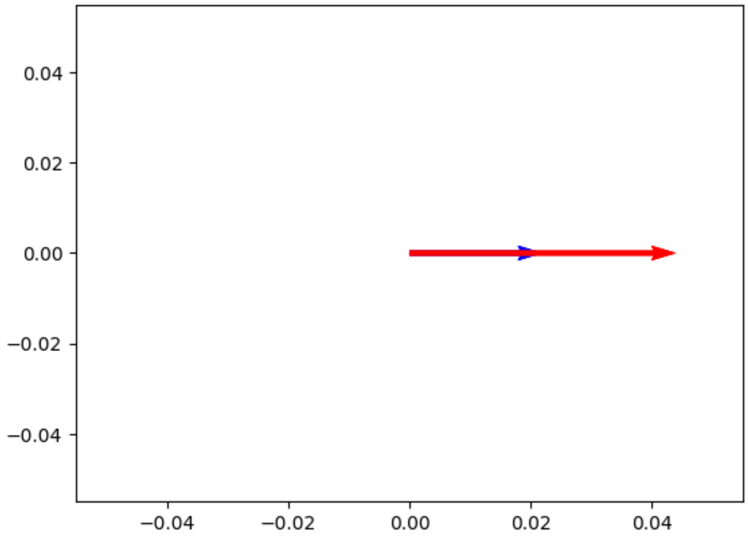
$$\begin{bmatrix} 1 & -1 & 1 & 1 \\ 0 & -3 & 0 & 0 \\ 0 & 0 & 2 & 0 \end{bmatrix}$$

4.4 Graphical representation of a transformation

4.4.1 Horizontal stretch:

Represent the horizontal stretch transformation $T : R^2 ßR^2$ geometrically
Find the image of vector $(10, 0)$ when it is stretched horizontally by 2 units.

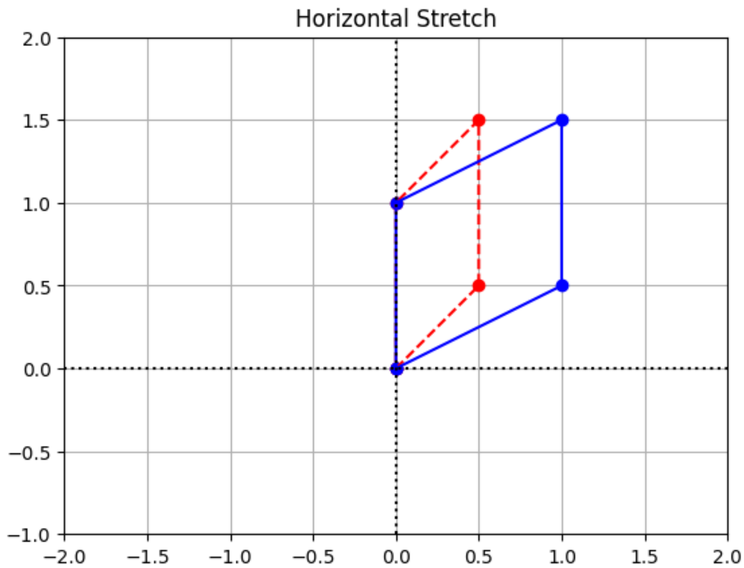
```
import numpy as np
import matplotlib.pyplot as plt
V = np.array([[10,0]])
origin = np.array([[0, 0, 0],[0, 0, 0]]) # origin point
A=np.matrix([[2,0],[0,1]])
V1=np.matrix(V)
V2=A*np.transpose(V1)
V2=np.array(V2)
plt.quiver(*origin, V[:,0], V[:,1], color=['b'], scale=50)
plt.quiver(*origin, V2[0,:], V2[1,:], color=['r'], scale=50)
plt.show()
```



Another example.

```
from math import pi, sin, cos
```

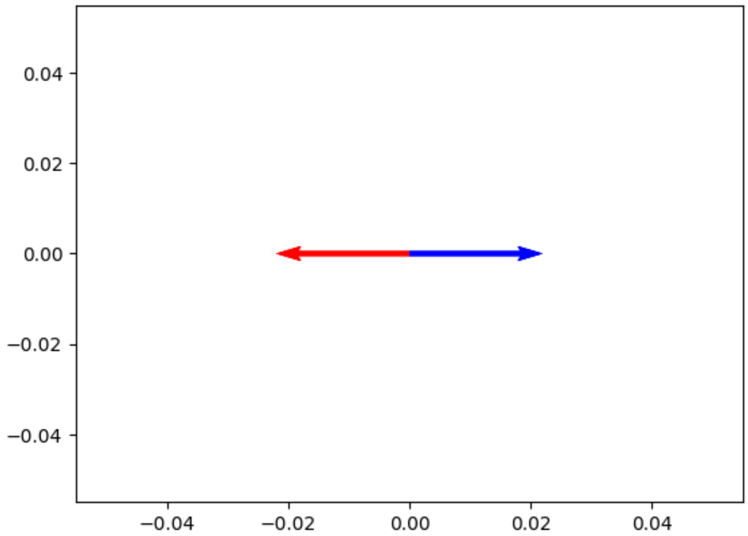
```
import matplotlib.pyplot as plt
import numpy as np

coords = np.array([[0,0],[0.5,0.5],[0.5,1.5],[0,1],[0,0]])
coords = coords.transpose()
coords
x = coords[0,:]
y = coords[1,:]

A = np.array([[2,0],[0,1]])
A_coords = A@coords
x_LT1 = A_coords[0,:]
y_LT1 = A_coords[1,:]

# Create the figure and axes objects
fig, ax = plt.subplots()

# Plot the points.  x and y are original vectors, x_LT1 and y_LT1 are
                                    images
ax.plot(x,y,'ro')
ax.plot(x_LT1,y_LT1,'bo')

# Connect the points by lines
ax.plot(x,y,'r',ls="--")
ax.plot(x_LT1,y_LT1,'b')

# Edit some settings
ax.axvline(x=0,color="k",ls=":")
ax.axhline(y=0,color="k",ls=":")
ax.grid(True)
ax.axis([-2,2,-1,2])
ax.set_aspect('equal')
ax.set_title("Horizontal Stretch");
```



4.4.2 Reflection:

Represent the reflection transformation $T : \mathbb{R}^2 \to \mathbb{R}^2$ geometrically.
Find the image of vector $(10, 0)$ when it is reflected about y axis.

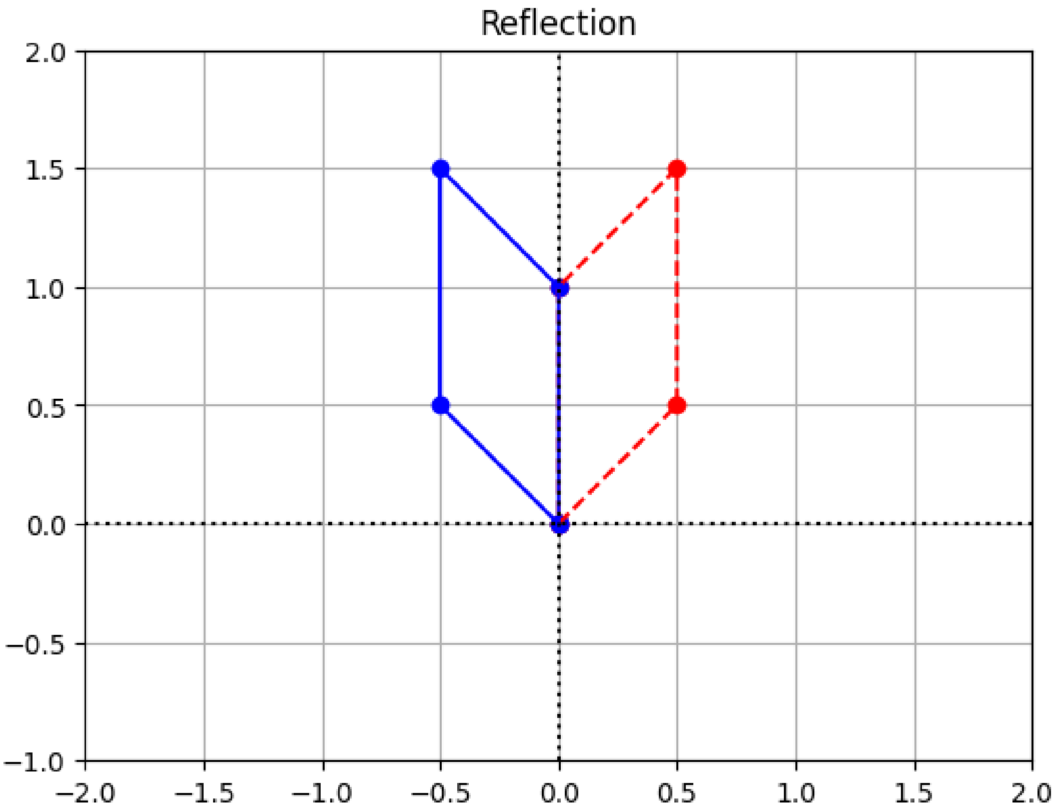
```
import numpy as np
import matplotlib.pyplot as plt
V = np.array([[10,0]])
origin = np.array([[0, 0, 0],[0, 0, 0]]) # origin point
A=np.matrix([[-1,0],[0,1]])
V1=np.matrix(V)
V2=A*np.transpose(V1)
V2=np.array(V2)
plt.quiver(*origin, V[:,0], V[:,1], color=['b'], scale=50)
plt.quiver(*origin, V2[0,:], V2[1,:], color=['r'], scale=50)
plt.show()
```



Another example.

```
B = np.array([[-1,0],[0,1]])
B_coords = B@coords

x_LT2 = B_coords[0,:]
y_LT2 = B_coords[1,:]

# Create the figure and axes objects
fig, ax = plt.subplots()

# Plot the points.  x and y are original vectors, x_LT1 and y_LT1 are
                                  images
ax.plot(x,y,'ro')
ax.plot(x_LT2,y_LT2,'bo')

# Connect the points by lines
ax.plot(x,y,'r',ls="--")
ax.plot(x_LT2,y_LT2,'b')

# Edit some settings
ax.axvline(x=0,color="k",ls=":")
```

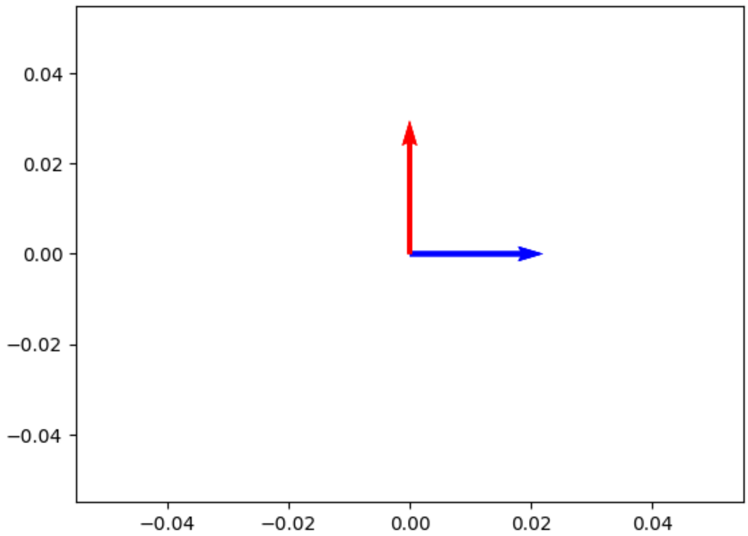
```
ax.axhline(y=0,color="k",ls=":")
ax.grid(True)
ax.axis([-2,2,-1,2])
ax.set_aspect('equal')
ax.set_title("Reflection");
```



4.4.3 Rotation:

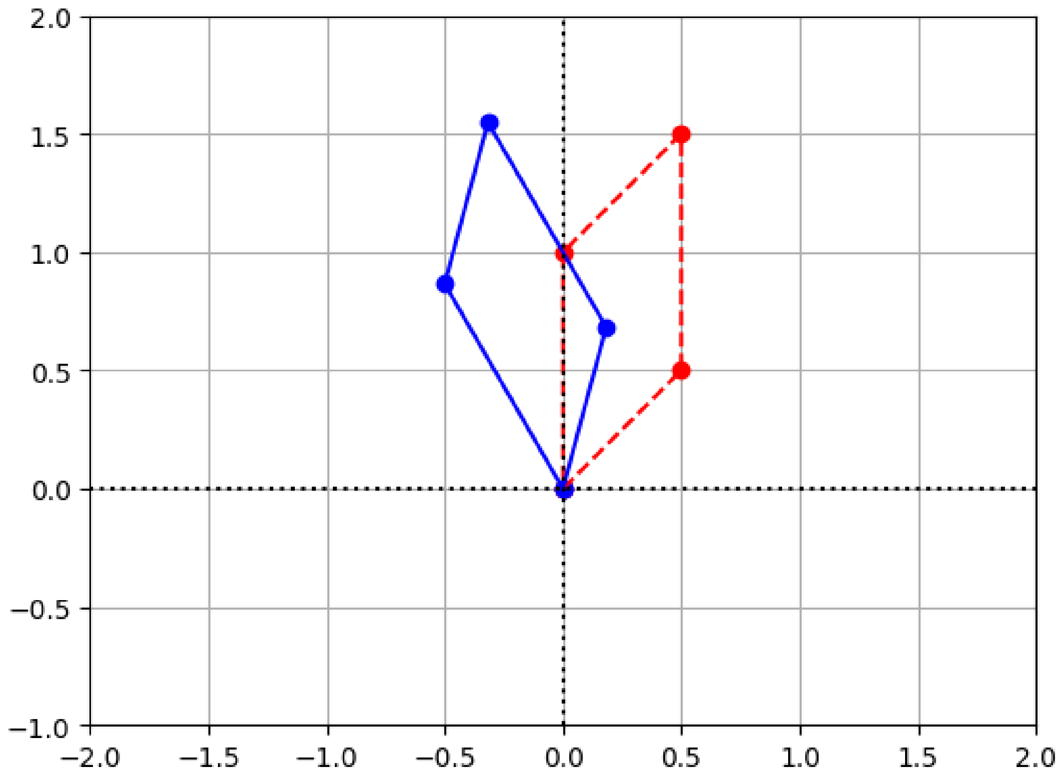
Represent the rotation transformation $T : \mathbb{R}^2 \to \mathbb{R}^2$ geometrically.
Find the image of vector $(10, 0)$ when it is rotated by $\pi/2$ radians.

```
import numpy as np
import matplotlib.pyplot as plt
V = np.array([[10,0]])
origin = np.array([[0, 0, 0],[0, 0, 0]]) # origin point
A=np.matrix([[0,-1],[1,1]])
V1=np.matrix(V)
V2=A*np.transpose(V1)
V2=np.array(V2)
plt.quiver(*origin, V[:,0], V[:,1], color=['b'], scale=50)
plt.quiver(*origin, V2[0,:], V2[1,:], color=['r'], scale=50)
plt.show()
```



Another example.

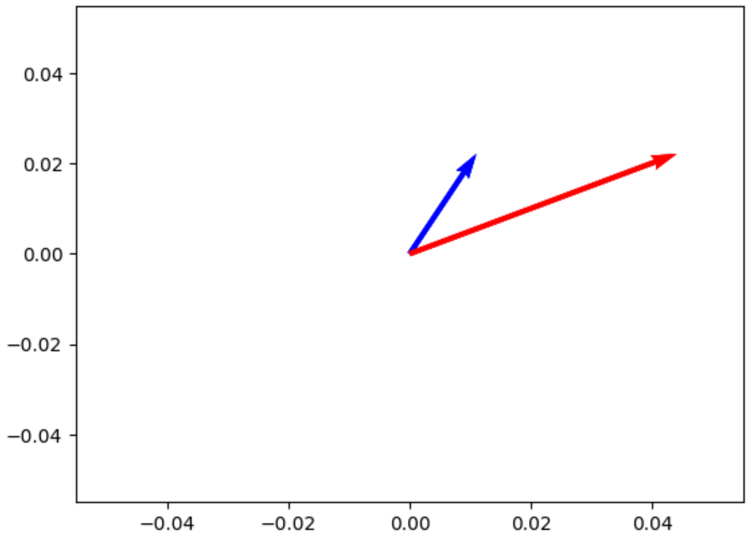
```
theta = pi/6
R = np.array([[cos(theta),-sin(theta)],[sin(theta),cos(theta)]])
R_coords = R@coords

x_LT3 = R_coords[0,:]
y_LT3 = R_coords[1,:]

# Create the figure and axes objects
fig, ax = plt.subplots()

# Plot the points.  x and y are original vectors, x_LT1 and y_LT1 are
                                        images
ax.plot(x,y,'ro')
ax.plot(x_LT3,y_LT3,'bo')

# Connect the points by lines
ax.plot(x,y,'r',ls="--")
ax.plot(x_LT3,y_LT3,'b')

# Edit some settings
ax.axvline(x=0,color="k",ls=":")
ax.axhline(y=0,color="k",ls=":")
ax.grid(True)
ax.axis([-2,2,-1,2])
ax.set_aspect('equal')
```



4.4.4 Shear Transformation

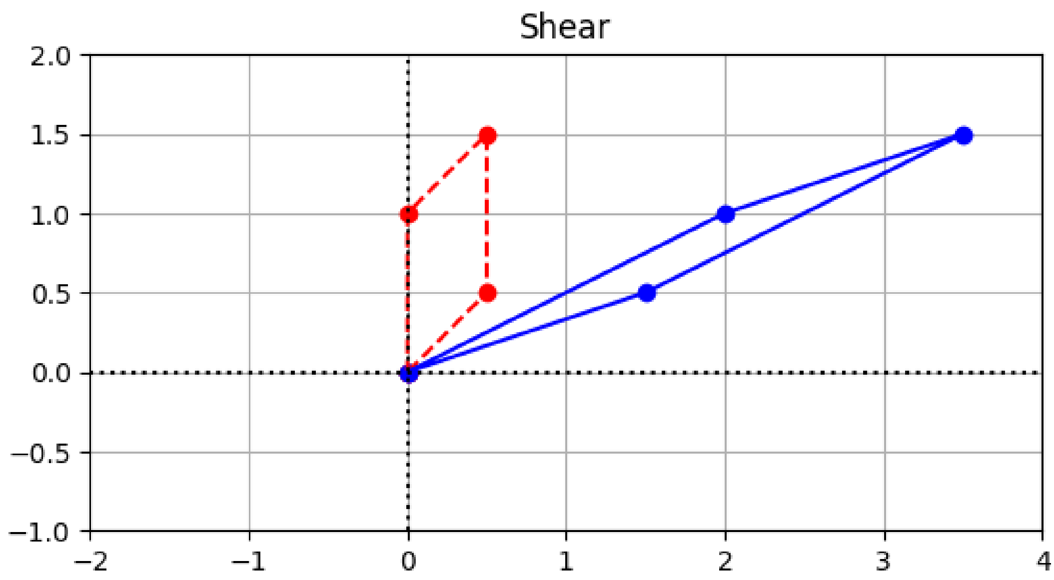
Represent the Shear transformation $T : \mathbb{R}^2 \to \mathbb{R}^2$ geometrically.
Find the image of $(2, 3)$ under shear transformation.

```
import numpy as np
import matplotlib.pyplot as plt
V = np.array([[2,3]])
origin = np.array([[0, 0, 0],[0, 0, 0]]) # origin point
A=np.matrix([[1,2],[0,1]])
V1=np.matrix(V)
V2=A*np.transpose(V1)
V2=np.array(V2)
print("Image of given vectors is:", V2)
plt.quiver(*origin, V[:,0], V[:,1], color=['b'], scale=20)
plt.quiver(*origin, V2[0,:], V2[1,:], color=['r'], scale=20)
plt.show()
```



Another example.

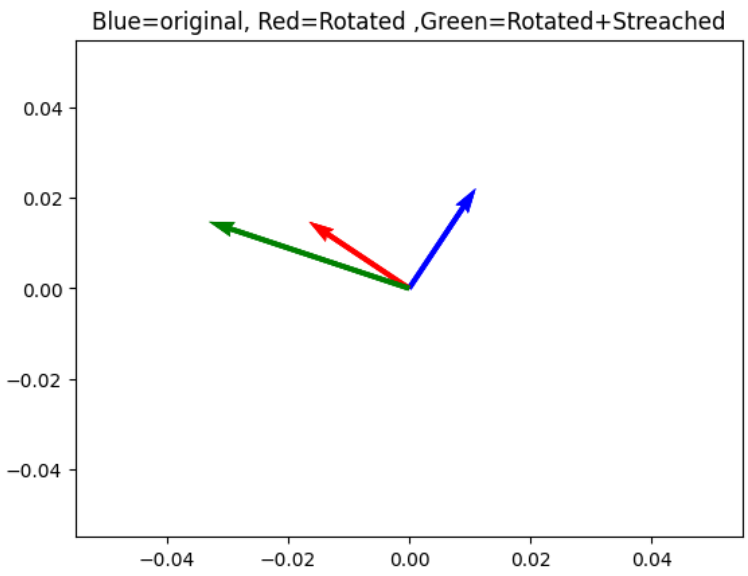
```
S = np.array([[1,2],[0,1]])
S_coords = S@coords

x_LT4 = S_coords[0,:]
y_LT4 = S_coords[1,:]


# Create the figure and axes objects
fig, ax = plt.subplots()

# Plot the points.  x and y are original vectors, x_LT1 and y_LT1 are
                                    images
ax.plot(x,y,'ro')
ax.plot(x_LT4,y_LT4,'bo')

# Connect the points by lines
ax.plot(x,y,'r',ls="--")
ax.plot(x_LT4,y_LT4,'b')

# Edit some settings
ax.axvline(x=0,color="k",ls=":")
ax.axhline(y=0,color="k",ls=":")
ax.grid(True)
ax.axis([-2,4,-1,2])
ax.set_aspect('equal')
ax.set_title("Shear");
```



4.4.5 Composition

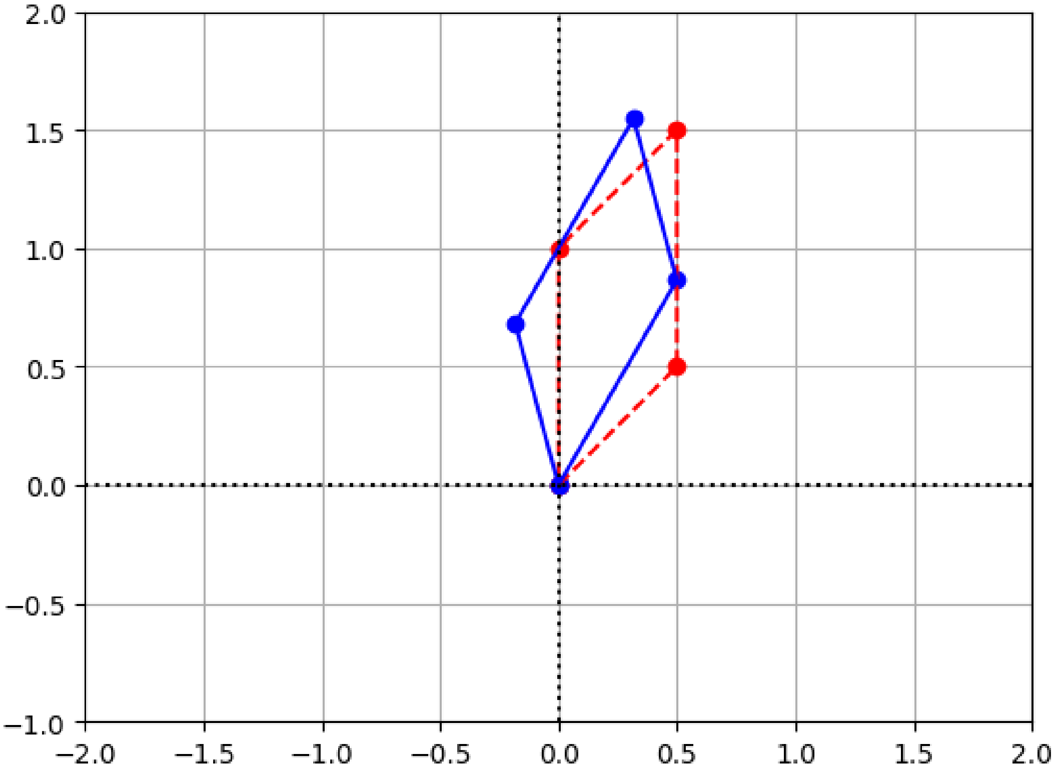
Represent the composition of two 2D transformations.
Find the image of vector $(10, 0)$ when it is rotated by $\pi/2$ radians then stretched horizontally 2 units.

```
import numpy as np
import matplotlib.pyplot as plt
V = np.array([[2,3]])
origin = np.array([[0, 0, 0],[0, 0, 0]]) # origin point
A=np.matrix([[0,-1],[1,0]])
B=np.matrix([[2,0],[0,1]])
V1=np.matrix(V)
V2=A*np.transpose(V1)
V3=B*V2
V2=np.array(V2)
V3=np.array(V3)
print("Image of given vectors is:", V3)
plt.quiver(*origin, V[:,0], V[:,1], color=['b'], scale=20)
plt.quiver(*origin, V2[0,:], V2[1,:], color=['r'], scale=20)
plt.quiver(*origin, V3[0,:], V3[1,:], color=['g'], scale=20)
plt.title('Blue=original, Red=Rotated ,Green=Rotated+Streached')
plt.show()
```



Another example.

```
C = np.array([[-cos(theta),sin(theta)],[sin(theta),cos(theta)]])
C_coords = C@coords

x_LT5 = C_coords[0,:]
y_LT5 = C_coords[1,:]


# Create the figure and axes objects
fig, ax = plt.subplots()

# Plot the points.  x and y are original vectors, x_LT1 and y_LT1 are
                                    images
ax.plot(x,y,'ro')
ax.plot(x_LT5,y_LT5,'bo')

# Connect the points by lines
ax.plot(x,y,'r',ls="--")
ax.plot(x_LT5,y_LT5,'b')

# Edit some settings
ax.axvline(x=0,color="k",ls=":")
ax.axhline(y=0,color="k",ls=":")
ax.grid(True)
ax.axis([-2,2,-1,2])
ax.set_aspect('equal')
```



4.5 Exercise:

1. Verify the rank nullity theorem for the following linear transformation
   a) $T : \mathbb{R}^2 \to \mathbb{R}^3$ defined by $T(x, y) = (x + 4y, 2x + 5y, 3x + 6y)$.
   Ans: Rank=2, Nullity=1, RNT verified
   b) $T : \mathbb{R}^3 \to \mathbb{R}^4$ defined by $T(x, y, z) = (x+4y-z, 2x+5y+8z, 3x+y+2z, x+y+z)$.
   Ans: Rank=3, Nullity=1, RNT verified
2. Find the dimension of the subspace spanned following set of vectors
   a) $S = (1, 2, 3, 4), (2, 4, 6, 8), (1, 1, 1, 1)$
   Ans: Dimension of subspace is 2
   b) $S = (1, -1, 3, 4), (2, 1, 6, 8), (1, 1, 1, 1), (3, 3, 3, 3)$
   Ans: Dimension of subspace is 3
3. Find the image of $(1, 3)$ under following $2D$ transformations
   a) Horizontal stretch
   b) Reflection
   c) Shear
   d) Rotation

LAB 5: Computing the inner product and orthogonality

5.1 Objectives:

Use python

1. to compute the inner product of two vectors.
2. to check whether the given vectors are orthogonal.

5.2 Inner Product of two vectors

Find the inner product of the vectors $(2, 1, 5, 4)$ and $(3, 4, 7, 8)$.

```
import numpy as np

#initialize arrays
A = np.array([2, 1, 5, 4])
B = np.array([3, 4, 7, 8])

#dot product
output = np.dot(A, B)

print(output)
```

```
77
```

5.3 Checking orthogonality

Verify whether the following vectors $(2, 1, 5, 4)$ and $(3, 4, 7, 8)$ are orthogonal.

```
import numpy as np

#initialize arrays
A = np.array([2, 1, 5, 4])
B = np.array([3, 4, 7, 8])

#dot product
output = np.dot(A, B)
print('Inner product is :',output)
if output==0:
  print('given vectors are orthognal ')
else:
  print('given vectors are not orthognal ')
```

```
Inner product is : 77
given vectors are not orthognal
```

5.4 Exercise:

1. Find the inner product of $(1, 2, 3)$ and $(3, 4, 5)$.

   Ans: 26

2. Find the inner product of $(1, -1, 2, 1)$ and $(4, 2, 1, 0)$.

   Ans: 4

3. Check whether the following vectors are orthogonal or not

   a) $(1, 1, -1)$ and $(2, 3, 5)$. Ans: True

   b) $(1, 0, 2, 0)$ and $(4, 2, -2, 5)$. Ans: True

   c) $(1, 2, 3, 4)$ and $(2, 3, 4, 5)$. Ans: False

LAB 6: Solution of algebraic and transcendental equation by Regula-Falsi and Newton-Raphson method

6.1 Objectives:

Use python

1. to solve algebraic and transcendental equation by Regula-Falsi method.
2. to solve algebraic and transcendental equation by Newton-Raphson method.

6.2 Regula-Falsi method to solve a transcendental equation

Obtain a root of the equation $x^3 - 2x - 5 = 0$ between 2 and 3 by regula-falsi method. Perform 5 iterations.

```
# Regula Falsi method
from sympy import *
x=Symbol('x')
g =input('Enter the function  ')  #%x^3-2*x-5;    %function
f=lambdify(x,g)
a=float(input('Enter a valus :')) #2
b=float(input('Enter b valus :')) # 3
N=int(input('Enter number of iterations :')) #5

for i in range(1,N+1):
    c=(a*f(b)-b*f(a))/(f(b)-f(a))
    if((f(a)*f(c)<0)):
        b=c
    else:
        a=c
    print('itration %d  \t  the root %0.3f \t function value %0.3f \n'%
                                        (i,c,f(c)));
```

```
Enter the function  x**3-2*x-5
Enter a valus :2
Enter b valus :3
Enter number of iterations :5
itration 1         the root 2.059        function value -0.391

itration 2         the root 2.081        function value -0.147

itration 3         the root 2.090        function value -0.055

itration 4         the root 2.093        function value -0.020

itration 5         the root 2.094        function value -0.007
```

Using tolerance value we can write the same program as follows:
Obtain a root of the equation $x^3 - 2x - 5 = 0$ between 2 and 3 by regula-falsi method. Correct to 3 decimal places.

```
# Regula Falsi method while loop2
from sympy import *
x=Symbol('x')
g =input('Enter the function  ')  #%x^3-2*x-5;    %function
f=lambdify(x,g)
a=float(input('Enter a valus :')) # 2
b=float(input('Enter b valus :')) # 3
N=float(input('Enter tolarence  :')) # 0.001
x=a;
c=b;
i=0
while (abs(x-c)>=N):
    x=c
    c=((a*f(b)-b*f(a))/(f(b)-f(a)));
    if((f(a)*f(c)<0)):
        b=c
    else:
        a=c
        i=i+1
    print('itration %d  \t  the root %0.3f \t function value %0.3f \n'%
                                    (i,c,f(c)));
print('final value of the root is %0.5f'%c)
```

```
Enter the function  x**3-2*x-5
Enter a valus :2
Enter b valus :3
Enter tolarence  :0.001
itration 1        the root 2.059        function value -0.391

itration 2        the root 2.081        function value -0.147

itration 3        the root 2.090        function value -0.055

itration 4        the root 2.093        function value -0.020

itration 5        the root 2.094        function value -0.007

itration 6        the root 2.094        function value -0.003

final value of the root is 2.09431
```

6.3 Newton-Raphson method to solve a transcendental equation

Find a root of the equation $3x = \cos x + 1$, near 1, by Newton Raphson method. Perform 5 iterations

```
from sympy import *
x=Symbol('x')
g =input('Enter the function  ')  #%3x-cos(x)-1;    %function
f=lambdify(x,g)
dg = diff(g);
```

```
df=lambdify(x,dg)
x0= float(input('Enter the intial approximation  ')); # x0=1
n= int(input('Enter the number of iterations  '));   #n=5;
for i in range(1,n+1):
    x1 =(x0 - (f(x0)/df(x0)))
    print('itration %d  \t  the root %0.3f \t function value %0.3f \n'%
                                    (i, x1,f(x1))); #print all
                                    iteration value
    x0 = x1
```

```
Enter the function  3*x-cos(x)-1
Enter the intial approximation  1
Enter the number of iterations  5
itration 1        the root 0.620        function value 0.046

itration 2        the root 0.607        function value 0.000

itration 3        the root 0.607        function value 0.000

itration 4        the root 0.607        function value 0.000

itration 5        the root 0.607        function value 0.000
```

6.4 Exercise:

1. Find a root of the equation $3x = \cos x + 1$, between 0 and 1, by Regula-falsi method. Perform 5 iterations.

   Ans: 0.607

2. Find a root of the equation $xe^x = 2$, between 0 and 1, by Regula-falsi method. Correct to 3 decimal places.

   Ans: 0.853

3. Obtain a real positive root of $x^4 - x = 0$, near 1, by Newton-Raphson method. Perform 4 iterations.

   Ans: 1.856

4. Obtain a real positive root of $x^4 + x^3 - 7x^2 - x + 5 = 0$, near 3, by Newton-Raphson method. Perform 7 iterations.

   Ans: 2.061

LAB 7: Interpolation /Extrapolation using Newton's forward and backward difference formula

7.1 Objectives:

Use python

1. to interpolate using Newton's Forward interpolation method.
2. to interpolate using Newton's backward interpolation method.
3. to extrapolate using Newton's backward interpolation method.

1. Use Newtons forward interpolation to obtain the interpolating polynomial and hence calculate y(2) for the following:

x:	1	3	5	7	9
y:	6	10	62	210	502

```
from sympy import *
import numpy as np
n = int(input('Enter number of data points: '))
210
x = np.zeros((n))
y = np.zeros((n,n))


# Reading data points
print('Enter data for x and y: ')
for i in range(n):
    x[i] = float(input( 'x['+str(i)+']='))
    y[i][0] = float(input( 'y['+str(i)+']='))

# Generating forward difference table
for i in range(1,n):
    for j in range(0,n-i):
        y[j][i] = y[j+1][i-1] - y[j][i-1]

print('\nFORWARD DIFFERENCE TABLE\n');

for i in range(0,n):
    print('%0.2f' %(x[i]), end='')
    for j in range(0, n-i):
        print('\t\t%0.2f' %(y[i][j]), end='')
    print()
 # obtaining the polynomial
t=symbols('t')
f=[] # f is a list type data


p=(t-x[0])/(x[1]-x[0])
f.append(p)
for i in range(1,n-1):
    f.append(f[i-1]*(p-i)/(i+1))
    poly=y[0][0]
for i in range(n-1):
    poly=poly+y[0][i+1]*f[i]
```

```
simp_poly=simplify(poly)
print('\nTHE INTERPOLATING POLYNOMIAL IS\n');
pprint(simp_poly)
# if you want to interpolate at some point the next session will help
inter=input('Do you want to interpolate at a point(y/n)? ') # y
if inter=='y':
    a=float(input('enter the point ')) #2
    interpol=lambdify(t,simp_poly)
    result=interpol(a)
    print('\nThe  value of the function at' ,a,'is\n',result);
```

```
Enter number of data points: 5
Enter data for x and y:
x[0]=1
y[0]=6
x[1]=3
y[1]=10
x[2]=5
y[2]=62
x[3]=7
y[3]=210
x[4]=9
y[4]=502

FORWARD DIFFERENCE TABLE

1.00            6.00            4.00            48.00           48.00           0.00
3.00            10.00           52.00           96.00           48.00
5.00            62.00           148.00          144.00
7.00            210.00          292.00
9.00            502.00

THE INTERPOLATING POLYNOMIAL IS

     3        2
1.0·t  - 3.0·t  + 1.0·t + 7.0
Do you want to interpolate at a point(y/n)? y
enter the point 2

The  value of the function at 2.0 is
 5.0
```

2. Use Newtons backward interpolation to obtain the interpolating polynomial and hence calculate y(8) for the following data:

x:	1	3	5	7	9
y:	6	10	62	210	502

```
from sympy import *
import numpy as np
import sys
print("This will use Newton's backword intepolation formula ")
# Reading number of unknowns
n = int(input('Enter number of data points: '))

# Making numpy array of n & n x n size and initializing
# to zero for storing x and y value along with differences of y
x = np.zeros((n))
y = np.zeros((n,n))


# Reading data points
```

```
print('Enter data for x and y: ')
for i in range(n):
    x[i] = float(input( 'x['+str(i)+']='))
    y[i][0] = float(input( 'y['+str(i)+']='))

# Generating backward difference table
for i in range(1,n):
    for j in range(n-1,i-2,-1):
        y[j][i] = y[j][i-1] - y[j-1][i-1]


print('\nBACKWARD DIFFERENCE TABLE\n');

for i in range(0,n):
    print('%0.2f' %(x[i]), end='')
    for j in range(0, i+1):
        print('\t%0.2f' %(y[i][j]), end='')
    print()


# obtaining the polynomial
t=symbols('t')
f=[]


p=(t-x[n-1])/(x[1]-x[0])
f.append(p)
for i in range(1,n-1):
       f.append(f[i-1]*(p+i)/(i+1))

poly=y[n-1][0]
print(poly)
for i in range(n-1):
       poly=poly+y[n-1][i+1]*f[i]
       simp_poly=simplify(poly)
print('\nTHE INTERPOLATING POLYNOMIAL IS\n');
pprint(simp_poly)
# if you want to interpolate at some point the next session will help
inter=input('Do you want to interpolate at a point(y/n)? ')
if inter=='y':
       a=float(input('enter the point '))
       interpol=lambdify(t,simp_poly)
       result=interpol(a)
       print('\nThe  value of the function at' ,a,'is\n',result);
```

```
This will use Newton's backword intepolation formula
Enter number of data points: 5
Enter data for x and y:
x[0]=1
y[0]=6
x[1]=3
y[1]=10
x[2]=5
y[2]=62
x[3]=7
y[3]=210
x[4]=9
y[4]=502

BACKWARD DIFFERENCE TABLE

1.00    6.00
3.00    10.00   4.00
5.00    62.00   52.00   48.00
7.00    210.00  148.00  96.00   48.00
9.00    502.00  292.00  144.00  48.00   0.00
502.0

THE INTERPOLATING POLYNOMIAL IS

     3        2
1.0·t  - 3.0·t  + 1.0·t + 7.0
Do you want to interpolate at a point(y/n)? y
enter the point 8

The  value of the function at 8.0 is
 335.0
```

7.2 Exercise:

1. Obtain the interpolating polynomial for the following data

x:	0	1	2	3
y:	1	2	1	10

   Ans: $2x^3 - 7x^2 + 6x + 1$

2. Find the number of men getting wage Rs. 100 from the following table:

wage:	50	150	250	350
No. of men:	9	30	35	42

   Ans: 23 men

3. Using Newton's backward interpolation method obtain y(160) for the following data

x :	100	150	200	250	300
y :	10	13	15	17	18

   Ans: 13.42

4. Using Newtons forward interpolation polynomial and calculate y(1) and y(10).

x :	3	4	5	6	7	8	9
y :	4.8	8.4	14.5	23.6	36.2	52.8	73.9

   Ans: 3.1 and 100

LAB 8: Computation of area under the curve using Trapezoidal, Simpson's $\left(\frac{1}{3}\right)^{\text{rd}}$ and Simpsons $\left(\frac{3}{8}\right)^{\text{th}}$ rule

8.1 Objectives:

Use python

1. to find area under the curve represented by a given function using Trapezoidal rule.
2. to find area under the curve represented by a given function using Simpson's $\left(\frac{1}{3}\right)^{\text{rd}}$ rule.
3. to find area under the curve represented by a given function using Simpson's $\left(\frac{3}{8}\right)^{\text{th}}$ rule.
4. to find the area below the curve when discrete points on the curve are given.

8.2 Trapezoidal Rule

Evaluate $\int_0^5 \frac{1}{1+x^2}$.

```python
# Definition of the function to integrate
def my_func(x):
    return 1 / (1 + x ** 2)
```

```python
# Function to implement trapezoidal method
def trapezoidal(x0, xn, n):
  h = (xn - x0) / n                               # Calculating step
                                                  size
  # Finding sum
  integration = my_func(x0) + my_func(xn)         # Adding first and
                                                  last terms
  for i in range(1, n):
    k = x0 + i * h                                # i-th step value
    integration = integration + 2 * my_func(k)    # Adding areas of the
                                                  trapezoids
  # Proportioning sum of trapezoid areas
  integration = integration * h / 2
  return integration
```

```python
# Input section
lower_limit = float(input("Enter lower limit of integration: "))
upper_limit = float(input("Enter upper limit of integration: "))
sub_interval = int(input("Enter number of sub intervals: "))

# Call trapezoidal() method and get result
result = trapezoidal(lower_limit, upper_limit, sub_interval)

# Print result
print("Integration result by Trapezoidal method is: " , result)
```

```
Enter lower limit of integration: 0
Enter upper limit of integration: 5
Enter number of sub intervals: 10
Integration result by Trapezoidal method is:  1.3731040812301099
```

8.3 Simpson's $\left(\frac{1}{3}\right)^{\text{rd}}$ Rule

Evaluate $\int_0^5 \frac{1}{1+x^2}$.

```
# Definition of the function to integrate
def my_func(x):
    return 1 / (1 + x ** 2)
```

```
# Function to implement the Simpson's one-third rule

def simpson13(x0,xn,n):
  h = (xn - x0) / n                    # calculating step size
  # Finding sum
  integration = (my_func(x0) + my_func(xn))
  k = x0
  for i in range(1,n):
    if i%2 == 0:
      integration = integration + 4 * my_func(k)
    else:
      integration = integration + 2 * my_func(k)
    k += h
  # Finding final integration value
  integration = integration * h * (1/3)
  return integration

# Input section
lower_limit = float(input("Enter lower limit of integration: "))
upper_limit = float(input("Enter upper limit of integration: "))
sub_interval = int(input("Enter number of sub intervals: "))

# Call trapezoidal() method and get result
result = simpson13(lower_limit, upper_limit, sub_interval)
print("Integration result by Simpson's 1/3 method is: %0.6f" % (result)
                                         )
```

```
Enter lower limit of integration: 0
Enter upper limit of integration: 5
Enter number of sub intervals: 100
Integration result by Simpson's 1/3 method is: 1.404120
```

8.4 Simpson's 3/8th rule

Evaluate $\int_0^6 \frac{1}{1+x^2}dx$ using Simpson's 3/8 th rule, taking 6 sub intervals

```
def simpsons_3_8_rule(f, a, b, n):
```

```
  h = (b - a) / n
  s = f(a) + f(b)
  for i in range(1, n, 3):
    s += 3 * f(a + i * h)
  for i in range(3, n-1, 3):
    s += 3 * f(a + i * h)
  for i in range(2, n-2, 3):
    s += 2 * f(a + i * h)
  return s * 3 * h / 8

def f(x):
  return 1/(1+x**2) # function here

a = 0   # lower limit
b = 6 #  upper limit
n = 6 # number of sub intervals

result = simpsons_3_8_rule(f, a, b, n)
print('%3.5f'%result)
```

```
1.27631
```

8.5 Exercise:

1. Evaluate the integral $\int_0^1 \frac{x^2}{1+x^3}dx$ using Simpson's $\frac{1}{3}$ rule.

   Ans: 0.23108

2. Use Simpson's $\frac{3}{8}$ rule to find $\int_0^{0.6} e^{-x^2}dx$ by taking seven ordinates.

   Ans: 0.5351

3. Evaluate using trapezoidal rule $\int_0^{\pi} sin^2 x dx$. Take $n = 6$.

   Ans: $\pi/2$

4. A solid of revolution is formed by rotating about the x-axis, the area between the x-axis, the lines $x = 0$ and $x = 1$, and a curve through the points with the following co-ordinates:

x	y
0.00	1.0000
0.25	0.9896
0.50	0.9589
0.75	0.9089
1.00	0.8415

Estimate the volume of the solid formed using Simpson's $\frac{1}{3}$rd rule. Hint: Required volume is $\int_0^1 y^2 * \pi dx$. **[Ans: 2.8192]**

5. The velocity v(km/min) of a moped which starts from rest, is given at fixed intervals of time t(min) as follows:

t:	2	4	6	8	10	12	14	16	18	20
v:	10	18	25	29	32	20	11	5	2	0

Estimate approximately the distance covered in twenty minutes.

Answer for 5.

We know that ds/dt=v. So to get distance (s) we have to integrate.
Here $h = 2.2$, $v_0 = 0$, $v_1 = 10$, $v_2 = 18$, $v_3 = 25$ etc.

```
# we shall use simpson's 1/3 rule directly to estimate

h=2
y= [0, 10 ,18, 25, 29,32 ,20, 11 ,5 ,2 , 0]
result=(h/3)*((y[0]+y[10])+4*(y[1]+y[3]+y[5]+y[7]+y[9])+2*(y[2]+y[4]+y[
                                          6]+y[8]))

print('%3.5f'%result,'km.')
```

```
309.33333 km.
```

LAB 9: Solution of ODE of first order and first degree by Taylor's series and Modified Euler's method

9.1 Objectives:

Use python

1. to solve ODE by Taylor series method.
2. to solve ODE by Modified Euler method.
3. to trace the solution curves.

9.2 Taylor series method to solve ODE

Solve: $\frac{dy}{dx} - 2y = 3e^x$ with $y(0) = 0$ using Taylor series method at $x = 0.1(0.1)0.3$.

```
## module taylor
'''X,Y = taylor(deriv,x,y,xStop,h).
4th-order Taylor series method for solving the initial value problem {y
                                            }' = {F(x,{y})}, where
{y} = {y[0],y[1],...y[n-1]}.
x,y = initial conditions
xStop = terminal value of x
h = increment of x
'''
from numpy import array
def taylor(deriv,x,y,xStop,h):
    X = []
    Y = []
    X.append(x)
    Y.append(y)
    while x < xStop:                    # Loop over integration steps
        D = deriv(x,y)                  # Derivatives of y
        H = 1.0
        for j in range(3):              # Build Taylor series
            H = H*h/(j + 1)
            y = y + D[j]*H              # H = h^j/j!
        x = x + h
        X.append(x) # Append results to
        Y.append(y) # lists X and Y

    return array(X),array(Y) # Convert lists into arrays

# deriv = user-supplied function that returns derivatives in the 4 x n
                                            array
'''
[y'[0] y'[1] y'[2] ... y'[n-1]
y''[0] y''[1] y''[2] ... y''[n-1]
y'''[0] y'''[1] y'''[2] ... y'''[n-1]
y''''[0] y''''[1] y''''[2] ... y''''[n-1]]
'''
def deriv(x,y):
    D = zeros((4,1))
```

```
    D[0] = [2*y[0] + 3*exp(x)]
    D[1] = [4*y[0]+ 9*exp(x)]
    D[2] = [8*y[0]+ 21*exp(x)]
    D[3] = [16*y[0]+ 45*exp(x)]
    return D

x = 0.0         # Initial value of x
xStop = 0.3       # last value
y = array([0.0])           # Initial values of y
h = 0.1               # Step size
X,Y = taylor(deriv,x,y,xStop,h)

print("The required values are :at x= %0.2f, y=%0.5f, x=%0.2f, y=%0.5f,
                                     x = %0.2f, y=%0.5f,x = %0.2f, y=%0
                                     .5f"%(X[0],Y[0],X[1],Y[1],X[2],Y[2]
                                     ,X[3],Y[3] ))
```

```
The required values are :at x= 0.00, y=0.00000, x=0.10, y=0.34850,
x = 0.20, y=0.81079,x = 0.30, y=1.41590
```

Solve $y' + 4y = x^2$ with initial conditions $y(0) = 1$ using Taylor series method at $x = 0.1, 0.2$.

```
from numpy import array
def taylor(deriv,x,y,xStop,h):
    X = []
    Y = []
    X.append(x)
    Y.append(y)
    while x < xStop:                  # Loop over integration steps
        D = deriv(x,y)               # Derivatives of y
        H = 1.0
        for j in range(3):           # Build Taylor series
            H = H*h/(j + 1)
            y = y + D[j]*H           # H = h^j/j!
        x = x + h
        X.append(x) # Append results to
        Y.append(y) # lists X and Y

    return array(X),array(Y) # Convert lists into arrays

# deriv = user-supplied function that returns derivatives in the 4 x n
                                     array
'''
[y'[0] y'[1] y'[2] ... y'[n-1]
y"[0] y"[1] y"[2] ... y"[n-1]
y'''[0] y'''[1] y'''[2] ... y'''[n-1]
y""[0] y""[1] y""[2] ... y""[n-1]]
'''
def deriv(x,y):
    D = zeros((4,1))
    D[0] = [x**2-4*y[0]]
    D[1] = [2*x-4*x**2+16*y[0]]
    D[2] = [2-8*x+16*x**2-64*y[0]]
    D[3] = [-8+32*x-64*x**2+256*y[0]]
```

```
    return D

x = 0.0         # Initial value of x
xStop = 0.2       # last value
y = array([1.0])           # Initial values of y
h = 0.1               # Step size
X,Y = taylor(deriv,x,y,xStop,h)

print("The required values are :at x= %0.2f, y=%0.5f, x=%0.2f, y=%0.5f,
                                    x = %0.2f, y=%0.5f"%(X[0],Y[0],X[1
                                   ],Y[1],X[2],Y[2]))
```

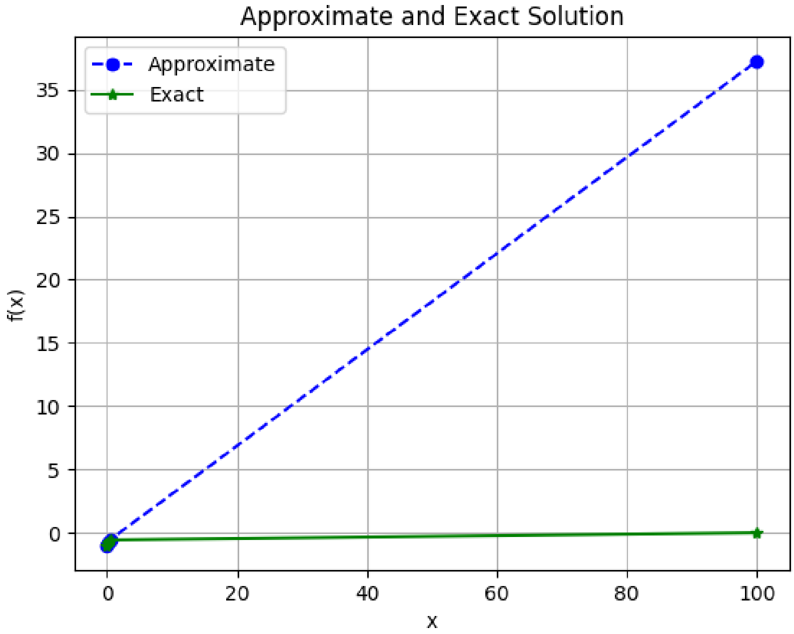
```
The required values are :at x= 0.00, y=1.00000, x=0.10, y=0.66967,
x = 0.20, y=0.45026
```

9.3 Euler's method to solve ODE:

To solve the ODE of the form $\frac{dy}{dx} = f(x, y)$ with initial conditions $y(x_0) = y_0$. The iterative formula is given by : $y(x_{(i+1)} = y(x_i) + hf(x_i, y(x_i))$.
Solve: $y' = e^{-x}$ with $y(0) = -1$ using Euler's method at $x = 0.2(0.2)0.6$.

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

# Define parameters
f = lambda x, y: np.exp(-x) # ODE
h = 0.2 # Step size
y0 = -1 # Initial Condition
n=3
# Explicit Euler Method

y[0] = y0
x[0]=0

for i in range(0, n):
    x[i+1]=x[i]+h
    y[i + 1] = y[i] + h*f(x[i], y[i])

print("The required values are at x= %0.2f, y=%0.5f, x=%0.2f, y=%0.5f,
                                    x = %0.2f, y=%0.5f,x = %0.2f, y=%0.
                                    5f"%(x[0],y[0],x[1],y[1],x[2],y[2],
                                     x[3],y[3]))
print("\n\n")

plt.plot(x, y, 'bo--', label='Approximate')
plt.plot(x, -np.exp(-x), 'g*-', label='Exact')
plt.title("Approximate and Exact Solution" )
plt.xlabel('x')
plt.ylabel('f(x)')
plt.grid()
plt.legend(loc='best')
plt.show()
```

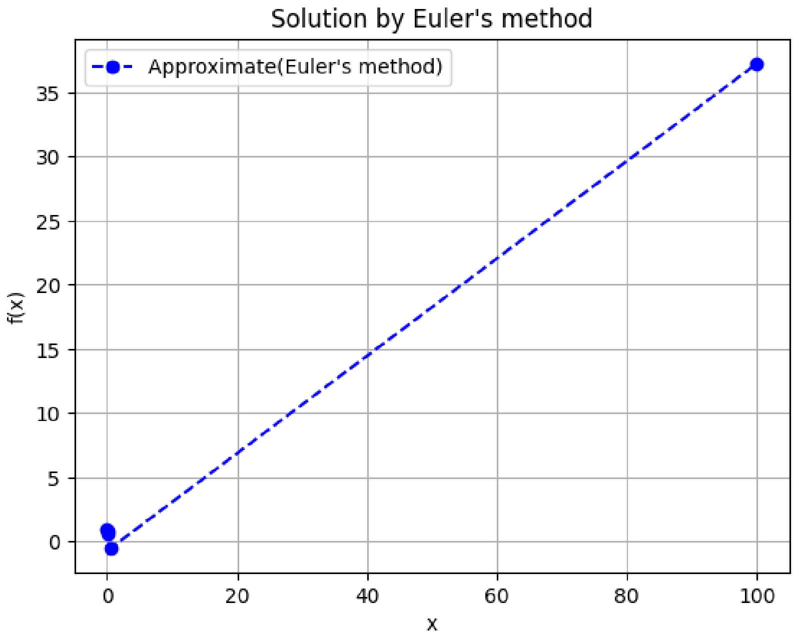
```
The required values are at x= 0.00, y=-1.00000, x=0.20, y=-0.80000,
x = 0.40, y=-0.63625,x = 0.60, y=-0.50219
```



Solve: $y' = -2y + x^3 e^{-2x}$ with $y(0) = 1$ using Euler's method at $x = 0.1, 0.2$.

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

# Define parameters
f = lambda x, y: -2*y+(x**3)*np.exp(-2*x) # ODE
h = 0.1 # Step size
y0 = 1 # Initial Condition
n=2
# Explicit Euler Method

y[0] = y0
x[0]=0

for i in range(0, n):
    x[i+1]=x[i]+h
    y[i + 1] = y[i] + h*f(x[i], y[i])


print("The required values are at x= %0.2f, y=%0.5f, x=%0.2f, y=%0.5f,x
        =%0.2f, y=%0.5f\n\n"%(x[0],y[0],x[1
        ],y[1],x[2],y[2]))

plt.plot(x, y, 'bo--', label="Approximate(Euler's method)")

plt.title("Solution by Euler's method")
plt.xlabel('x')
plt.ylabel('f(x)')
plt.grid()
plt.legend(loc='best')
plt.show()
```

```
The required values are at x= 0.00, y=1.00000, x=0.10, y=0.80000,
x=0.20, y=0.64008
```



9.4 Modified Euler's method

The iterative formula is:

$$y_1^{(n+1)} = y_0 + \frac{h}{2}[f(x_0, y_0) + f(x_1, y_1^{(n)})], \qquad n = 0, 1, 2, 3, \dots,$$

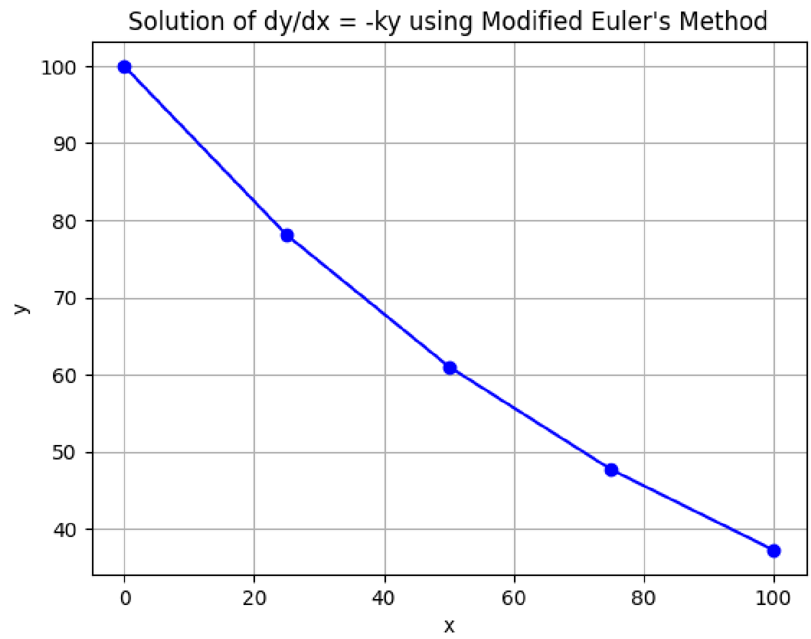
where, $y_1^{(n)}$ is the n^{th} approximation to y_1.

The first iteration will use Euler's method: $y_1^{(0)} = y_0 + hf(x_0, y_0)$.
Solve $y' = -ky$ with $y(0) = 100$ using modified Euler's method at $x = 100$, by taking $h = 25$.

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

def modified_euler(f, x0, y0, h, n):
    x = np.zeros(n+1)
    y = np.zeros(n+1)

    x[0] = x0
    y[0] = y0

    for i in range(n):
        x[i+1] = x[i] + h
        k1 = h * f(x[i], y[i])
        k2 = h * f(x[i+1], y[i] + k1)
        y[i+1] = y[i] + 0.5 * (k1 + k2)

    return x, y
```

```
def f(x, y):
    return -0.01 * y          # ODE dy/dx = -ky

x0 = 0.0
y0 = 100.0
h = 25
n = 4

x, y = modified_euler(f, x0, y0, h, n)

print("The required value at x= %0.2f, y=%0.5f"%(x[4],y[4]))
print("\n\n")

# Plotting the results
plt.plot(x, y, 'bo-')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Solution of dy/dx = -ky using Modified Euler\'s Method')
plt.grid(True)
plt.show()
```

```
The required value at x= 100.00, y=37.25290
```



9.5 Exercise:

1. Find $y(0.1)$ by Taylor Series exapnsion when $y' = x - y^2, y(0) = 1$.

   Ans: $y(0.1) = 0.9138$

2. Find $y(0.2)$ by Taylor Series exapnsion when $y' = x^2y - 1, y(0) = 1, h = 0.1$.

   Ans: $y(0.2) = 0.80227$

3. Evaluate by modified Euler's method: $y' = ln(x+y), y(0) = 2$ at $x = 0(0.2)0.8$.
   Ans: $2.0656, 2.1416, 2.2272, 2.3217$
4. Solve by modified Euler's method: $y' = x + y, y(0) = 1, h = 0.1, x = 0(0.1)0.3$.
   Ans: $1.1105, 1.2432, 1.4004$

LAB 10: Solution of ODE of first order and first degree by Runge-Kutta 4th order method and Milne's predictor and corrector method

10.1 Objectives:

1. To write a python program to solve first order differential equation using 4th order Runge Kutta method.

2. To write a python program to solve first order differential equation using Milne's predictor and corrector method.

10.2 Runge-Kutta method

Apply the Runge Kutta method to find the solution of $dy/dx = 1 + (y/x)$ at $y(2)$ taking $h = 0.2$. Given that $y(1) = 2$.

```
from sympy import *
import numpy  as np
def RungeKutta(g,x0,h,y0,xn):

  x,y=symbols('x,y')
  f=lambdify([x,y],g)
  xt=x0+h
  Y=[y0]
  while xt<=xn:
      k1=h*f(x0,y0)
      k2=h*f(x0+h/2, y0+k1/2)
      k3=h*f(x0+h/2, y0+k2/2)
      k4=h*f(x0+h, y0+k3)
      y1=y0+(1/6)*(k1+2*k2+2*k3+k4)
      Y.append(y1)
      #print('y(%3.3f'%xt,') is %3.3f'%y1)
      x0=xt
      y0=y1
      xt=xt+h
  return np.round(Y,2)
RungeKutta('1+(y/x)',1,0.2,2,2)
```

```
array([2.  , 2.62, 3.27, 3.95, 4.66, 5.39])
```

10.3 Milne's predictor and corrector method

Apply Milne's predictor and corrector method to solve $dy/dx = x^2 + (y/2)$ at y(1.4). Given that y(1)=2, y(1.1)=2.2156, y(1.2)=2.4649, y(1.3)=2.7514. Use corrector formula thrice.

```
# Milne's method to solve first order DE
# Use corrector formula thrice
x0=1
y0=2
```

```
y1=2.2156
y2=2.4649
y3=2.7514
h=0.1
x1=x0+h
x2=x1+h
x3=x2+h
x4=x3+h
def f(x,y):
  return x**2+(y/2)

y10=f(x0, y0)
y11=f(x1,y1)
y12=f(x2,y2)
y13=f(x3,y3)
y4p=y0+(4*h/3)*(2*y11-y12+2*y13)
print('predicted value of y4 is %3.3f'%y4p)
y14=f(x4,y4p);
for i in range(1,4):
  y4=y2+(h/3)*(y14+4*y13+y12);
  print('corrected value of y4 after \t iteration  %d  is \t %3.5f\t '%
                                    (i,y4))
  y14=f(x4,y4);
```

```
predicted value of y4 is 3.079
corrected value of y4 after          iteration  1  is            3.07940
corrected value of y4 after          iteration  2  is            3.07940
corrected value of y4 after          iteration  3  is            3.07940
```

In the next program, function will take all the inputs from the user and display the answer.
Apply Milne's predictor and corrector method to solve $dy/dx = x^2 + (y/2)$ at y(1.4). Given that y(1)=2, y(1.1)=2.2156, y(1.2)=2.4649, y(1.3)=2.7514. Use corrector formula thrice.

```
from sympy import *
def Milne(g,x0,h,y0,y1,y2,y3):
    x,y=symbols('x,y')
    f=lambdify([x,y],g)
    x1=x0+h
    x2=x1+h
    x3=x2+h
    x4=x3+h

    y10=f(x0, y0)
    y11=f(x1,y1)
    y12=f(x2,y2)
    y13=f(x3,y3)
    y4p=y0+(4*h/3)*(2*y11-y12+2*y13)
    print('predicted value of y4',y4p)
    y14=f(x4,y4p)
    for i in range(1,4):
        y4=y2+(h/3)*(y14+4*y13+y12)
        print('corrected value of y4 , iteration  %d '%i,y4)
```

```
        y14=f(x4,y4)
Milne('x**2+y/2',1,0.1,2,2.2156,2.4649,2.7514)
```

```
predicted value of y4 3.0792733333333335
corrected value of y4 , iteration  1  3.0793962222222224
corrected value of y4 , iteration  2  3.079398270370371
corrected value of y4 , iteration  3  3.079398304506173
```

Apply Milne's predictor and corrector method to solve $dy/dx = x - y^2$, y(0)=2 obtain y(0.8). Take h=0.2. Use Runge-Kutta method to calculate required initial values.

```
Y=RungeKutta('x-y**2',0,0.2,0,0.8)
print('y values from Runge -Kutta method:',Y)
Milne('x-y**2',0,0.2,Y[0],Y[1],Y[2],Y[3])
```

```
y values from Runge -Kutta method: [0.   0.02 0.08 0.18 0.3 ]
predicted value of y4 0.3042133333333334
corrected value of y4 , iteration  1  0.3047636165214815
corrected value of y4 , iteration  2  0.3047412758696499
corrected value of y4 , iteration  3  0.3047421836520892
```

10.4 Exercise:

1. Find $y(0.1)$ by Runge Kutta method when $y' = x - y^2, y(0) = 1$.

   Ans: $y(0.1) = 0.91379$

2. Evaluate by Runge Kutta method : $y' = log(x + y), y(0) = 2$ at $x = 0(0.2)0.8$.

   Ans: $2.155, 2.3418, 2.557, 2.801$

3. Solve by Milnes method: $y' = x + y$, y(0)=1, h=0.1, Calculate $y(0.4)$. Calculate required initial values from Runge Kutta method.

   Ans: 1.583649219

Contents: Mechanical & Civil Engineering Stream

LAB 1: Programme to compute area, volume and center of gravity

1.1 Objectives:

Use python

1. to evaluate double integration.
2. to compute area and volume.
3. to calculate center of gravity of 2D object.

Syntax for the commands used:

1. Data pretty printer in Python:

```
pprint()
```

2. integrate:

```
integrate(function,(variable, min_limit, max_limit))
```

1.2 Double and triple integration

Example 1:

Evaluate the integral $\int_0^1 \int_0^x (x^2+y^2)dydx$

```
from sympy import *
x,y,z=symbols('x y z')
w1=integrate(x**2+y**2,(y,0,x),(x,0,1))
print(w1)
```

```
1/3
```

Example 2:

Evaluate the integral $\int_0^3 \int_0^{3-x} \int_0^{3-x-y} (xyz)dzdydx$

```
from sympy import *
x=Symbol('x')
y=Symbol('y')
z=Symbol('z')
w2=integrate((x*y*z),(z,0,3-x-y),(y,0,3-x),(x,0,3))
print(w2)
```

```
81/80
```

Example 3:

Prove that $\int\int(x^2+y^2)dydx = \int\int(x^2+y^2)dxdy$

```
from sympy import *
x=Symbol('x')
y=Symbol('y')
z=Symbol('z')
w3=integrate(x**2+y**2,y,x)
pprint(w3)
w4=integrate(x**2+y**2,x,y)
pprint(w4)
```

1.3 Area and Volume

Area of the region R in the cartesian form is $\int\limits_R\int dxdy$

Example 4:

Find the area of an ellipse by double integration. A=4 $\int\limits_0^a \int\limits_0^{(b/a)\sqrt{a^2-x^2}} dydx$

```
from sympy import *
x=Symbol('x')
y=Symbol('y')
#a=Symbol('a')
#b=Symbol('b')
a=4
b=6
w3=4*integrate(1,(y,0,(b/a)*sqrt(a**2-x**2)),(x,0,a))
print(w3)
```

```
24.0*pi
```

1.4 Area of the region R in the polar form is $\int\limits_R\int rdrd\theta$

Example 5:

Find the area of the cardioid $r = a(1+cos\theta)$ by double integration

```
from sympy import *
r=Symbol('r')
t=Symbol('t')
a=Symbol('a')
#a=4

w3=2*integrate(r,(r,0,a*(1+cos(t))),(t,0,pi))
pprint(w3)
```

1.5 Volume of a solid is given by $\int\limits_V \int \int dxdydz$

Example 6:

Find the volume of the tetrahedron bounded by the planes x=0,y=0 and z=0, $\frac{x}{a}+\frac{y}{b}+\frac{z}{c}=1$

```
from sympy import *
x=Symbol('x')
y=Symbol('y')
z=Symbol('z')
a=Symbol('a')
b=Symbol('b')
c=Symbol('c')
w2=integrate(1,(z,0,c*(1-x/a-y/b)),(y,0,b*(1-x/a)),(x,0,a))
print(w2)
```

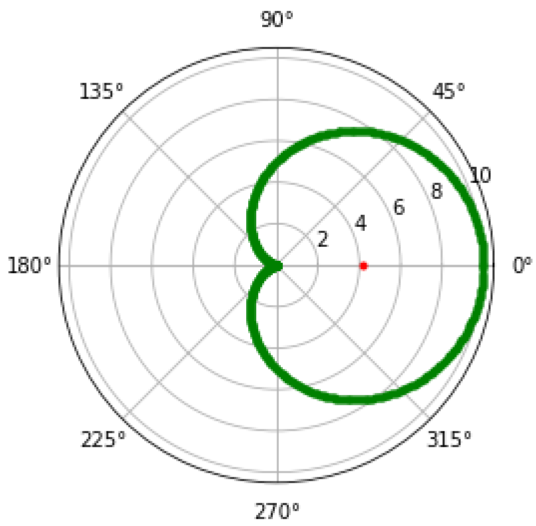
```
a*b*c/6
```

1.6 Center of Gravity

Find the center of gravity of cardioid . Plot the graph of cardioid and mark the center of gravity.

```
import numpy as np
import matplotlib.pyplot as plt
import math
from sympy import *
r=Symbol('r')
t=Symbol('t')
a=Symbol('a')
I1=integrate(cos(t)*r**2,(r,0,a*(1+cos(t))),(t,-pi,pi))
I2=integrate(r,(r,0,a*(1+cos(t))),(t,-pi,pi))
I=I1/I2
print(I)
I=I.subs(a,5)
plt.axes(projection = 'polar')
a=5


rad = np.arange(0, (2 * np.pi), 0.01)

# plotting the cardioid
for i in rad:
    r = a + (a*np.cos(i))
    plt.polar(i,r,'g.')

plt.polar(0,I,'r.')
plt.show()
```

```
5*a/6
```



1.7 Exercise:

1. Evaluate $\int_0^1 \int_0^x (x+y)dydx$
   Ans: 0.5

2. Find the $\int_0^{log(2)} \int_0^x \int_0^{x+log(y)} (e^{x+y+z})dzdydx$
   Ans: -0.2627

3. Find the area of positive quadrant of the circle $x^2 + y^2 = 16$
   Ans: 4π

4. Find the volume of the tetrahedron bounded by the planes x=0,y=0 and z=0, $\frac{x}{2} + \frac{y}{3} + \frac{z}{4} = 1$
   Ans: 4

LAB 2: Evaluation of improper integrals, Beta and Gamma functions

2.1 Objectives:

Use python

1. to evaluate improper integrals using Beta function.
2. to evaluate improper integrals using Gamma function.

Syntax for the commands used:

1. `gamma`

```
math.gamma(x)
```

   Parameters :

   `x` : The number whose gamma value needs to be computed.

2. `beta`

```
math.beta(x,y)
```

   Parameters :

   `x` ,y: The numbers whose beta value needs to be computed.

3. **Note:** We can evaluate improper integral involving infinity by using `inf`.

Example 1:

Evaluate $\int_0^\infty e^{-x}dx$.

```
from sympy import *
x=symbols('x')
w1=integrate(exp(-x),(x,0,float('inf')))
print(simplify(w1))
```

1

Gamma function is $x(n) = \int_0^\infty e^{-x}x^{n-1}dx$

Example 2:

Evaluate $\Gamma(5)$ by using definition

```
from sympy import *
x=symbols('x')
w1=integrate(exp(-x)*x**4,(x,0,float('inf')))
print(simplify(w1))
```

24

Example 3:

Evaluate $\int_0^\infty e^{-st}\cos(4t)dt$. That is Laplace transform of $\cos(4t)$

```
from sympy import *
t,s=symbols('t,s')
# for infinity in sympy we use oo
w1=integrate(exp(-s*t)*cos(4*t),(t,0,oo))
display(simplify(w1))
```

$$\begin{cases} \frac{s}{s^2+16} & \text{for } 2\left|\arg{(s)}\right| < \pi \\ \int_0^\infty e^{-st}\cos{(4t)}\,dt & \text{otherwise} \end{cases}$$

Example 4:

Find Beta(3,5), Gamma(5)

```
#beta and gamma functions
from sympy import beta, gamma
m=input('m :');
n=input('n :');
m=float(m);
n=float(n);
s=beta(m,n);
t=gamma(n)
print('gamma (',n,') is %3.3f'%t)
print('Beta (',m,n,') is  %3.3f'%s)
```

```
m :3
n :5
gamma ( 5.0 ) is 24.000
Beta ( 3.0 5.0 ) is  0.010
```

Example 5:

Calculate Beta(5/2,7/2) and Gamma(5/2).

```
#beta and gamma functions
# If the number is a fraction give it in decimals. Eg 5/2=2.5
from sympy import beta, gamma
m=float(input('m : '));
n=float(input('n :'));

s=beta(m,n);
t=gamma(n)
print('gamma (',n,') is %3.3f'%t)
print('Beta (',m,n,') is %3.3f '%s)
```

```
m : 2.5
n :3.5
gamma ( 3.5 ) is 3.323
Beta ( 2.5 3.5 ) is 0.037
```

Example 6:

Verify that $Beta(m, n) = Gamma(m)Gamma(n)/Gamma(m + n)$ for m=5 and n=7

```
from sympy import beta, gamma
m=5;
n=7;
m=float(m);
n=float(n);
s=beta(m,n);
t=(gamma(m)*gamma(n))/gamma(m+n);
print(s,t)
if (abs(s-t)<=0.00001):
    print('beta and gamma are related')
else:
    print('given values are wrong')
```

```
0.0004329004329004329004329004333 0.000432900432900433
beta and gamma are related
```

2.2 Exercise:

1. Evaluate $\int_{0}^{\infty} e^{-t}cos(2t)dt$
   Ans: 1/5

2. Find the value of Beta(5/2,9/2)
   Ans: 0.0214

3. Find the value of Gamma(13)
   Ans: 479001600

4. Verify that $Beta(m, n) = Gamma(m)Gamma(n)/Gamma(m + n)$ for m=7/2 and n=11/2
   Ans: True

LAB 3: Finding gradient, divergent, curl and their geometrical interpretation

1.1 Objectives:

Use python

1. to find the gradient of a given scalar function.
2. to find find divergence and curl of a vector function.

1.2 Method I:

To find gradient of $\phi = x^2y + 2xz - 4$.

```
#To find gradient of scalar point function.
from sympy.vector import *
from sympy import symbols
N=CoordSys3D('N') #Setting the coordinate system
x,y,z=symbols('x y z')
A=N.x**2*N.y+2*N.x*N.z-4 #Variables x,y,z to be used with coordinate
                                        system N
delop=Del() #Del operator
display(delop(A)) #Del operator applied to A
gradA=gradient(A) #Gradient function is used
print(f"\n Gradient of {A} is \n")
display(gradA)
```

$$\left(\frac{\partial}{\partial \mathbf{x_N}}\left(\mathbf{x_N}^2\mathbf{y_N} + 2\mathbf{x_N}\mathbf{z_N} - 4\right)\right)\hat{\mathbf{i}}_\mathbf{N} + \left(\frac{\partial}{\partial \mathbf{y_N}}\left(\mathbf{x_N}^2\mathbf{y_N} + 2\mathbf{x_N}\mathbf{z_N} - 4\right)\right)\hat{\mathbf{j}}_\mathbf{N} + \left(\frac{\partial}{\partial \mathbf{z_N}}\left(\mathbf{x_N}^2\mathbf{y_N} + 2\mathbf{x_N}\mathbf{z_N} - 4\right)\right)\hat{\mathbf{k}}_\mathbf{N}$$

```
Gradient of N.x**2*N.y + 2*N.x*N.z - 4 is
```

$$\left(2\mathbf{x_N}\mathbf{y_N} + 2\mathbf{z_N}\right)\hat{\mathbf{i}}_\mathbf{N} + \left(\mathbf{x_N}^2\right)\hat{\mathbf{j}}_\mathbf{N} + \left(2\mathbf{x_N}\right)\hat{\mathbf{k}}_\mathbf{N}$$

To find divergence of $\vec{F} = x^2yz\hat{i} + y^2zx\hat{j} + z^2xy\hat{k}$

```
#To find divergence of a vector point function
from sympy.vector import *
from sympy import symbols
N=CoordSys3D('N')
x,y,z=symbols('x y z')
A=N.x**2*N.y*N.z*N.i+N.y**2*N.z*N.x*N.j+N.z**2*N.x*N.y*N.k
delop=Del()
divA=delop.dot(A)
display(divA)

print(f"\n Divergence of {A} is \n")
display(divergence(A))
```

$$\frac{\partial}{\partial z_N} x_N y_N z_N^2 + \frac{\partial}{\partial y_N} x_N y_N^2 z_N + \frac{\partial}{\partial x_N} x_N^2 y_N z_N$$

```
Divergence of N.x**2*N.y*N.z*N.i + N.x*N.y**2*N.z*N.j + N.x*N.y*N.z**2*N.k is
```

$$6 x_N y_N z_N$$

To find curl of $\vec{F} = x^2yz\hat{i} + y^2zx\hat{j} + z^2xy\hat{k}$

```
#To find curl of a vector point function
from sympy.vector import *
from sympy import symbols
N=CoordSys3D('N')
x,y,z=symbols('x y z')
A=N.x**2*N.y*N.z*N.i+N.y**2*N.z*N.x*N.j+N.z**2*N.x*N.y*N.k
delop=Del()
curlA=delop.cross(A)
display(curlA)

print(f"\n Curl of {A} is \n")
display(curl(A))
```

$$\left(\frac{\partial}{\partial y_N} x_N y_N z_N^2 - \frac{\partial}{\partial z_N} x_N y_N^2 z_N\right)\hat{i}_N + \left(-\frac{\partial}{\partial x_N} x_N y_N z_N^2 + \frac{\partial}{\partial z_N} x_N^2 y_N z_N\right)\hat{j}_N + \left(\frac{\partial}{\partial x_N} x_N y_N^2 z_N - \frac{\partial}{\partial y_N} x_N^2 y_N z_N\right)\hat{k}_N$$

```
Curl of N.x**2*N.y*N.z*N.i + N.x*N.y**2*N.z*N.j + N.x*N.y*N.z**2*N.k is
```

$$\left(-x_N y_N^2 + x_N z_N^2\right)\hat{i}_N + \left(x_N^2 y_N - y_N z_N^2\right)\hat{j}_N + \left(-x_N^2 z_N + y_N^2 z_N\right)\hat{k}_N$$

1.3 Method II:

To find gradient of $\phi = x^2yz$.

```
#To find gradient of a scalar point function x^2yz
from sympy.physics.vector import *
from sympy import var,pprint
var('x,y,z')
v=ReferenceFrame('v')
F=v[0]**2*v[1]*v[2]
G=gradient(F,v)
F=F.subs([(v[0],x),(v[1],y),(v[2],z)])
print("Given scalar function F=")
display(F)
G=G.subs([(v[0],x),(v[1],y),(v[2],z)])
print("\n Gradient of F=")
display(G)
```

Given scalar function F=

x^2yz

Gradient of F=

$2xyz\hat{\mathbf{v}}_{\mathbf{x}} + x^2z\hat{\mathbf{v}}_{\mathbf{y}} + x^2y\hat{\mathbf{v}}_{\mathbf{z}}$

To find divergence of $\vec{F} = x^2y\hat{\imath} + yz^2\hat{\jmath} + x^2z\hat{k}$.

```
#To find divergence of F=x^2yi+yz^2j+x^2zk
from sympy.physics.vector import *
from sympy import var
var('x,y,z')
v=ReferenceFrame('v')
F=v[0]**2*v[1]*v.x+v[1]*v[2]**2*v.y+v[0]**2*v[2]*v.z
G=divergence(F,v)
F=F.subs([(v[0],x),(v[1],y),(v[2],z)])
print("Given vector point function is ")
display(F)

G=G.subs([(v[0],x),(v[1],y),(v[2],z)])
print("Divergence of F=")
display(G)
```

Given vector point function is

$x^2y\hat{\mathbf{v}}_{\mathbf{x}} + yz^2\hat{\mathbf{v}}_{\mathbf{y}} + x^2z\hat{\mathbf{v}}_{\mathbf{z}}$

Divergence of F=

$x^2 + 2xy + z^2$

To find curl of $\vec{F} = xy^2\hat{\imath} + 2x^2yz\hat{\jmath} - 3yz^2\hat{k}$

```
#To find curl of F=xy^2i+2x^2yzj-3yz^2k
from sympy.physics.vector import *
from sympy import var
var('x,y,z')
v=ReferenceFrame('v')
F=v[0]*v[1]**2*v.x+2*v[0]**2*v[1]*v[2]*v.y-3*v[1]*v[2]**2*v.z
G=curl(F,v)
F=F.subs([(v[0],x),(v[1],y),(v[2],z)])
print("Given vector point function is ")
display(F)

G=G.subs([(v[0],x),(v[1],y),(v[2],z)])
print("curl of F=")
display(G)
```

Given vector point function is

$$xy^2\hat{\mathbf{v}}_{\mathbf{x}} + 2x^2yz\hat{\mathbf{v}}_{\mathbf{y}} - 3yz^2\hat{\mathbf{v}}_{\mathbf{z}}$$

curl of F=

$$(-2x^2y - 3z^2)\hat{\mathbf{v}}_{\mathbf{x}} + (4xyz - 2xy)\hat{\mathbf{v}}_{\mathbf{z}}$$

1.4 Exercise:

1. If $u = x + y + z$, $v = x^2 + y^2 + z^2$, $w = yz + zx + xy$, find grad$u$, grad$v$ and grad$w$.
   Ans: $\hat{i} + \hat{j} + \hat{k}$, $2(x\hat{i} + y\hat{j} + z\hat{k})$, $(y+z)\hat{i} + (z+x)\hat{j} + (z+x)\hat{k}$.

2. Evaluate div$F$ and curl$F$ at the point (1,2,3), given that $\vec{F} = x^2yz\hat{i} + xy^2z\hat{j} + xyz^2\hat{k}$.
   Ans: $6xyz$, $x(z^2 - y^2)\hat{i} + y(x^2 - z^2)\hat{j} + z(y^2 - x^2)\hat{k}$.

3. Prove that the vector $(yz - x^2)\hat{i} + (4y - z^2x)\hat{j} + (2xz - 4z)\hat{k}$ is solenoidal.

4. Find the vector normal to the surface $xy^3z^2 = 4$ at the point $(-1, -1, 2)$.
   Ans: $-4\hat{i} - 12\hat{j} + 4\hat{k}$.

5. If $\vec{R} = x\hat{i} + y\hat{j} + z\hat{k}$, show that (i) $\nabla \cdot \vec{R} = 3$, (ii) $\nabla \times \vec{R} = 0$.

LAB 4: Verification of Green's theorem

1.1 Objectives:

Use python

1. to evaluate integrals using Green's theorem.

1.2 Green's theorem

Statement of Green's theorem in the plane: If $P(x, y)$ and $Q(x, y)$ be two continuous functions having continuous partial derivatives in a region R of the xy-plane, bounded by a simple closed curve C, then

$$\oint (Pdx + Qdy) = \int\int_R \left(\frac{\partial Q}{\partial x} - \frac{\partial P}{\partial y}\right) dxdy.$$

1. Using Green's theorem, evaluate $\oint_c [(x + 2y)dx + (x - 2y)dy]$, where c is the region bounded by coordinate axes and the line $x = 1$ and $y = 1$.

```
from sympy import *
var('x,y')
p=x+2*y
q=x-2*y
f=diff(q,x)-diff(p,y)
soln=integrate(f,[x,0,1],[y,0,1])
print("I=",soln)
```

```
I= -1
```

2. Using Green's theorem, evaluate $\oint_c [(xy + y^2)dx + x^2 dy]$, where c is the closed curve bounded by $y = x$ and $y = x^2$.

```
from sympy import *
var('x,y')
p=x*y+y**2
q=x**2
f=diff(q,x)-diff(p,y)
soln=integrate(f,[y,x**2,x],[x,0,1])
print("I=",soln)
```

```
I= -1/20
```

1.3 Exercise:

1. Using Green's theorem, evaluate $\oint_c [(3x + 4y)dx + (2x - 3y)dy]$, where c is the boundary of the circle $x^2 + y^2 = 4$.

   Ans: -8π

LAB 5: Solution of Lagrange's linear partial differential equations

1.1 Objectives:

Use python

1. to solve linear Partial Differential Equations of first order

Solve the PDE, $xp + yq = z$, where $z = f(x, y)$

```python
from sympy.solvers.pde import pdsolve
from sympy import Function, Eq,cot,classify_pde,pprint
from sympy.abc import x, y, a
f = Function('f')
z = f(x, y)
zx = z.diff(x)
zy = z.diff(y)
#Solve xp+yq=z
eq = Eq(x*zx+y*zy, z)
pprint(eq)
print("\n")
soln=pdsolve(eq,z)
pprint(soln)
```

```
  ∂                   ∂
x⋅──(f(x, y)) + y⋅──(f(x, y)) = f(x, y)
  ∂x                  ∂y

              ⎛y⎞
f(x, y) = x⋅F⎜─⎟
              ⎝x⎠
```

Solve the PDE $2p + 3q = 1$, where $p = \frac{\partial z}{\partial x}$ and $p = \frac{\partial z}{\partial y}$

```python
from sympy.solvers.pde import pdsolve
from sympy import Function, Eq,cot,classify_pde,pprint
from sympy.abc import x, y, a
f = Function('f')
z = f(x, y)
zx = z.diff(x)
zy = z.diff(y)
#Solve 2p+3q=1
eq = Eq(2*zx+3*zy, 1)
pprint(eq)
print("\n")
soln=pdsolve(eq,z)
pprint(soln)
```

```
  ∂                   ∂
2·──(f(x, y)) + 3·──(f(x, y)) = 1
  ∂x                  ∂y
```

```
          2·x   3·y
f(x, y) = ─── + ─── + F(3·x - 2·y)
           13    13
```

Solve the PDE $x^2p + y^2q = (x + y)z$, where $p = \frac{\partial z}{\partial x}$ and $p = \frac{\partial z}{\partial y}$

```
from sympy.solvers.pde import pdsolve
from sympy import Function, Eq,cot,classify_pde,pprint
from sympy.abc import x, y, a
f = Function('f')
z = f(x, y)
zx = z.diff(x)
zy = z.diff(y)
#Solve x^2p+y^2q=(x+y)z
eq=Eq(x**2*zx+y**2*zy,(x+y)*z)
pprint(eq)

print("\n")
soln=pdsolve(eq,z)
pprint(soln)
```

```
 2 ∂                 2 ∂
x ·──(f(x, y)) + y ·──(f(x, y)) = (x + y)·f(x, y)
   ∂x                  ∂y

                         ⎛-x + y⎞
f(x, y) = (x - y)·F⎜──────⎟
                         ⎝ x·y  ⎠
```

1.2 Exercise:

1. Solve $y^2p + x^2q = y^2x$

   Ans: $z = x^2/2 + F(y^3 - x^3)$

2. Solve $xp + yq = 3z$

   Ans: $z = x^3F(y/x)$

3. Solve $y^2p - xyq = x(z - 2y)$

   Ans: $x^2 + y^2 = F(y^2 - yz)$

LAB 6: Solution of algebraic and transcendental equation by Regula-Falsi and Newton-Raphson method

6.1 Objectives:

Use python

1. to solve algebraic and transcendental equation by Regula-Falsi method.
2. to solve algebraic and transcendental equation by Newton-Raphson method.

6.2 Regula-Falsi method to solve a transcendental equation

Obtain a root of the equation $x^3 - 2x - 5 = 0$ between 2 and 3 by regula-falsi method. Perform 5 iterations.

```
# Regula Falsi method
from sympy import *
x=Symbol('x')
g =input('Enter the function  ')  #%x^3-2*x-5;    %function
f=lambdify(x,g)
a=float(input('Enter a valus :')) #2
b=float(input('Enter b valus :')) # 3
N=int(input('Enter number of iterations :')) #5

for i in range(1,N+1):
    c=(a*f(b)-b*f(a))/(f(b)-f(a))
    if((f(a)*f(c)<0)):
        b=c
    else:
        a=c
    print('itration %d  \t  the root %0.3f \t function value %0.3f \n'%
                                    (i,c,f(c)));
```

```
Enter the function  x**3-2*x-5
Enter a valus :2
Enter b valus :3
Enter number of iterations :5
itration 1       the root 2.059       function value -0.391

itration 2       the root 2.081       function value -0.147

itration 3       the root 2.090       function value -0.055

itration 4       the root 2.093       function value -0.020

itration 5       the root 2.094       function value -0.007
```

Using tolerance value we can write the same program as follows:
Obtain a root of the equation $x^3 - 2x - 5 = 0$ between 2 and 3 by regula-falsi method. Correct to 3 decimal places.

```
# Regula Falsi method while loop2
from sympy import *
x=Symbol('x')
g =input('Enter the function  ')  #%x^3-2*x-5;    %function
f=lambdify(x,g)
a=float(input('Enter a valus :')) # 2
b=float(input('Enter b valus :')) # 3
N=float(input('Enter tolarence  :')) # 0.001
x=a;
c=b;
i=0
while (abs(x-c)>=N):
    x=c
    c=((a*f(b)-b*f(a))/(f(b)-f(a)));
    if((f(a)*f(c)<0)):
        b=c
    else:
        a=c
        i=i+1
    print('itration %d  \t  the root %0.3f \t function value %0.3f \n'%
                                   (i,c,f(c)));
print('final value of the root is %0.5f'%c)
```

```
Enter the function  x**3-2*x-5
Enter a valus :2
Enter b valus :3
Enter tolarence  :0.001
itration 1         the root 2.059        function value -0.391

itration 2         the root 2.081        function value -0.147

itration 3         the root 2.090        function value -0.055

itration 4         the root 2.093        function value -0.020

itration 5         the root 2.094        function value -0.007

itration 6         the root 2.094        function value -0.003

final value of the root is 2.09431
```

6.3 Newton-Raphson method to solve a transcendental equation

Find a root of the equation $3x = \cos x + 1$, near 1, by Newton Raphson method. Perform 5 iterations

```
from sympy import *
x=Symbol('x')
g =input('Enter the function  ')  #%3x-cos(x)-1;    %function
f=lambdify(x,g)
dg = diff(g);
```

```
df=lambdify(x,dg)
x0= float(input('Enter the intial approximation  ')); # x0=1
n= int(input('Enter the number of iterations  '));   #n=5;
for i in range(1,n+1):
    x1 =(x0 - (f(x0)/df(x0)))
    print('itration %d  \t  the root %0.3f \t function value %0.3f \n'%
                                        (i, x1,f(x1))); #print all
                                        iteration value
    x0 = x1
```

```
Enter the function  3*x-cos(x)-1
Enter the intial approximation  1
Enter the number of iterations  5
itration 1       the root 0.620        function value 0.046

itration 2       the root 0.607        function value 0.000

itration 3       the root 0.607        function value 0.000

itration 4       the root 0.607        function value 0.000

itration 5       the root 0.607        function value 0.000
```

6.4 Exercise:

1. Find a root of the equation $3x = \cos x + 1$, between 0 and 1, by Regula-falsi method. Perform 5 iterations.

   Ans: 0.607

2. Find a root of the equation $xe^x = 2$, between 0 and 1, by Regula-falsi method. Correct to 3 decimal places.

   Ans: 0.853

3. Obtain a real positive root of $x^4 - x = 0$, near 1, by Newton-Raphson method. Perform 4 iterations.

   Ans: 1.856

4. Obtain a real positive root of $x^4 + x^3 - 7x^2 - x + 5 = 0$, near 3, by Newton-Raphson method. Perform 7 iterations.

   Ans: 2.061

LAB 7: Interpolation /Extrapolation using Newton's forward and backward difference formula

7.1 Objectives:

Use python

1. to interpolate using Newton's Forward interpolation method.
2. to interpolate using Newton's backward interpolation method.
3. to extrapolate using Newton's backward interpolation method.

1. Use Newtons forward interpolation to obtain the interpolating polynomial and hence calculate y(2) for the following:

x:	1	3	5	7	9
y:	6	10	62	210	502

```
from sympy import *
import numpy as np
n = int(input('Enter number of data points: '))
210
x = np.zeros((n))
y = np.zeros((n,n))


# Reading data points
print('Enter data for x and y: ')
for i in range(n):
    x[i] = float(input( 'x['+str(i)+']='))
    y[i][0] = float(input( 'y['+str(i)+']='))

# Generating forward difference table
for i in range(1,n):
    for j in range(0,n-i):
        y[j][i] = y[j+1][i-1] - y[j][i-1]

print('\nFORWARD DIFFERENCE TABLE\n');

for i in range(0,n):
    print('%0.2f' %(x[i]), end='')
    for j in range(0, n-i):
        print('\t\t%0.2f' %(y[i][j]), end='')
    print()
 # obtaining the polynomial
t=symbols('t')
f=[] # f is a list type data


p=(t-x[0])/(x[1]-x[0])
f.append(p)
for i in range(1,n-1):
    f.append(f[i-1]*(p-i)/(i+1))
    poly=y[0][0]
for i in range(n-1):
    poly=poly+y[0][i+1]*f[i]
```

```
simp_poly=simplify(poly)
print('\nTHE INTERPOLATING POLYNOMIAL IS\n');
pprint(simp_poly)
# if you want to interpolate at some point the next session will help
inter=input('Do you want to interpolate at a point(y/n)? ') # y
if inter=='y':
    a=float(input('enter the point ')) #2
    interpol=lambdify(t,simp_poly)
    result=interpol(a)
    print('\nThe  value of the function at' ,a,'is\n',result);
```

```
Enter number of data points: 5
Enter data for x and y:
x[0]=1
y[0]=6
x[1]=3
y[1]=10
x[2]=5
y[2]=62
x[3]=7
y[3]=210
x[4]=9
y[4]=502

FORWARD DIFFERENCE TABLE

1.00          6.00          4.00          48.00         48.00         0.00
3.00          10.00         52.00         96.00         48.00
5.00          62.00         148.00        144.00
7.00          210.00        292.00
9.00          502.00

THE INTERPOLATING POLYNOMIAL IS

     3        2
1.0⋅t  - 3.0⋅t  + 1.0⋅t + 7.0
Do you want to interpolate at a point(y/n)? y
enter the point 2

The  value of the function at 2.0 is
 5.0
```

2. Use Newtons backward interpolation to obtain the interpolating polynomial and hence calculate y(8) for the following data:

x:	1	3	5	7	9
y:	6	10	62	210	502

```
from sympy import *
import numpy as np
import sys
print("This will use Newton's backword intepolation formula ")
# Reading number of unknowns
n = int(input('Enter number of data points: '))

# Making numpy array of n & n x n size and initializing
# to zero for storing x and y value along with differences of y
x = np.zeros((n))
y = np.zeros((n,n))


# Reading data points
```

```
print('Enter data for x and y: ')
for i in range(n):
    x[i] = float(input( 'x['+str(i)+']='))
    y[i][0] = float(input( 'y['+str(i)+']='))

# Generating backward difference table
for i in range(1,n):
    for j in range(n-1,i-2,-1):
        y[j][i] = y[j][i-1] - y[j-1][i-1]


print('\nBACKWARD DIFFERENCE TABLE\n');

for i in range(0,n):
    print('%0.2f' %(x[i]), end='')
    for j in range(0, i+1):
        print('\t%0.2f' %(y[i][j]), end='')
    print()


# obtaining the polynomial
t=symbols('t')
f=[]


p=(t-x[n-1])/(x[1]-x[0])
f.append(p)
for i in range(1,n-1):
       f.append(f[i-1]*(p+i)/(i+1))

poly=y[n-1][0]
print(poly)
for i in range(n-1):
       poly=poly+y[n-1][i+1]*f[i]
       simp_poly=simplify(poly)
print('\nTHE INTERPOLATING POLYNOMIAL IS\n');
pprint(simp_poly)
# if you want to interpolate at some point the next session will help
inter=input('Do you want to interpolate at a point(y/n)? ')
if inter=='y':
       a=float(input('enter the point '))
       interpol=lambdify(t,simp_poly)
       result=interpol(a)
       print('\nThe  value of the function at' ,a,'is\n',result);
```

```
This will use Newton's backword intepolation formula
Enter number of data points: 5
Enter data for x and y:
x[0]=1
y[0]=6
x[1]=3
y[1]=10
x[2]=5
y[2]=62
x[3]=7
y[3]=210
x[4]=9
y[4]=502

BACKWARD DIFFERENCE TABLE

1.00    6.00
3.00    10.00   4.00
5.00    62.00   52.00   48.00
7.00    210.00  148.00  96.00   48.00
9.00    502.00  292.00  144.00  48.00   0.00
502.0

THE INTERPOLATING POLYNOMIAL IS

     3        2
1.0·t  - 3.0·t  + 1.0·t + 7.0
Do you want to interpolate at a point(y/n)? y
enter the point 8

The  value of the function at 8.0 is
 335.0
```

7.2 Exercise:

1. Obtain the interpolating polynomial for the following data

x:	0	1	2	3
y:	1	2	1	10

   Ans: $2x^3 - 7x^2 + 6x + 1$

2. Find the number of men getting wage Rs. 100 from the following table:

wage:	50	150	250	350
No. of men:	9	30	35	42

   Ans: 23 men

3. Using Newton's backward interpolation method obtain y(160) for the following data

x :	100	150	200	250	300
y :	10	13	15	17	18

   Ans: 13.42

4. Using Newtons forward interpolation polynomial and calculate y(1) and y(10).

x :	3	4	5	6	7	8	9
y :	4.8	8.4	14.5	23.6	36.2	52.8	73.9

   Ans: 3.1 and 100

LAB 8: Computation of area under the curve using Trapezoidal, Simpson's $\left(\frac{1}{3}\right)^{\text{rd}}$ and Simpsons $\left(\frac{3}{8}\right)^{\text{th}}$ rule

8.1 Objectives:

Use python

1. to find area under the curve represented by a given function using Trapezoidal rule.
2. to find area under the curve represented by a given function using Simpson's $\left(\frac{1}{3}\right)^{\text{rd}}$ rule.
3. to find area under the curve represented by a given function using Simpson's $\left(\frac{3}{8}\right)^{\text{th}}$ rule.
4. to find the area below the curve when discrete points on the curve are given.

8.2 Trapezoidal Rule

Evaluate $\int_0^5 \frac{1}{1+x^2}$.

```
# Definition of the function to integrate
def my_func(x):
    return 1 / (1 + x ** 2)
```

```
# Function to implement trapezoidal method
def trapezoidal(x0, xn, n):
  h = (xn - x0) / n                              # Calculating step
                                                 size
  # Finding sum
  integration = my_func(x0) + my_func(xn)        # Adding first and
                                                 last terms
  for i in range(1, n):
    k = x0 + i * h                               # i-th step value
    integration = integration + 2 * my_func(k)   # Adding areas of the
                                                    trapezoids
  # Proportioning sum of trapezoid areas
  integration = integration * h / 2
  return integration
```

```
# Input section
lower_limit = float(input("Enter lower limit of integration: "))
upper_limit = float(input("Enter upper limit of integration: "))
sub_interval = int(input("Enter number of sub intervals: "))

# Call trapezoidal() method and get result
result = trapezoidal(lower_limit, upper_limit, sub_interval)

# Print result
print("Integration result by Trapezoidal method is: " , result)
```

```
Enter lower limit of integration: 0
Enter upper limit of integration: 5
Enter number of sub intervals: 10
Integration result by Trapezoidal method is:  1.3731040812301099
```

8.3 Simpson's $\left(\frac{1}{3}\right)^{\text{rd}}$ Rule

Evaluate $\int\limits_0^5 \frac{1}{1+x^2}$.

```
# Definition of the function to integrate
def my_func(x):
    return 1 / (1 + x ** 2)
```

```
# Function to implement the Simpson's one-third rule

def simpson13(x0,xn,n):
  h = (xn - x0) / n                    # calculating step size
  # Finding sum
  integration = (my_func(x0) + my_func(xn))
  k = x0
  for i in range(1,n):
    if i%2 == 0:
      integration = integration + 4 * my_func(k)
    else:
      integration = integration + 2 * my_func(k)
    k += h
  # Finding final integration value
  integration = integration * h * (1/3)
  return integration

# Input section
lower_limit = float(input("Enter lower limit of integration: "))
upper_limit = float(input("Enter upper limit of integration: "))
sub_interval = int(input("Enter number of sub intervals: "))

# Call trapezoidal() method and get result
result = simpson13(lower_limit, upper_limit, sub_interval)
print("Integration result by Simpson's 1/3 method is: %0.6f" % (result)
                                      )
```

```
Enter lower limit of integration: 0
Enter upper limit of integration: 5
Enter number of sub intervals: 100
Integration result by Simpson's 1/3 method is: 1.404120
```

8.4 Simpson's 3/8th rule

Evaluate $\int_0^6 \frac{1}{1+x^2} dx$ using Simpson's 3/8 th rule, taking 6 sub intervals

```
def simpsons_3_8_rule(f, a, b, n):
```

```
  h = (b - a) / n
  s = f(a) + f(b)
  for i in range(1, n, 3):
    s += 3 * f(a + i * h)
  for i in range(3, n-1, 3):
    s += 3 * f(a + i * h)
  for i in range(2, n-2, 3):
    s += 2 * f(a + i * h)
  return s * 3 * h / 8

def f(x):
  return 1/(1+x**2) # function here

a = 0   # lower limit
b = 6 #  upper limit
n = 6 # number of sub intervals

result = simpsons_3_8_rule(f, a, b, n)
print('%3.5f'%result)
```

```
1.27631
```

8.5 Exercise:

1. Evaluate the integral $\int_0^1 \frac{x^2}{1+x^3}dx$ using Simpson's $\frac{1}{3}$ rule.

   Ans: 0.23108

2. Use Simpson's $\frac{3}{8}$ rule to find $\int_0^{0.6} e^{-x^2}dx$ by taking seven ordinates.

   Ans: 0.5351

3. Evaluate using trapezoidal rule $\int_0^{\pi} sin^2 x dx$. Take $n = 6$.

   Ans: $\pi/2$

4. A solid of revolution is formed by rotating about the x-axis, the area between the x-axis, the lines $x = 0$ and $x = 1$, and a curve through the points with the following co-ordinates:

x	y
0.00	1.0000
0.25	0.9896
0.50	0.9589
0.75	0.9089
1.00	0.8415

Estimate the volume of the solid formed using Simpson's $\frac{1}{3}$rd rule. Hint: Required volume is $\int_0^1 y^2 * \pi dx$. **[Ans: 2.8192]**

5. The velocity v(km/min) of a moped which starts from rest, is given at fixed intervals of time t(min) as follows:

t:	2	4	6	8	10	12	14	16	18	20
v:	10	18	25	29	32	20	11	5	2	0

Estimate approximately the distance covered in twenty minutes.

Answer for 5.

We know that ds/dt=v. So to get distance (s) we have to integrate.

Here $h = 2.2$, $v_0 = 0$, $v_1 = 10$, $v_2 = 18$, $v_3 = 25$ etc.

```
# we shall use simpson's 1/3 rule directly to estimate

h=2
y= [0, 10 ,18, 25, 29,32 ,20, 11 ,5 ,2 , 0]
result=(h/3)*((y[0]+y[10])+4*(y[1]+y[3]+y[5]+y[7]+y[9])+2*(y[2]+y[4]+y[
                                        6]+y[8]))

print('%3.5f'%result,'km.')
```

```
309.33333 km.
```

LAB 9: Solution of ODE of first order and first degree by Taylor's series and Modified Euler's method

9.1 Objectives:

Use python

1. to solve ODE by Taylor series method.
2. to solve ODE by Modified Euler method.
3. to trace the solution curves.

9.2 Taylor series method to solve ODE

Solve: $\frac{dy}{dx} - 2y = 3e^x$ with $y(0) = 0$ using Taylor series method at $x = 0.1(0.1)0.3$.

```
## module taylor
'''X,Y = taylor(deriv,x,y,xStop,h).
4th-order Taylor series method for solving the initial value problem {y
                                }' = {F(x,{y})}, where
{y} = {y[0],y[1],...y[n-1]}.
x,y = initial conditions
xStop = terminal value of x
h = increment of x
'''
from numpy import array
def taylor(deriv,x,y,xStop,h):
    X = []
    Y = []
    X.append(x)
    Y.append(y)
    while x < xStop:                    # Loop over integration steps
        D = deriv(x,y)                 # Derivatives of y
        H = 1.0
        for j in range(3):            # Build Taylor series
            H = H*h/(j + 1)
            y = y + D[j]*H            # H = h^j/j!
        x = x + h
        X.append(x) # Append results to
        Y.append(y) # lists X and Y

    return array(X),array(Y) # Convert lists into arrays

# deriv = user-supplied function that returns derivatives in the 4 x n
                                array
'''
[y'[0] y'[1] y'[2] ... y'[n-1]
y''[0] y''[1] y''[2] ... y''[n-1]
y'''[0] y'''[1] y'''[2] ... y'''[n-1]
y''''[0] y''''[1] y''''[2] ... y''''[n-1]]
'''
def deriv(x,y):
    D = zeros((4,1))
```

```
    D[0] = [2*y[0] + 3*exp(x)]
    D[1] = [4*y[0]+ 9*exp(x)]
    D[2] = [8*y[0]+ 21*exp(x)]
    D[3] = [16*y[0]+ 45*exp(x)]
    return D

x = 0.0        # Initial value of x
xStop = 0.3      # last value
y = array([0.0])          # Initial values of y
h = 0.1              # Step size
X,Y = taylor(deriv,x,y,xStop,h)

print("The required values are :at x= %0.2f, y=%0.5f, x=%0.2f, y=%0.5f,
                                     x = %0.2f, y=%0.5f,x = %0.2f, y=%0
                                    .5f"%(X[0],Y[0],X[1],Y[1],X[2],Y[2]
                                    ,X[3],Y[3] ))
```

```
The required values are :at x= 0.00, y=0.00000, x=0.10, y=0.34850,
x = 0.20, y=0.81079,x = 0.30, y=1.41590
```

Solve $y' + 4y = x^2$ with initial conditions $y(0) = 1$ using Taylor series method at $x = 0.1, 0.2$.

```
from numpy import array
def taylor(deriv,x,y,xStop,h):
    X = []
    Y = []
    X.append(x)
    Y.append(y)
    while x < xStop:                  # Loop over integration steps
        D = deriv(x,y)               # Derivatives of y
        H = 1.0
        for j in range(3):          # Build Taylor series
            H = H*h/(j + 1)
            y = y + D[j]*H          # H = h^j/j!
        x = x + h
        X.append(x) # Append results to
        Y.append(y) # lists X and Y

    return array(X),array(Y) # Convert lists into arrays

# deriv = user-supplied function that returns derivatives in the 4 x n
                                    array
'''
[y'[0] y'[1] y'[2] ... y'[n-1]
y"[0] y"[1] y"[2] ... y"[n-1]
y'''[0] y'''[1] y'''[2] ... y'''[n-1]
y""[0] y""[1] y""[2] ... y""[n-1]]
'''
def deriv(x,y):
    D = zeros((4,1))
    D[0] = [x**2-4*y[0]]
    D[1] = [2*x-4*x**2+16*y[0]]
    D[2] = [2-8*x+16*x**2-64*y[0]]
    D[3] = [-8+32*x-64*x**2+256*y[0]]
```

```
    return D

x = 0.0         # Initial value of x
xStop = 0.2       # last value
y = array([1.0])           # Initial values of y
h = 0.1               # Step size
X,Y = taylor(deriv,x,y,xStop,h)

print("The required values are :at x= %0.2f, y=%0.5f, x=%0.2f, y=%0.5f,
                                    x = %0.2f, y=%0.5f"%(X[0],Y[0],X[1
                                   ],Y[1],X[2],Y[2]))
```

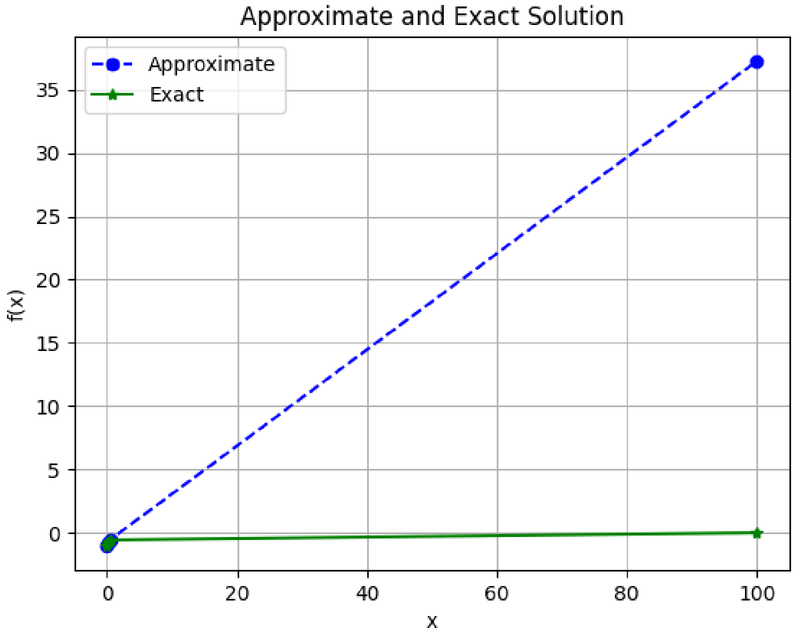
```
The required values are :at x= 0.00, y=1.00000, x=0.10, y=0.66967,
x = 0.20, y=0.45026
```

9.3 Euler's method to solve ODE:

To solve the ODE of the form $\frac{dy}{dx} = f(x, y)$ with initial conditions $y(x_0) = y_0$. The iterative formula is given by : $y(x_{(i+1)} = y(x_i) + hf(x_i, y(x_i))$.
Solve: $y' = e^{-x}$ with $y(0) = -1$ using Euler's method at $x = 0.2(0.2)0.6$.

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

# Define parameters
f = lambda x, y: np.exp(-x) # ODE
h = 0.2 # Step size
y0 = -1 # Initial Condition
n=3
# Explicit Euler Method

y[0] = y0
x[0]=0

for i in range(0, n):
    x[i+1]=x[i]+h
    y[i + 1] = y[i] + h*f(x[i], y[i])

print("The required values are at x= %0.2f, y=%0.5f, x=%0.2f, y=%0.5f,
                                    x = %0.2f, y=%0.5f,x = %0.2f, y=%0.
                                    5f"%(x[0],y[0],x[1],y[1],x[2],y[2],
                                     x[3],y[3]))
print("\n\n")

plt.plot(x, y, 'bo--', label='Approximate')
plt.plot(x, -np.exp(-x), 'g*-', label='Exact')
plt.title("Approximate and Exact Solution" )
plt.xlabel('x')
plt.ylabel('f(x)')
plt.grid()
plt.legend(loc='best')
plt.show()
```

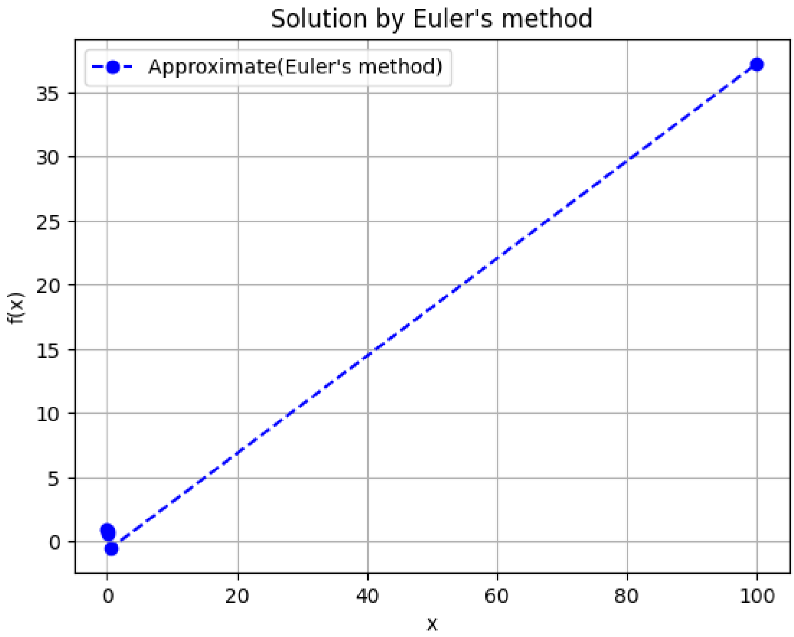
```
The required values are at x= 0.00, y=-1.00000, x=0.20, y=-0.80000,
x = 0.40, y=-0.63625,x = 0.60, y=-0.50219
```



Solve: $y' = -2y + x^3 e^{-2x}$ with $y(0) = 1$ using Euler's method at $x = 0.1, 0.2$.

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

# Define parameters
f = lambda x, y: -2*y+(x**3)*np.exp(-2*x) # ODE
h = 0.1 # Step size
y0 = 1 # Initial Condition
n=2
# Explicit Euler Method

y[0] = y0
x[0]=0

for i in range(0, n):
    x[i+1]=x[i]+h
    y[i + 1] = y[i] + h*f(x[i], y[i])


print("The required values are at x= %0.2f, y=%0.5f, x=%0.2f, y=%0.5f,x
      =%0.2f, y=%0.5f\n\n"%(x[0],y[0],x[1
      ],y[1],x[2],y[2]))

plt.plot(x, y, 'bo--', label="Approximate(Euler's method)")

plt.title("Solution by Euler's method")
plt.xlabel('x')
plt.ylabel('f(x)')
plt.grid()
plt.legend(loc='best')
plt.show()
```

```
The required values are at x= 0.00, y=1.00000, x=0.10, y=0.80000,
x=0.20, y=0.64008
```



9.4 Modified Euler's method

The iterative formula is:

$$y_1^{(n+1)} = y_0 + \frac{h}{2}[f(x_0, y_0) + f(x_1, y_1^{(n)})], \qquad n = 0, 1, 2, 3, \ldots,$$

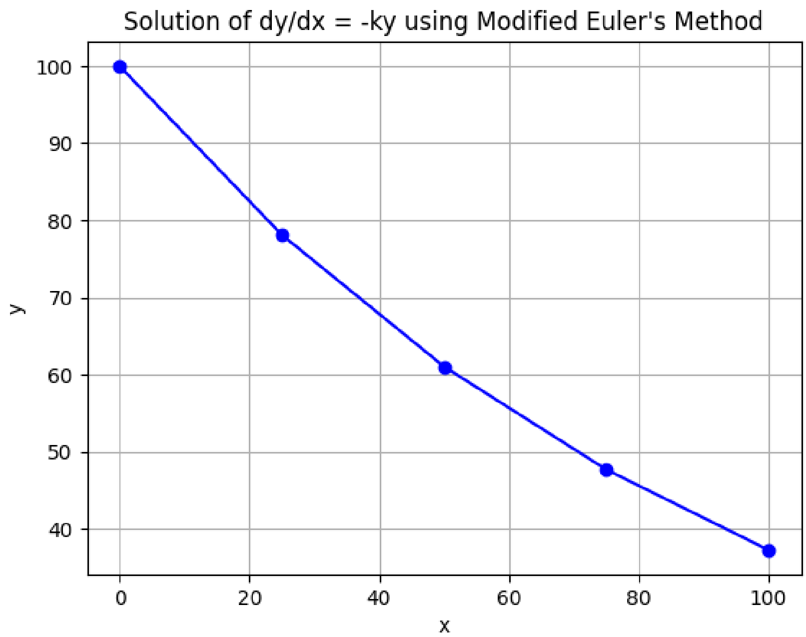
where, $y_1^{(n)}$ is the n^{th} approximation to y_1.

The first iteration will use Euler's method: $y_1^{(0)} = y_0 + hf(x_0, y_0)$.
Solve $y' = -ky$ with $y(0) = 100$ using modified Euler's method at $x = 100$, by taking $h = 25$.

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

def modified_euler(f, x0, y0, h, n):
    x = np.zeros(n+1)
    y = np.zeros(n+1)

    x[0] = x0
    y[0] = y0

    for i in range(n):
        x[i+1] = x[i] + h
        k1 = h * f(x[i], y[i])
        k2 = h * f(x[i+1], y[i] + k1)
        y[i+1] = y[i] + 0.5 * (k1 + k2)

    return x, y
```

```
def f(x, y):
    return -0.01 * y          # ODE dy/dx = -ky

x0 = 0.0
y0 = 100.0
h = 25
n = 4

x, y = modified_euler(f, x0, y0, h, n)

print("The required value at x= %0.2f, y=%0.5f"%(x[4],y[4]))
print("\n\n")

# Plotting the results
plt.plot(x, y, 'bo-')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Solution of dy/dx = -ky using Modified Euler\'s Method')
plt.grid(True)
plt.show()
```

```
The required value at x= 100.00, y=37.25290
```



9.5 Exercise:

1. Find $y(0.1)$ by Taylor Series exapnsion when $y' = x - y^2, y(0) = 1$.

   Ans: $y(0.1) = 0.9138$

2. Find $y(0.2)$ by Taylor Series exapnsion when $y' = x^2y - 1, y(0) = 1, h = 0.1$.

   Ans: $y(0.2) = 0.80227$

3. Evaluate by modified Euler's method: $y' = ln(x + y), y(0) = 2$ at $x = 0(0.2)0.8$.
   Ans: $2.0656, 2.1416, 2.2272, 2.3217$
4. Solve by modified Euler's method: $y' = x + y, y(0) = 1, h = 0.1, x = 0(0.1)0.3$.
   Ans: $1.1105, 1.2432, 1.4004$

LAB 10: Solution of ODE of first order and first degree by Runge-Kutta 4th order method and Milne's predictor and corrector method

10.1 Objectives:

1. To write a python program to solve first order differential equation using 4th order Runge Kutta method.

2. To write a python program to solve first order differential equation using Milne's predictor and corrector method.

10.2 Runge-Kutta method

Apply the Runge Kutta method to find the solution of $dy/dx = 1 + (y/x)$ at $y(2)$ taking $h = 0.2$. Given that $y(1) = 2$.

```
from sympy import *
import numpy  as np
def RungeKutta(g,x0,h,y0,xn):

  x,y=symbols('x,y')
  f=lambdify([x,y],g)
  xt=x0+h
  Y=[y0]
  while xt<=xn:
      k1=h*f(x0,y0)
      k2=h*f(x0+h/2, y0+k1/2)
      k3=h*f(x0+h/2, y0+k2/2)
      k4=h*f(x0+h, y0+k3)
      y1=y0+(1/6)*(k1+2*k2+2*k3+k4)
      Y.append(y1)
      #print('y(%3.3f'%xt,') is %3.3f'%y1)
      x0=xt
      y0=y1
      xt=xt+h
  return np.round(Y,2)
RungeKutta('1+(y/x)',1,0.2,2,2)
```

```
array([2.  , 2.62, 3.27, 3.95, 4.66, 5.39])
```

10.3 Milne's predictor and corrector method

Apply Milne's predictor and corrector method to solve $dy/dx = x^2 + (y/2)$ at y(1.4). Given that y(1)=2, y(1.1)=2.2156, y(1.2)=2.4649, y(1.3)=2.7514. Use corrector formula thrice.

```
# Milne's method to solve first order DE
# Use corrector formula thrice
x0=1
y0=2
```

```
y1=2.2156
y2=2.4649
y3=2.7514
h=0.1
x1=x0+h
x2=x1+h
x3=x2+h
x4=x3+h
def f(x,y):
  return x**2+(y/2)

y10=f(x0, y0)
y11=f(x1,y1)
y12=f(x2,y2)
y13=f(x3,y3)
y4p=y0+(4*h/3)*(2*y11-y12+2*y13)
print('predicted value of y4 is %3.3f'%y4p)
y14=f(x4,y4p);
for i in range(1,4):
  y4=y2+(h/3)*(y14+4*y13+y12);
  print('corrected value of y4 after \t iteration  %d  is \t %3.5f\t '%
                                    (i,y4))
  y14=f(x4,y4);
```

```
predicted value of y4 is 3.079
corrected value of y4 after         iteration  1  is           3.07940
corrected value of y4 after         iteration  2  is           3.07940
corrected value of y4 after         iteration  3  is           3.07940
```

In the next program, function will take all the inputs from the user and display the answer.
Apply Milne's predictor and corrector method to solve $dy/dx = x^2 + (y/2)$ at y(1.4). Given that y(1)=2, y(1.1)=2.2156, y(1.2)=2.4649, y(1.3)=2.7514. Use corrector formula thrice.

```
from sympy import *
def Milne(g,x0,h,y0,y1,y2,y3):
    x,y=symbols('x,y')
    f=lambdify([x,y],g)
    x1=x0+h
    x2=x1+h
    x3=x2+h
    x4=x3+h

    y10=f(x0, y0)
    y11=f(x1,y1)
    y12=f(x2,y2)
    y13=f(x3,y3)
    y4p=y0+(4*h/3)*(2*y11-y12+2*y13)
    print('predicted value of y4',y4p)
    y14=f(x4,y4p)
    for i in range(1,4):
        y4=y2+(h/3)*(y14+4*y13+y12)
        print('corrected value of y4 , iteration  %d '%i,y4)
```

```
        y14=f(x4,y4)
Milne('x**2+y/2',1,0.1,2,2.2156,2.4649,2.7514)
```

```
predicted value of y4 3.0792733333333335
corrected value of y4 , iteration  1  3.0793962222222224
corrected value of y4 , iteration  2  3.079398270370371
corrected value of y4 , iteration  3  3.079398304506173
```

Apply Milne's predictor and corrector method to solve $dy/dx = x - y^2$, y(0)=2 obtain y(0.8). Take h=0.2. Use Runge-Kutta method to calculate required initial values.

```
Y=RungeKutta('x-y**2',0,0.2,0,0.8)
print('y values from Runge -Kutta method:',Y)
Milne('x-y**2',0,0.2,Y[0],Y[1],Y[2],Y[3])
```

```
y values from Runge -Kutta method: [0.   0.02 0.08 0.18 0.3 ]
predicted value of y4 0.3042133333333334
corrected value of y4 , iteration  1  0.3047636165214815
corrected value of y4 , iteration  2  0.3047412758696499
corrected value of y4 , iteration  3  0.3047421836520892
```

10.4 Exercise:

1. Find $y(0.1)$ by Runge Kutta method when $y' = x - y^2, y(0) = 1$.

   Ans: $y(0.1) = 0.91379$

2. Evaluate by Runge Kutta method : $y' = log(x + y), y(0) = 2$ at $x = 0(0.2)0.8$.

   Ans: $2.155, 2.3418, 2.557, 2.801$

3. Solve by Milnes method: $y' = x + y$, y(0)=1, h=0.1, Calculate $y(0.4)$. Calculate required initial values from Runge Kutta method.

   Ans: 1.583649219