1) To find gradient of ϕ = x 2 y + 2xz − 4

```
[ ] from sympy.vector import *
    from sympy import symbols

    N = CoordSys3D('N')
    x, y, z = symbols('x y z')
    A = N.x**2 * N.y + 2 * N.x * N.z - 4

    delop = Del()
    display(delop(A))

    gradA = gradient(A)
    print(f"\nGradient of {A} is\n")
    display(gradA)
```

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

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

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

2. To find divergence of F⃗ = x 2 yzˆi + y 2 zxˆj + z 2xyˆk

```
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"\nDivergence of {A} is\n")
display(divergence(A))
```

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

```
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
```

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

3. To find curl of F⃗ = x 2 yzˆi + y 2 zxˆj + z 2xyˆ

```
[ ] 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"\nCurl of {A} is\n")
    display(curl(A))
```

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

```
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
```

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

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

```
from sympy import *
x = Symbol('x')
g = input('Enter the function ')  # e.g., x**3 - 2*x - 5
f = lambdify(x, g)
a = float(input('Enter a value: '))  # e.g., 2
b = float(input('Enter b value: '))  # e.g., 3
N = int(input('Enter number of iterations: '))  # e.g., 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('Iteration %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 value: 2
Enter b value: 3
Enter number of iterations: 5
Iteration 1      the root 2.059          function value -0.391

Iteration 2      the root 2.081          function value -0.147

Iteration 3      the root 2.090          function value -0.055

Iteration 4      the root 2.093          function value -0.020

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

5. 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 ')  # e.g., 3*x - cos(x) - 1
f = lambdify(x, g)
dg = diff(g)
df = lambdify(x, dg)
x0 = float(input('Enter the initial approximation: '))  # e.g., 1
n = int(input('Enter the number of iterations: '))  # e.g., 5

for i in range(1, n+1):
    x1 = x0 - (f(x0) / df(x0))
    print('Iteration %d \t the root %0.3f \t function value %0.3f \n' % (i, x1, f(x1)))
    x0 = x1
```

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

Iteration 2      the root 0.607          function value 0.000

Iteration 3      the root 0.607          function value 0.000

Iteration 4      the root 0.607          function value 0.000

Iteration 5      the root 0.607          function value -0.000
```

6.1. Trapezoidal Rule Evaluate R 5 0 1 1+x2

```
def my_func(x):
    return 1 / (1 + x ** 2)

def trapezoidal(x0, xn, n):
    h = (xn - x0) / n
    integration = my_func(x0) + my_func(xn)

    for i in range(1, n):
        k = x0 + i * h
        integration += 2 * my_func(k)

    integration *= h / 2
    return integration

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: "))

result = trapezoidal(lower_limit, upper_limit, sub_interval)
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: 6
Integration result by Trapezoidal method is: 1.374219468982665
```

6.2 simsons 1/3 rd rule lim 0 to 5 1/(1+x^2)

```python
def my_func(x):
    return 1 / (1 + x ** 2)

def simpson13(x0, xn, n):
    h = (xn - x0) / n
    integration = my_func(x0) + my_func(xn)

    for i in range(1, n):
        if i % 2 == 0:
            integration += 2 * my_func(x0 + i * h)
        else:
            integration += 4 * my_func(x0 + i * h)

    integration *= h * (1 / 3)
    return integration

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: "))

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: 6
Integration result by Simpson's 1/3 method is: 1.350901
```

6.3 simsons 3/8 rd rule lim 0 to 5 1/(1+x^2)

```python
def f(x):
    return 1 / (1 + x ** 2 )

def simpson38(x0, xn, n):
    h = (xn - x0) / n
    Integration = f(x0) + f(xn)

    for i in range(1, n):
        if i % 3 == 0:
            Integration += 2 * f(x0 + i * h)
        else:
            Integration += 3 * f(x0 + i * h)

    Integration *= 3 * h / 8
    return Integration

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: "))

result = simpson38(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: 6
Integration result by Simpson's 1/3 method is: 1.340634
```

7. 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)

        x0 = xt
        y0 = y1
        xt = xt + h

    return np.round(Y, 2)

result = RungeKutta('1+(y/x)', 1, 0.2, 2, 2)
print(result)
```

```
[2.   2.62 3.27 3.95 4.66 5.39]
```

8. 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

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]))
```

```
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
```