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

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

2) Evaluate the integral $\int_0^3 \int_0^{3-x} \int_0^{3-x-y} (xyz) dz dy dx$

```
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 )
```

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

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

1.4 Area of the region R in the polar form is $\int\int_R r dr d\theta$

4)

```
from sympy import *
r= Symbol ('r')
t= Symbol ('t')
a= Symbol ('a')
w3=2* integrate (r ,( r ,0 , a*( 1+cos ( t ) ) ) ,(t ,0 ,pi) )
pprint ( w3 )
```

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

```
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 )
```

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

```
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 )
```

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

```
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"\n Gradient of {A} is \n")
display ( gradA )
```

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

```
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 ) )
```

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

```
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 ) )
```

10)Find the image of vector (10, 0) when it is rotated by π/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]])\
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 ()
```

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

```
import numpy as np
A = np . array ([2 , 1 , 5 , 4])
B = np . array ([3 , 4 , 7 , 8]) \
output = np . dot(A , B )
print ( output )
```

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

```
import numpy as np
A = np . array ([2 , 1 , 5 , 4])
B = np . array ([3 , 4 , 7 , 8])
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 ')
```

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

```
from sympy import *
x= Symbol ('x')
g = input ('Enter the function ')
f= lambdify (x , g )
a= float ( input ('Enter a valus :') )
b= float ( input ('Enter b valus :') )
N=int( input ('Enter number of iterations :') )
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 ) ) );
```

14)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 ')
f= lambdify (x , g )
dg = diff ( g ) ;
df= lambdify (x , dg )
x0= float ( input ('Enter the intial approximation ') ) ;
n= int( input ('Enter the number of iterations ') ) ;
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 ) ) ) ;
x0 = x1
```

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 ) )

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 )+']= ') )
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 ()
t= symbols ('t')
f=[]
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 )
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 ) ;
```

15)

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 ")
n = int( input ('Enter number of data points : ') )
x = np . zeros (( n ) )
y = np . zeros (( n , n ) )
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 )+']= ') )
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 ()
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 )
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 ) ;
```

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

```
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 ) )
   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
   integration = 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 ))
```

17) 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 )
a = 0
b = 6
n = 6
result = simpsons_3_8_rule (f , a , b , n )
print ('%3.5f '% result )
```

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

```
from numpy import array
def taylor ( deriv ,x ,y , xStop , h ):
    X = []
    Y = []
    X . append ( x )
    Y . append ( y )
    while x < xStop :
        D = deriv (x , 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 )
        Y.append ( y )
    return array ( X ) ,array ( Y )
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
xStop = 0 . 3
y = array ([0 . 0])
h = 0 . 1
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] ) )
```

19) 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")

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 )
```

20) 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 )
RungeKutta ('1+(y/x)',1 , 0 .2 ,2 , 2 )
```

21) 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 )
```