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

1.2 Double and triple integration

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)
display(w3)
w4=integrate(x**2+y**2,x,y)
display(w4)
```

$$\frac{x^3y}{3}+\frac{xy^3}{3}$$

$$\frac{x^3y}{3}+\frac{xy^3}{3}$$

1.3 Area and Volume

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

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
```

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

---

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
```

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
```

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')
```

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

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

---

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))
```

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

```
 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
```

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

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))
```

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

```
 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
```

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

LAB 4: Computation of dimension for a vector space and graphical representation of linear trans-formation

---

4.2 Rank Nullity Theorem

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

```python
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("Nullity 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]]
Nullity 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 $(3, 1, 2)$.

```python
import numpy as np

# Define the vector space V
V = np.array([
    [1, 2, 3],
    [2, 3, 1],
    [3, 1, 2]])
# Find the dimension of V

dimension = V.shape[0]

print("Dimension of the matrix",dimension)
```

```
Dimension of the matrix 3
```

LAB 5: Computing the inner product and orthogo-nality

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
```

LAB 6: Solution of algebraic and transcendental equation by Regula-Falsi and 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 regular-falsi method. Correct to 3 decimal places.

```python
# 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
```

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

8.2 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
```

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

10.2 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])
```