# program to compute area,sufrace area and volume

## A

In [1]:
```
from sympy import*
x,y=symbols('x,y')
w=integrate(x**2+y**2,(y,0,x),(x,0,1))
print(w)
```

```
1/3
```

## B

In [2]:
```
from sympy import*
x,y=symbols('x,y')
w=integrate(x+y,(y,0,x),(x,0,1))
print(w)
```

```
1/2
```

## C

In [3]:
```
from sympy import*
x,y,z=symbols('x,y,z')
w=integrate(x*y*z,(z,0,3-x-y),(y,0,3-x),(x,0,3))
print(w)
```

```
81/80
```

# D

In [6]:
```
from sympy import*
x,y,z=symbols('x,y,z')
w=integrate(exp(x+y+z),(z,0,x+log(y)),(y,0,x),(x,0,log(2)))
print(w)
```

```
-19/9 + 8*log(2)/3
```

# 2

# Find the area of an ellipse by double integration

# A

In [8]:
```
from sympy import*
x,y=symbols('x,y')
a=4
b=6
w=4*integrate(1,(y,0,b/a*sqrt(a**2-x**2)),(x,0,a))
print(w)
```

```
24.0*pi
```

# B

## Find the area of positive quadrant of the circle

In [9]:
```
from sympy import*
x,y=symbols('x,y')
w=integrate(1,(y,0,sqrt(16-x**2)),(x,0,4))
print(w)
```

```
4*pi
```

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

In [10]:
```
from sympy import*
r,t,a=symbols('r,t,a')
area=2*integrate(r,(r,0,a*(1+cos(t))),(t,0,pi))
print(area)
```

```
3*pi*a**2/2
```

# 3

# A

# Find the volume of the tetrahedral bounded by the planes(x=0,y=0,z=0,x/a+y/b+z/c=1)

In [11]:
```
from sympy import*
x,y,z,a,b,c=symbols('x,y,z,a,b,c')
volume=integrate(1,(z,0,c*(1-x/a-y/b)),(y,0,b*(1-x/a)),(x,0,a))
print(volume)
```

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

# B Find the volume of tetrahedral bonded by the planes(x=0,y=0,z=0,x/2+y/3+z/4=1)

In [12]:
```
from sympy import*
x,y,z=symbols('x,y,z')
a=2
b=3
c=4
volume=integrate(1,(z,0,c*(1-x/2-y/3)),(y,0,b*(1-x/2)),(x,0,2))
print(volume)
```

```
4
```

# II

# Evaluation of beta and Gamma function

# 1) a)

In [16]:
```
from sympy import*
x=Symbol('x')
u=integrate(exp(-x),(x,0,oo))
print(u)
```

```
1
```

# 1) b

In [17]:
```
from sympy import*
t=Symbol('t')
u=integrate(exp(-t)*cos(2*t),(t,0,oo))
print(u)
```

```
1/5
```

# 1) C Evaluate Gamma(5) using definition gamma(5)

In [5]:
```
from sympy import*
x=Symbol('x')
Gamma=integrate(exp(-x)*x**4,(x,0,float('inf')))
print(simplify(Gamma))
```

```
24
```

# 2 a) Find beta(3,5) and gamma(5)

In [2]:

```
from sympy import beta,gamma
m=float(3)
n=float (5)
beta3_5=beta(m,n)
gamma5=gamma(n)
print("Beta (3,5)=",beta3_5)
print("Gamma (5)=",gamma5)
```

```
Beta (3,5)= 0.00952380952380952
Gamma (5)= 24.0000000000000
```

# 2 B) Find beta(5/2,7/2) and Gamma(5/2)

In [4]:

```
from sympy import beta,gamma
m=float(5/2)
n=float (7/2)
beta_value=beta(m,n)
gamma_value=gamma(m)
print("Beta (5/2,7/2)=",beta_value)
print("Gamma (5/2)=",gamma_value)
```

```
Beta (5/2,7/2)= 0.0368155389092554
Gamma (5/2)= 1.32934038817914
```

In [9]:
```
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(s==t):
    print("beta and gamma are related")
else:
    print("given values are wrong")
```

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

## Finding gradient ,divergence and curl

## To find gradient of phi=x^2y+2xz-4

In [15]:
```
from sympy.vector import*
from sympy import symbols
x,y,z=symbols('x,y,z')
N=CoordSys3D('N')
A=N.x**2*N.y+2*N.x*N.z-4
print("\n Gradient is:")
display(gradient(A))
```

```
 Gradient is:
```

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

# To find divergence of F=x^2yzî +Y^2zxĵ +z^2xyk̂

In [20]:
```
from sympy.vector import*
from sympy import symbols
x,y,z=symbols('x,y,z')
N=CoordSys3D('N')
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
print("\n Divergence is:")
display(divergence(A))
```

```
 Divergence is:
```

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

# To find curl of F=x^2yzî +Y^2zxĵ +z^2xyk̂

In [21]:
```
from sympy.vector import*
from sympy import symbols
x,y,z=symbols('x,y,z')
N=CoordSys3D('N')
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
print("\n curl is:")
display(curl(A))
```

```
 curl is:
```

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

## computing the inner product and orthogonality

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

In [1]:
```
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 of the vectors is",output)
```

```
Inner product of the vectors is 77
```

# 2) verify the vectors (2,1,5,4) and (3,4,7,8) are orthogonal

In [2]:
```
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 of the vectors is",output)
if output==0:
    print("Given vectors are orthogonal")
else:
    print("Given vectors are not orthogonal")
```

```
Inner product of the vectors is 77
Given vectors are not orthogonal
```

# Rank nulity theorem and dimension of the vector space

## 1) Verify the rank nullity theorem for the linear transformation by T(x,y,z)=(x+4y7z,2x+5y+8z,3x+6y+9z)

In [5]:
```
import numpy as np
from scipy.linalg import null_space
A=np.array([[1,2,3],[4,5,6],[7,8,9]])
rank=np.linalg.matrix_rank(A)
print("rank of the matrix ",rank)
ns=null_space(A)
print("null space of the matrix",ns)
nullity=ns.shape[1]
print("nullity of the matrix is",nullity )
if rank+nullity==A.shape[1]:
    print("Rank-nullity theorem hold")
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 is 1
Rank-nullity theorem hold
```

# 2) find the dimension of the subspace spanned by the vectors (1,2,3),(2,3,1) and (3,1,2)

In [6]:
```
import numpy as np
V=np.array([[1,2,3],[2,3,1],[3,1,2]])
dimension=np.linalg.matrix_rank(V)
print("Dimension of the subspace spanned by the vector is",dimension)
```

```
Dimension of the subspace spanned by the vector is 3
```

# Computation of area under the curve using trapezoidal ,1/3rd ,and 3/8th rule.

# 1a)

In [22]:
```
def y(x):
    return 1/(1+x**2)
xo=float(input("Enter the lower limit of integration:"))
xn=float(input("Enter the upper limit of integration:"))
n=int(input("Enter sub-intervals:"))
def trapezoidal(xo,xn,n):
    h=(xn-xo)/n
    sum=y(xo)+y(xn)
    for i in range(1,n):
        k=xo+i*h
        sum=sum+2*y(k)
        integration=sum*h/2
    return integration
print("Integration result by trapezoidal method is:%.6f"%trapezoidal(xo,xn,n))
```

```
Enter the lower limit of integration:0
Enter the upper limit of integration:5
Enter sub-intervals:6
Integration result by trapezoidal method is:1.374219
```

# 1b

In [28]:

```
from sympy import *
def y(x):
    return sin(x)**2
xo=0
xn=pi
n=6
def trapezoidal(xo,xn,n):
    h=(xn-xo)/n
    sum=y(xo)+y(xn)
    for i in range(1,n):
        k=xo+i*h
        sum=sum+2*y(k)
        integration=sum*h/2
    return integration
print("Integration result by trapezoidal method is:%.4f"%trapezoidal(xo,xn,n))
```

```
Integration result by trapezoidal method is:1.5708
```

## 2a

In [24]:
```
def y(x):
    return 1/(1+x**2)
xo=float(input("Enter the lower limit of integration:"))
xn=float(input("Enter the upper limit of integration:"))
n=int(input("Enter sub-intervals:"))
def simpson1_3 (xo,xn,n):
    h=(xn-xo)/n
    sum=y(xo)+y(xn)
    for i in range(1,n):
        k=xo+i*h
        if i%2==0:
            sum=sum+2*y(k)
        else:
            sum=sum+4*y(k)
        integration=sum*h/3
    return integration
result=simpson1_3(xo,xn,n)
print("Integration result by simpson method is:%.6f"%result)
```

```
Enter the lower limit of integration:0
Enter the upper limit of integration:5
Enter sub-intervals:6
Integration result by simpson method is:1.350901
```

# 2b

In [27]:
```
def y(x):
    return x**2/(1+x**3)
xo=0
xn=1
n=6
def simpson1_3 (xo,xn,n):
    h=(xn-xo)/n
    sum=y(xo)+y(xn)
    for i in range(1,n):
        k=xo+i*h
        if i%2==0:
            sum=sum+2*y(k)
        else:
            sum=sum+4*y(k)
        integration=sum*h/3
    return integration
result=simpson1_3(xo,xn,n)
print("Integration result by simpson1_3 method is:%.6f"%result)
```

```
Integration result by simpson1_3 method is:0.231057
```

# 3a)

In [19]:
```
def y(x):
    return 1/(1+x**2)
xo=0
xn=5
n=6
def simpson3_8 (xo,xn,n):
    h=(xn-xo)/n
    sum=y(xo)+y(xn)
    for i in range(1,n):
        k=xo+i*h
        if i%3==0:
            sum=sum+2*y(k)
        else:
            sum=sum+3*y(k)
        integration=sum*h*(3/8)
    return integration
result=simpson3_8(xo,xn,n)
print("Integration result by simpson3_8 method is:%.6f"%result)
```

```
Integration result by simpson3_8 method is:1.340634
```

# 3b)

In [21]:
```
from sympy import *
def y(x):
    return exp(-x**2)
xo=0
xn=0.6
n=6
def simpson3_8 (xo,xn,n):
    h=(xn-xo)/n
    sum=y(xo)+y(xn)
    for i in range(1,n):
        k=xo+i*h
        if i%3==0:
            sum=sum+2*y(k)
        else:
            sum=sum+3*y(k)
        integration=sum*h*(3/8)
    return integration
result=simpson3_8(xo,xn,n)
print("Integration result by simpson3_8 method is:%.6f"%result)
```

```
Integration result by simpson3_8 method is:0.535158
```

In [ ]:

# Solution of algebraic and transcendenatl equation by Newton Raphons method and Regula falsi

# REGULA FALSI

In [12]:
```
from sympy import *
x=Symbol('x')
fn=input("Enter the function")
f=lambdify(x,fn)
a=float(input("Enter the value of a:"))
b=float(input("Enter the value of b:"))
N=int(input("enter no 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)))
print("Hence x= %0.3f"%c)
```

```
Enter the functionx**3-2*x-5
Enter the value of a:2
Enter the value of b:3
enter no of iterations5
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

Hence x= 2.094
```

# 2 Newton Raphson

In [24]:
```
from sympy import *
x=Symbol('x')
fn=input("Enter the function:")
f=lambdify(x,fn)
d_fn=diff(fn)
d_f=lambdify(x,d_fn)
x0=float(input("enter the intial approximation:"))
N=int(input("enter no of iterations"))
for i in range (1,N+1):
    x1=(x0-(f(x0)/d_f(x0)))
    print("Iteration %d \t the root %0.3f \t function value %0.3f \n "%(i,x1,f(x1)))
    x0=x1
print("Hence x= %0.3f"%x1)
```

```
Enter the function:3*x-cos(x)-1
enter the intial approximation:1
enter no of iterations5
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

Hence x= 0.607
```

# 10 solution of ODE of 1st order and 1st degree

# 1 R-K method

In [13]:
```
from sympy import *
import numpy as np
def R_K(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)
R_K('1+y/x',1,0.2,2,2)
```

Out[13]:
```
array([2.  , 2.62, 3.27, 3.95, 4.66, 5.39])
```

# 2) solve y'=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

In [12]:

```
from sympy import *
x0=1
h=0.1
x1=x0+h
x2=x1+h
x3=x2+h
x4=x3+h
y0=2
y1=2.2156
y2=2.4649
y3=2.7514
def f(x,y):
    return x**2+(y/2)
f0=f(x0,y0)
f1=f(x1,y1)
f2=f(x2,y2)
f3=f(x3,y3)
y4P=y0+(4*h/3)*(2*f1-f2+2*f3)
print("Predicted value of y4 is: %0.3f"%y4P)
f4=f(x4,y4P)
for i in range (1,4):
    y4C=y2+(h/3)*(f2+4*f3+f4)
    print("The corrected  value y4 after \t iteration  %d \t %0.5f"%(i,y4C))
    f4=f(x4,y4C)
```

```
Predicted value of y4 is: 3.079
The corrected  value y4 after    iteration  1    3.07940
The corrected  value y4 after    iteration  2    3.07940
The corrected  value y4 after    iteration  3    3.07940
```

# solve y'-2y=3e^(x) with y(0)=0 bt taylor series method at x=0.1(0.1)0.3

In [42]:

```
from numpy import array,zeros,exp
x=0.0
xn=0.3
y=array([0.0])
h=0.1
def taylor(derivative,x,y,xn,h):
    X=[]
    Y=[]
    X.append(x)
    Y.append(y)
    while x<xn:
        D=derivative(x,y)
        H=1.0
        for j in range (3):
            H=H*h/(j+1)
            y=y+D[j]*H
        x=x+h
        X.append(x)
        Y.append(y)
    return array(X),array(Y)
def derivative (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,Y=taylor(derivative,x,y,xn,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[
```

```
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'=e^(x) with y(0)=-1 using Eulers method at x=0.2(0.2)0.6

In [40]:
```
import numpy as np
import matplotlib.pyplot as plt
f=lambda x,y:np.exp(-x)
h=0.2
y0=-1
n=3
Y=np.zeros(n+1)
X=np.zeros(n+1)
X[0]=0
Y[0]=y0
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[
```

```
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
```

# use newton Forward interpolation to obtain interpolating polynomial and hence calculate y(2)

In [8]:
```
from sympy import *
import numpy as np
n=int(input("Enter the number of data point:"))
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("\n Forward 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=Symbol('t')
f=[]
h=x[1]-x[0]
p=(t-x[0])/h
f.append(p)
for i in range (1,n-1):
    f.append(f[i-1]*(p-i)/(i+1))
sum=y[0][0]
for i in range (n-1):
    sum=sum+y[0][i+1]*f[i]
    sum=simplify(sum)
print('\n The interpolating polynomial is :\n')
display(sum)
inter=input("Do you want to interpolate at a point(yes/no)?")
if inter=='yes':
    a=float(input("Enter the x value or data point:"))
    y_value=lambdify(t,sum)
    result=y_value(a)
print("\n They y value at x=",a,'is:',result)
```

```
Enter the number of data point: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 :
```

$$1.0t^3 - 3.0t^2 + 1.0t + 7.0$$

```
Do you want to interpolate at a point(yes/no)?yes
Enter the x value or data point:2

 They y value at x= 2.0 is: 5.0
```

# use Newton's Backward interpolation formula to obtain the interpolating formula and hence calculate y(8)

In [9]:
```
from sympy import *
import numpy as np
n=int(input("Enter the number of data point:"))
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("\n Backward difference table \n")
for i in range(0,n):
    print("%0.2f "%(x[i]),end='')
    for j in range(0,i+1):
        print('\t \t %0.2f'%(y[i][j]),end='')
    print()
t=symbols('t')
f=[]
h=x[1]-x[0]
p=(t-x[n-1])/h
f.append(p)
for i in range (1,n-1):
    f.append(f[i-1]*(p+i)/(i+1))
sum=y[n-1][0]
for i in range(n-1):
    sum=sum+y[n-1][i+1]*f[i]
    sum=simplify(sum)
print('\n The interpolating polynomial is :\n')
display(sum)
inter=input("Do you want to interpolate at a point(yes/no)?")
if inter=='yes':
    a=float(input("Enter the x value or data point:"))
    y_value=lambdify(t,sum)
    result=y_value(a)
print("\n They y value at x=",a,'is:',result)
```

```
Enter the number of data point: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

 The interpolating polynomial is :
```

$$1.0t^3 - 3.0t^2 + 1.0t + 7.0$$

```
Do you want to interpolate at a point(yes/no)?yes
Enter the x value or data point:8

 They y value at x= 8.0 is: 335.0
```

In [ ]: