MALNAD COLLEGE OF ENGINEERING, HASSAN

(An Autonomous Institution Affiliated to VTU, Belgaum)

Autonomous Programmes

Bachelor of Engineering

DEPARTMENT OF MATHEMATICS

LAB MANUAL

II Semester

DEPARTMENT OF MATHEMATICS
MCE, HASSAN

Programming in Python

Sl. No	List of Programmes (SEM-1)	CO	PO	LEVEL
1.	Product of matrices & finding Inverse of a matrix.	5	1,2,5	4
2.	Solution of first order ordinary differential equation using Taylor series & Range-kutta method	5	1,2,5	4
3.	Solution of system of linear equations using Gauss-Seidal iteration method.	5	1,2,5	4
4.	Solution of first order ordinary differential equation using Milne's Predictor-Corrector method.	5	1,2,5	4
5.	Finding gradient, divergence and curl.	5	1,2,5	4
6.	Solution of higher order differential equations	5	1,2,5	4
7.	Verification of Green's theorem in vector integration.	5	1,2,5	4
8.	Numerical solution of simultaneous differential equations by Range-kutta method.	5	1,2,5	4
9.	Computation of area, volume and center of gravity.	5	1,2,5	4
10.	Solution of system of equations by Gauss elimination method.	5	1,2,5	4

Course outcome of Mathematical procedures using python programming.

CO 5 - At the end of course, students will be able to write the program in python for the mathematical procedures connected with calculus, numerical methods, differential equations, vector calculus and execute the same with correct output.

CO	PO 1	PO 2	PO 5
CO 5	3	2	1

Rubrics for Evaluation

Daily Evaluation (for 15 Marks)	Marks	CO	PO	Level
Manual Solving	4	CO 5	PO 1	L 3
Record writing & Observation	3	CO 5	PO 1	L 3
Executing the Programme with correct output	8	CO 5	PO 2, PO 5	L 4
Final CIE	5	CO 5	PO 2, PO 5	L 4

In [ ]:
```
Matrix Multiplication and Matrix Inverse
```

In [5]:
```
from numpy import *
A = [[12,7,3],[4,5,6],[7,8,9]]
B = [[5,8,1,2],[6,7,3,0],[4,5,9,1]]
r = dot(A,B)
print (r)
```

```
[[114 160  60  27]
 [ 74  97  73  14]
 [119 157 112  23]]
```

In [6]:
```
from numpy import *
A = [[12,7,3],[4,5,6],[7,8,9]]
B = linalg.inv(A)

print (B)
```

```
[[  1.          13.          -9.        ]
 [ -2.         -29.          20.        ]
 [  1.          15.66666667 -10.66666667]]
```

R-K FOURTH ORDER METHOD

In [1]:
```
from math import *
dy=lambda x,y:x+y
x=float(input('given x value='))
y=float(input('given y value='))
h=float(input('given step length='))
k1=h*dy(x,y)
k2=h*(dy(x+h/2,y+k1/2))
k3=h*(dy(x+h/2,y+k2/2))
k4=h*(dy(x+h,y+k3))
rk=y+1/6*(k1+2*k2+2*k3+k4)
display(rk)
```

```
given x value=0
given y value=1
given step length=0.2
```

```
1.2428
```

In [ ]:
```
Gauss-Siedal Iteration Method
```

In [2]:
```
from numpy import *
import sys

a=array([[10.0,1.0,1.0],[1.0,10.0,1.0],[1.0,1.0,10.0]])
x=array([[0.0],[0.0],[0.0]])
b=array([[12.0],[12.0],[12.0]])
n=len(a)
for i in range (0, n):
    asum = 0
    for j in range (0, n):
        if i!=j:
            asum= asum + abs(a[i][j])

    if (asum<=a[i][i]):
        continue
    else:
        sys.exit("The system is not diagonally dominant")

def seidel(a,x,b):
    for i in range (0,n):
        d=b[i]
        for j in range (0,n):
            if i!=j:
                d=d-a[i][j]*x[j]
        x[i]=d/a[i][i]
    return x

for i in range (0,3):
    x=seidel(a,x,b)
print(x)
```

```
[[0.9996492 ]
 [1.00001628]
 [1.00003345]]
```

In [ ]:
```
Milne's Predictor-Corrector Method
```

In [15]:
```
from numpy import *
def milnes_method(f,y0,x0,x_end,h):
    x=arange(x0,x_end+h,h)
    y=zeros(len(x))
    y[0]=y0
    for i in range(1,len(x)):
        y_pred=y[i-1]+((4*h)/3)*f(x[i-1],y[i-1])
        y_corrected=y[i-1]+(h/3)*(3*f(x[i],y_pred)-f(x[i-1],y[i-1]))
        y[i]=y_corrected
    return x,y
def f(x,y):
    return 2*exp(x)-y
y0=2
x0=0
x_end=0.4
h=0.1
x,y=milnes_method(f,y0,x0,x_end,h)
for i in range(len(x)):
    print('x=',x[i],'y=',y[i])
```

```
x= 0.0 y= 2.0
x= 0.1 y= 2.0210341836151295
x= 0.2 y= 2.0543769597672963
x= 0.30000000000000004 y= 2.100784359333292
x= 0.4 y= 2.1611206443233435
```

In [2]:
```
from numpy import *
def milnes_method(f,y0,x0,x_end,h):
    x=arange(x0,x_end+h,h)
    y=zeros(len(x))
    y[0]=y0
    for i in range(1,len(x)):
        y_pred=y[i-1]+((4*h)/3)*f(x[i-1],y[i-1])
        y_corrected=y[i-1]+(h/3)*(3*f(x[i],y_pred)-f(x[i-1],y[i-1]))
        y[i]=y_corrected
    return x,y
def f(x,y):
    return x-y**2
y0=0
x0=0
x_end=0.8
h=0.2
x,y=milnes_method(f,y0,x0,x_end,h)
for i in range(len(x)):
    print('x=',x[i],'y=',y[i])
```

```
x= 0.0 y= 0.0
x= 0.2 y= 0.04000000000000001
x= 0.4 y= 0.10504700359111112
x= 0.6000000000000001 y= 0.19039892590495636
x= 0.8 y= 0.28959607844602
```

In [ ]:
```
Vector Calculus
```

In [1]:
```
#To find gradient
from sympy.vector import *
from sympy import *
N=CoordSys3D('N')
x,y,z=symbols('x,y,z')
PHI=N.x**2*N.y+2*N.x*N.z-4
delop=Del()
display(delop(PHI))
gradPHI=gradient(PHI)
display(gradPHI)
```

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

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

In [2]:
```
#To find divergence
from sympy.vector import *
from sympy import *
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)
display(divergence(A))
```

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

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

In [1]:
```
#To find curl
from sympy.vector import *
from sympy import *
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)
display(curl(A))
```

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

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

In [ ]:
```
Solving Differential Equations
```

In [3]:
```
from sympy import *
x = symbols('x')
y = Function('y')(x)
c1,c2,c3,c4 = symbols('c1,c2,c3,c4')
Dy = diff(y,x)
D2y= diff(y,x,2)
D3y= diff(y,x,3)
D4y= diff(y,x,4)
de = Eq(D4y-18*D2y+81*y-36*exp(3*x),0)
display(de)
z = dsolve(de)
display(z)
```

$$81y(x) - 36e^{3x} - 18\frac{d^2}{dx^2}y(x) + \frac{d^4}{dx^4}y(x) = 0$$

$$y(x) = (C_1 + C_2x)\,e^{-3x} + \left(C_3 + x\left(C_4 + \frac{x}{2}\right)\right)e^{3x}$$

In [5]:
```
from sympy import *
x = symbols('x')
y = Function('y')(x)
c1,c2 = symbols('c1,c2')
Dy = diff(y,x)
D2y= diff(y,x,2)
de = Eq(D2y-4*y-3**x,0)
display(de)
z = dsolve(de)
display(z)
```

$$-3^x - 4y(x) + \frac{d^2}{dx^2}y(x) = 0$$

$$y(x) = \frac{3^x}{-4 + \log(3)^2} + C_1e^{-2x} + C_2e^{2x}$$

In [6]:
```
from sympy import *
x = symbols('x')
y = Function('y')(x)
c1,c2 = symbols('c1,c2')
Dy = diff(y,x)
D2y= diff(y,x,2)
de = Eq(D2y+4*Dy+4*y-3*sin(x)-4*cos(x),0)
display(de)
z = dsolve(de)
display(z)
```

$$4y(x) - 3\sin(x) - 4\cos(x) + 4\frac{d}{dx}y(x) + \frac{d^2}{dx^2}y(x) = 0$$

$$y(x) = (C_1 + C_2 x)\,e^{-2x} + \sin(x)$$

In [7]:
```
from sympy import *
x = symbols('x')
y = Function('y')(x)
c1,c2 = symbols('c1,c2')
Dy = diff(y,x)
D2y= diff(y,x,2)
de = Eq(D2y+9*y-cos(2*x)*cos(x),0)
display(de)
z = dsolve(de)
display(z)
```

$$9y(x) - \cos(x)\cos(2x) + \frac{d^2}{dx^2}y(x) = 0$$

$$y(x) = C_2\cos(3x) + \left(C_1 + \frac{x}{12}\right)\sin(3x) + \frac{\cos(x)}{16}$$

In [8]:
```
from sympy import *
x = symbols('x')
y = Function('y')(x)
c1,c2,c3 = symbols('c1,c2,c3')
Dy = diff(y,x)
D2y= diff(y,x,2)
D3y= diff(y,x,3)
de = Eq(D3y+8*y-x**4-2*x-1,0)
display(de)
z = dsolve(de)
display(z)
```

$$-x^4 - 2x + 8y(x) + \frac{d^3}{dx^3}y(x) - 1 = 0$$

$$y(x) = C_3 e^{-2x} + \frac{x^4}{8} - \frac{x}{8} + \left(C_1\sin\left(\sqrt{3}x\right) + C_2\cos\left(\sqrt{3}x\right)\right)e^x + \frac{1}{8}$$

In [1]:
```
from sympy import *
x = symbols('x')
y = Function('y')(x)
c1,c2,c3 = symbols('c1,c2,c3')
Dy = diff(y,x)
D2y= diff(y,x,2)
D3y= diff(y,x,3)
de = Eq(D3y+2*D2y+Dy-x**3,0)
display(de)
z = dsolve(de)
display(z)
```

$$-x^3 + \frac{d}{dx}y(x) + 2\frac{d^2}{dx^2}y(x) + \frac{d^3}{dx^3}y(x) = 0$$

$$y(x) = C_1 + \frac{x^4}{4} - 2x^3 + 9x^2 - 24x + (C_2 + C_3x)\,e^{-x}$$

In [10]:
```
from sympy import *
x = symbols('x')
y = Function('y')(x)
c1,c2 = symbols('c1,c2')
Dy = diff(y,x)
D2y= diff(y,x,2)
de = Eq(D2y-2*Dy+5*y-exp(2*x)*sin(x),0)
display(de)
z = dsolve(de)
display(z)
```

$$5y(x) - e^{2x}\sin(x) - 2\frac{d}{dx}y(x) + \frac{d^2}{dx^2}y(x) = 0$$

$$y(x) = \left(C_1\sin(2x) + C_2\cos(2x) + \frac{(2\sin(x) - \cos(x))\,e^x}{10}\right)e^x$$

In [11]:
```
from sympy import *
x = symbols('x')
y = Function('y')(x)
c1,c2,c3 = symbols('c1,c2,c3')
Dy = diff(y,x)
D2y= diff(y,x,2)
D3y= diff(y,x,3)
de = Eq(D3y+y-5*exp(x)*x**2,0)
display(de)
z = dsolve(de)
display(z)
```

$$-5x^2e^x + y(x) + \frac{d^3}{dx^3}y(x) = 0$$

$$y(x) = C_3e^{-x} + \left(C_1\sin\left(\frac{\sqrt{3}x}{2}\right) + C_2\cos\left(\frac{\sqrt{3}x}{2}\right)\right)e^{\frac{x}{2}} + \frac{5\cdot\left(2x^2 - 6x + 3\right)e^x}{4}$$

In [12]:
```
from sympy import *
x = symbols('x')
y = Function('y')(x)
c1,c2 = symbols('c1,c2')
Dy = diff(y,x)
D2y= diff(y,x,2)
de = Eq(D2y-y-x**2*cos(x),0)
display(de)
z = dsolve(de)
display(z)
```

$$-x^2\cos(x) - y(x) + \frac{d^2}{dx^2}y(x) = 0$$

$$y(x) = C_1e^{-x} + C_2e^{x} - \frac{x^2\cos(x)}{2} + x\sin(x) + \frac{\cos(x)}{2}$$

In [ ]:
```
Green's Theorem
```

In [3]:
```
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
```

In [4]:
```
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
```