NAGARJUNA COLLEGE OF ENGINEERING AND TECHNOLOGY

(An autonomous institution under VTU)

Mudugurki Village, Venkatagirikote Post, Devanahalli Taluk,

Bengaluru – 562164

DEPARTMENT OF MATHEMATICS

Lab Component Of 2nd Semester Engineering Mathematics

Advanced Calculus And Numerical Methods

(Course Code - 22MATS21/22MATE21/22MATC21)

1	Finding gradient and it's geometrical interpretation.
2	Finding Divergence and it's geometrical interpretation.
3	Finding Curl and it's geometrical interpretation.
4	Verification of Green's theorem.
5	Solution of algebraic and transcendental equation by Regula-Falsi method.
6	Interpolation using Newton's forward interpolation formula.
7	Interpolation using Newton's backward interpolation formula
8	Computation of area under the curve using Trapezoidal, Simpson's (1/3)rd and Simpson's (3/8)th rule
9	Solutions of higher order ordinary differential equations
10	Solutions of Partial Differential Equations

1) **Write the Python code and Execute the program to find gradient of following scalar functions.**

i) x^2yz .

Method-1

```
from sympy.physics.vector import *
from sympy import var
var('x,y,z')
v=ReferenceFrame('v')
F=v[0]**2*v[1]*v[2]
G=gradient(F,v)
F=F.subs([(v[0],x), (v[1],y), (v[2],z)])
print("Given scalar function F = ")
display(F)
G=G.subs([(v[0],x), (v[1],y), (v[2],z)])
print("\n Gradient of F=")
display(G)
```

Output:-

Given scalar function F =

$\mathbf{x^2 yz}$

Gradient of F=

$(\mathbf{2xyz})\widehat{v_x} + (\mathbf{x^2\ z})\ \widehat{v_y} + (\mathbf{x^2\ y})\ \widehat{v_z}$

Method-2

```
from sympy.vector import*
from sympy import symbols
C=CoordSys3D('')
f=C.x**2*(C.y)*(C.z)
print("\n The given scalar point function is \n")
display(f)
delop=Del()
display(delop(f))
gradf=gradient(f)
print("\n Gradient of {f} is \n")
display(gradf)
```

Output:-

The given scalar point function is

$\mathbf{x^2yz}$

$\frac{\partial}{\partial x}\mathbf{x}^2\mathbf{yz})\hat{\imath}$ + $(\frac{\partial}{\partial y}\mathbf{x}^2\mathbf{yz})\hat{\jmath}$ + $(\frac{\partial}{\partial z}\mathbf{x}^2\mathbf{yz})\hat{k}$

Gradient of {f} is

$(2\mathbf{xyz})\hat{\imath}+(\mathbf{x}^2\mathbf{z})\hat{\jmath}+(\mathbf{x}^2\mathbf{y})\hat{k}$

ii) $3x^2yz$

```
from sympy.physics.vector import *
from sympy import var
var('x,y,z')
v=ReferenceFrame('v')
F=3*v[0]**2*v[1]*v[2]
G=gradient(F,v)
F=F.subs([(v[0],x), (v[1],y), (v[2],z)])
print("Given scalar function F = ")
display(F)
G=G.subs([(v[0],x), (v[1],y), (v[2],z)])
print("\n Gradient of F=")
display(G)
```

Output:- Given scalar function F =

$3x^2yz$

Gradient of F=

$(6xyz)\hat{\mathbf{v}}_{\mathbf{x}}+(3x^2z)\hat{\mathbf{v}}_{\mathbf{y}}+3x^2y\hat{\mathbf{v}}_{\mathbf{z}}$

iii) $\mathbf{x^2\,y^3\,z^4}$

```
from sympy.physics.vector import *
from sympy import var
var('x,y,z')
v=ReferenceFrame('v')
F=v[0]**2*v[1]**3*v[2]**4
G=gradient(F,v)
F=F.subs([(v[0],x), (v[1],y), (v[2],z)])
print("Given scalar function F = ")
display(F)
G=G.subs([(v[0],x), (v[1],y), (v[2],z)])
print("\n Gradient of F=")
display(G)
```

Output:- Given scalar function F =

$x^2y^3z^4$

Gradient of F=

$(2\mathbf{x}\mathbf{y}^3\mathbf{z}^4)\ \widehat{v_x} + (3\mathbf{x}^2\mathbf{y}^2\mathbf{z}^4)\ \widehat{v_y} + (4\mathbf{x}^2\mathbf{y}^3\mathbf{z}^3)\ \widehat{v_z}$

2) Write Python code and Execute the program to find divergence of following vector functions.

i) $\vec{F} = x^2y\hat{i} + yz^2\hat{j} + x^2z\hat{k}$

```
from sympy.physics.vector import *
from sympy import var
var('x,y,z')
v=ReferenceFrame('v')
F=v[0]**2*v[1]*v.x+v[1]*v[2]**2*v.y+v[0]**2*v[2]*v.z
G=divergence(F,v)
F=F.subs([(v[0],x), (v[1],y), (v[2],z)])
print("Given vector point function is ")
display(F)
G=G.subs([(v[0],x), (v[1],y), (v[2],z)])
print("\n Divergence of F=")
display(G)
```

Output:- Given vector point function is

$(x^2y)\hat{v}\ x + (yz^2)\hat{v}\ y + (x^2z)\hat{v}\ z$

Divergence of F=

$x^2+2xy+z^2$

ii) $\vec{F} = 2x^2z\hat{\imath} - xy^2z\hat{\jmath} + 3yz^2\hat{k}$

```
from sympy.physics.vector import *
from sympy import var
var('x,y,z')
v=ReferenceFrame('v')
F=2*v[0]**2*v[2]*v.x-v[0]*v[1]**2*v[2]*v.y+3*v[1]*v[2]**2*v.z
G=divergence(F,v)
F=F.subs([(v[0],x), (v[1],y), (v[2],z)])
print("Given vector point function is ")
display(F)
G=G.subs([(v[0],x), (v[1],y), (v[2],z)])
print("\n Divergence of F=")
```

```
display(G)
```

Output:- Given vector point function is

$2x^2z\hat{v}\,x - xy^2z\hat{v}\,y + 3yz^2\hat{v}\,z$

Divergence of F=

4xz-2xyz+6yz

iii) $\vec{F} = xyz\hat{\imath} + 3x^2y\hat{\jmath} + (xz^2 - y^2z)\hat{k}$

```
from sympy.physics.vector import *
from sympy import var
var('x,y,z')
v=ReferenceFrame('v')
F=v[0]*v[1]*v[2]*v.x+3*v[0]**2*v[1]*v.y+(v[0]*v[2]**2-v[1]**2*v[2])*v.z
G=divergence(F,v)
F=F.subs([(v[0],x), (v[1],y), (v[2],z)])
print("Given vector point function is ")
display(F)
G=G.subs([(v[0],x), (v[1],y), (v[2],z)])
print("\n Divergence of F=")
display(G)
```

Output:- Given vector point function is

$xyz\hat{\imath} + 3x^2y\hat{\jmath} + (xz^2 - y^2z)\hat{k}$

Divergence of F=

$3x^2 + 2xz - y^2 + yz$

3) Write the Python code and Execute program to find curl of given vector point function.

i) $\vec{F} = xy^2\hat{\imath} + 2x^2yz\,\hat{\jmath} - 3yz^2\,\hat{k}$ **.**

```
from sympy.physics.vector import *
from sympy import var
var('x,y,z')
v=ReferenceFrame('v')
F=v[0]*v[1]**2*v.x+2*v[0]**2*v[1]*v[2]*v.y-3*v[1]*v[2]**2*v.z
G=curl(F,v)
F=F.subs([(v[0],x), (v[1],y), (v[2],z)])
print("Given vector point function is ")
display(F)
G=G.subs([(v[0],x), (v[1],y), (v[2],z)])
print("\n Curl of F=")
```

```
display(G)
```

Output:- Given vector point function is

$$xy^2\ \hat{\mathbf{v}}\mathbf{x} + 2x^2yz\hat{\mathbf{v}}\mathbf{y} - 3yz^2\ \hat{\mathbf{v}}\mathbf{z}$$

```
Curl of F=
```

$$(-2x^2y - 3z^2)\hat{\mathbf{v}}\mathbf{x} + (4xyz - 2xy)\hat{\mathbf{v}}\mathbf{z}$$

ii) $\vec{F} = x^2yz\,\hat{\imath} + 2xy^2z\,\hat{\jmath} + xyz^2\,\hat{k}$

```
from sympy.physics.vector import *
from sympy import var
var('x,y,z')
v=ReferenceFrame('v')
F=v[0]**2*v[1]*v[2]*v.x+2*v[0]*v[1]**2*v[2]*v.y+v[0]*v[1]*v[2]**2*v.z
G=curl(F,v)
F=F.subs([(v[0],x), (v[1],y), (v[2],z)])
print("Given vector point function is ")
display(F)
G=G.subs([(v[0],x), (v[1],y), (v[2],z)])
print("\n Curl of F=")
display(G)
```

Output:- Given vector point function is

$$x^2yz\mathbf{v}\hat{}\mathbf{x} + 2xy^2z\mathbf{v}\hat{}\mathbf{y} + xyz^2\mathbf{v}\hat{}\mathbf{z}$$

```
Curl of F=
```

$$(-2xy^2 + xz^2)\ \hat{\mathbf{v}}\mathbf{x} + (x^2y - yz^2)\ \hat{\mathbf{v}}\mathbf{y} + (-x^2z + 2y^2z)\hat{\mathbf{v}}\mathbf{z}$$

iii) $\vec{\mathbf{F}} = \mathbf{2x^2z\,\hat{\imath} - xy^2z\,\hat{\jmath} + 3yz^2\,\hat{k}}$

```
from sympy.physics.vector import *

from sympy import var
var('x,y,z')
v=ReferenceFrame('v')
F=2*v[0]**2*v[2]*v.x-v[0]*v[1]**2*v[2]*v.y+3*v[1]*v[2]**2*v.z
G=curl(F,v)
F=F.subs([(v[0],x), (v[1],y), (v[2],z)])
print("Given vector function F = ")
display(F)
G=G.subs([(v[0],x), (v[1],y), (v[2],z)])
print("\n Curl of F=")
display(G)
```

Output:-

Given vector point function is

$2x^2z\,\hat{v}\,x - xy^2z\,\hat{v}\,y + 3yz^2\hat{v}\,z$

Curl of F=

$(xy^2 + 3z^2)\,\mathbf{v}\hat{\ }\mathbf{x} + 2x^2\,\mathbf{v}\hat{\ }\mathbf{y} - y^2z\mathbf{v}\hat{\ }\mathbf{z}$

4) a) Using Green's theorem, Evaluate $\oint_c[(x+2y)\,dx + (x-2y)dy]$ where C is the region bounded by coordinate axes and the line x=1 and y=1.

```
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)
```

Output:-

I= -1

b) Using Green's theorem, Evaluate $\oint_c[(xy+y^2)\,dx + (x^2)dy]$ where C is the closed curve bounded by y=x and y=x² .

```
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)
```

Output:-

I= -1/20

c)Using Green's theorem, Evaluate $\oint_c [(3x+4y)\,dx + (2x-3y)dy]$ where C is the boundary of the circle $x^2 + y^2 = 4$

```
from sympy import *
var('x,y')
p=3*x+4*y
q=2*x-3*y
f=diff(q,x)-diff(p,y)
soln=integrate(f,[y,-sqrt(4-x**2),sqrt(4-x**2)],[x,-2,2])
print("I=",soln)
```

Output:-

```
I= -8*pi
```

5) a) Write the Python code and Execute the program for Finding a real root of the equation x^3 – 5x - 7=0 by using Regula - Falsi method correct to three decimal places.

```
from numpy import*
def f(x):
return x**3-5*x-7
n=int(input("Number of iterations = "))
a=float(input("Enter  the value of  a = "))
b=float(input("Enter  the value of  b = "))
E=0.000001
if f(a)*f(b)>0:
print("The given equation do not possesses the root between a and b")
exit()
print('___________________________')
print('Iteration\t\t  a  \t\t  b  \t\t  c  \t\t f(c)'  )
print('_______________________________')
 for  i in range(n):
c=(a*f(b)-b*f(a))/(f(b)-f(a))
print(str(i+1)+'\t\t%10.8f\t%10.8f\t%10.8f\t%10.8f'%(a,b,c,f(c)))
if abs(f(c))<E:
print('________________')
print('Root found : '+str(c))
if f(a)*f(c)<0:
```

```
b=c
else:
a=c
print("______________________')
if  i==n-1:
print('\n\n Max iterations reached!')
print('Approximation to the Root after max iterations is :'+str(c))
```

Output:-

Number of iterations = 5
Enter the value of a = 2
Enter the value of b = 3

____

Iteration	a	b	c	f(c)
1	2.00000000	3.00000000	2.64285714	-1.75473761
2	2.64285714	3.00000000	2.73563528	-0.20550152
3	2.73563528	3.00000000	2.74607181	-0.02247761
4	2.74607181	3.00000000	2.74720824	-0.00243995
5	2.74720824	3.00000000	2.74733154	-0.00026464

____

Max iterations reached!
Approximation to the Root after max iterations is :2.7473315411888417

5) b) Write the Python code and Execute the program for Finding a real root of the equation $x^3 - 2x - 5 = 0$ by the method of false position correct to three decimal places.

```
from numpy import*
def f(x):
return x**3 - 2*x - 5
n=int(input("Number of iterations = "))
a=float(input("Enter  the value of  a = "))
b=float(input("Enter  the value of  b = "))
E=0.000001
if f(a)*f(b)>0:
print("The given equation do not possesses the root between a and b")
exit()
print('___________________________')
print('Iteration\t\t  a  \t\t  b  \t\t  c  \t\t f(c)'  )
```

```
print('____________________________')
 for  i in range(n):
c=(a*f(b)-b*f(a))/(f(b)-f(a))
print(str(i+1)+'\t\t%10.8f\t%10.8f\t%10.8f\t%10.8f'%(a,b,c,f(c)))
if abs(f(c))<E:
print('_______________')
print('Root found : '+str(c))
if f(a)*f(c)<0:
b=c
else:
a=c
print("_______________________')
if  i==n-1:
print('\n\n Max iterations reached!')
print('Approximation to the Root after max iterations is :'+str(c))
```

Output:-

```
Number of iterations = 6
Enter  the value of  a = 2
Enter  the value of  b = 3
________________________________
Iteration            a            b            c            f(c)
_________________________
1            2.00000000   3.00000000   2.05882353   -0.39079992
2            2.05882353   3.00000000   2.08126366   -0.14720406
3            2.08126366   3.00000000   2.08963921   -0.05467650
4            2.08963921   3.00000000   2.09273957   -0.02020287
5            2.09273957   3.00000000   2.09388371   -0.00745051
6            2.09388371   3.00000000   2.09430545   -0.00274567
_____________________

Max iterations reached!
Approximation to the Root after max iterations is :2.09430545112526
```

5) c) Write the Python code and Execute the program for Finding a real root of the equation cos(x) = 3x - 1 by using Regula - Falsi method correct to three decimal places.

```
 from numpy import*
def f(x):
return cos (x) – 3*x + 1
n=int(input("Number of iterations = "))
```

```
a=float(input("Enter  the value of  a = "))
b=float(input("Enter  the value of  b = "))
E=0.000001
if f(a)*f(b)>0:
print("The given equation do not possesses the root between a and b")
exit()
print('____________________________')
print('Iteration\t\t  a  \t\t  b  \t\t  c  \t\t f(c)'  )
print('____________________________')
 for  i in range(n):
c=(a*f(b)-b*f(a))/(f(b)-f(a))
print(str(i+1)+'\t\t%10.8f\t%10.8f\t%10.8f\t%10.8f'%(a,b,c,f(c)))
if abs(f(c))<E:
print('_______________')
print('Root found : '+str(c))
if f(a)*f(c)<0:
b=c
else:
a=c
print("______________________')
if  i==n-1:
print('\n\n Max iterations reached!')
print('Approximation to the Root after max iterations is :'+str(c))
```

Output:-

```
Number of iterations = 6
Enter  the value of  a = 0
Enter  the value of  b = 1
____________________________________
Iteration            a             b             c            f(c)
____________________________
1             0.00000000   1.00000000   0.57808519   0.10325491
2             0.57808519   1.00000000   0.60595857   0.00408080
3             0.60595857   1.00000000   0.60705710   0.00015905
4             0.60705710   1.00000000   0.60709991   0.00000620
5             0.60709991   1.00000000   0.60710158   0.00000024
_______________
Root found : 0.6071015805150821
6             0.60710158   1.00000000   0.60710165   0.00000001
_______________
Root found : 0.6071016454704268
```

____

```
 Max iterations reached!
Approximation to the Root after max iterations is :0.607101645470426
8
```

6) Write the Python code and Execute the program for Finding y from the following data at x=5 by using Newton Forward Interpolation formula.

X	4	6	8	10
Y	1	3	8	16

```
from numpy import*
n=int(input("Enter number of datapoints:"))
x=zeros((n))
y=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()
#Newton's Forward Interpolation Formula
V=float(input("Enter the value to be interpolated: "))
def u_cal(u,n):
temp=u;
for i in range(1,n):
temp=temp*(u – i);
return temp;
def fact(n):
```

```
f=1;
for i in range(2,n+1):
f*=i
return f;
sum=y[0][0];
u=(v-x[0])/(x[1]-x[0]);
for i in range(1,n):
sum=sum+( u_cal(u,i)*y[0][i])/fact(i);
print("\n Interpolated value at ", v,"is", round(sum, 6));
```

Output:-

```
Enter number of datapoints:4
Enter data for x and y:
x[0]=4
y[0]=1
x[1]=6
y[1]=3
x[2]=8
y[2]=8
x[3]=10
y[3]=16

 FORWARD DIFFERENCE TABLE

4.00         1.00         2.00         3.00         0.00
6.00         3.00         5.00         3.00
8.00         8.00         8.00
10.00        16.00
Enter the value to be interpolated: 5

 Interpolated value at  5.0 is 1.625
```

7) Write the Python code and Execute the program for Finding y from the following data at x=2.65 by using Newton Backward Interpolation formula.

X	-1	0	1	2	3
Y	-21	6	15	12	3

```
from numpy import*
n=int(input("Enter number of data  points:"))
x=zeros((n))
y=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-1,-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()
#Newton's Backward Interpolation Formula
V=float(input("Enter the value to be interpolated: "))
def u_cal(u,n):
temp=u;
for i in range(1,n):
temp=temp*(u + i);
return temp;
def fact(n):
f=1;
for i in range(2,n+1):
f  *=i
return f
sum=y[n-1][0]
u=(v-x[n-1])/(x[1]-x[0]);
for i in range(1,n):
sum=sum+( u_cal(u,i)*y[n-1][i])/fact(i);
print("\n Interpolated value at ", v,"is", round(sum, 6));
```

Output:-

```
Enter number of datapoints:5
Enter data for x and y:
x[0]=-1
y[0]=-21
x[1]=0
y[1]=6
x[2]=1
y[2]=15
x[3]=2
```

```
y[3]=12
x[4]=3
y[4]=3

BACKWARD DIFFERENCE TABLE

-1.00  -21.00
0.00   6.00   27.00
1.00   15.00  9.00   -18.00
2.00   12.00  -3.00  -12.00 6.00
3.00   3.00   -9.00  -6.00  6.00   0.00
Enter the value to be interpolated: 2.65

 Interpolated value at  2.65 is 6.457125
```

8) a)Write Python code and Execute the program to evaluate $\int_0^1 \sqrt{1+x^3}$ dx using i)Simpson's 1/3 rd ii)Trapezoidal rule iii)Simpson's 3/8 rule

```
from numpy import *
a=0
b=1
p=6
n=7
h=(b-a)/p
x=linspace(a,b,n)
f=sqrt(1+x**3)
for i in range(0,n):
   print(round(f[i],ndigits=4))
TR=round((h/2)*(f[0]+2*sum(f[1:n-1])+f[n-1]),ndigits=4)
S13=round((h/3)*((f[0]+f[n-1])+4*sum(f[1:n-1:2]) +2*sum(f[2:n-1:2])),
ndigits=4)
S38=round((3*h/8)*((f[0]+f[n-1])+2*sum(f[3:n-2:3])
+3*sum(f[1:n:3]+(f[2:n-1:3]))),ndigits=4)
print("The required integral by Trapezoidal Rule is",TR)
print("The required integral by Simpson's 1/3 Rule is",S13)
print("The required integral by Simpson's 3/8 Rule is :",S38)
```

Output:-

```
1.0
1.0023
1.0184
1.0607
1.1386
1.2565
```

```
1.4142
The required integral by Trapezoidal Rule is 1.1139
The required integral by Simpson's 1/3  Rule is 1.1114
The required integral by Simpson's 3/8  Rule is 1.1114
```

8) b)Write the Python code and Execute the program to evaluate $\int_0^1 \frac{dx}{1+x^2}$ using i)Simpson's 1/3 rd rule ii)Trapezoidal Rule iii)Simpson's 3/8 rule

```
from numpy import *
a=0
b=1
p=6
n=7
h=(b-a)/p
x=linspace(a,b,n)
f=(1/(1+x**2))
for i in range(0,n):
   print(round(f[i],ndigits=4))
TR=round((h/2)*(f[0]+2*sum(f[1:n-1])+f[n-1]),ndigits=4)
S13=round((h/3)*((f[0]+f[n-1])+4*sum(f[1:n-1:2]) +2*sum(f[2:n-1:2])), ndigits=4)
S38=round((3*h/8)*((f[0]+f[n-1])+2*sum(f[3:n-2:3]) +3*sum(f[1:n:3]+(f[2:n-1:3]))),
ndigits=4)
print("The required integral by Trapezoidal Rule is",TR)
print("The required integral by Simpson's 1/3 Rule is",S13)
print("The required integral by Simpson's 3/8 Rule is :",S38)
```

Output:-

```
1.0
0.973
0.9
0.8
0.6923
0.5902
0.5
The required integral by Trapezoidal Rule is 0.7842
The required integral by Simpson's 1/3  Rule is 0.7854
The required integral by Simpson's 3/8  Rule is 0.7854
```

8) c) Write the Python code and Execute the program to evaluate $\int_0^6 \frac{dx}{(1+x)^2}$ using i)Simpson's 1/3 rd rule ii)Trapezoidal Rule iii)Simpson's 3/8 rule

```
from numpy import*
a=0
b=6
p=6
```

```
n=7
h=(b-a)/p
x=linspace(a,b,n)
f=(1/(1+x)**2)
for i in range(0,n):
   print(round(f[i],ndigits=4))
TR=round((h/2)*(f[0]+2*sum(f[1:n-1])+f[n-1]),ndigits=4)
S13=round((h/3)*((f[0]+f[n-1])+4*sum(f[1:n-1:2]) +2*sum(f[2:n-1:2])), ndigits=4)
S38=round((3*h/8)*((f[0]+f[n-1])+2*sum(f[3:n-2:3]) +3*sum(f[1:n:3]+(f[2:n-1:3]))),
ndigits=4)
print("The required integral by Trapezoidal Rule is",TR)
print("The required integral by Simpson's 1/3 Rule is",S13)
print("The required integral by Simpson's 3/8 Rule is :",S38)
```

Output:-

```
1.0
0.25
0.1111
0.0625
0.04
0.0278
0.0204
The required integral by Trapezoidal Rule is 1.0016
The required integral by Simpson's 1/3  Rule is 0.8946
The required integral by Simpson's 3/8  Rule is 0.912
```

8) d)Write the Python code and Execute the program to evaluate $\int_0^1 \frac{dx}{(1+x)}$ using i)Simpson's 1/3 rd rule ii)Trapezoidal Rule iii)Simpson's 3/8 rule

```
from numpy import*
a=0
b=1
p=6
n=7
h=(b-a)/p
x=linspace(a,b,n)
f=(1/(1+x))
for i in range(0,n):
   print(round(f[i],ndigits=4))
TR=round((h/2)*(f[0]+2*sum(f[1:n-1])+f[n-1]),ndigits=4)
S13=round((h/3)*((f[0]+f[n-1])+4*sum(f[1:n-1:2]) +2*sum(f[2:n-1:2])),ndigits=4)
S38=round((3*h/8)*((f[0]+f[n-1])+2*sum(f[3:n-2:3]) +3*sum(f[1:n:3]+(f[2:n-1:3]))),
ndigits=4)
```

```
print("The required integral by Trapezoidal Rule is",TR)
print("The required integral by Simpson's 1/3 Rule is",S13)
print("The required integral by Simpson's 3/8 Rule is :",S38)
```

Output:-

```
1.0
0.8571
0.75
0.6667
0.6
0.5455
0.5
The required integral by Trapezoidal Rule is 0.6949
The required integral by Simpson's 1/3  Rule is 0.6932
The required integral by Simpson's 3/8  Rule is 0.6932
```

8) e) 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)
```

Output:-

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

9) a) Write the Python code and Execute :: (4 D^2 - 4D -3)y =e^{2x}

```
from sympy import *
x=symbols('x')
y=Function('y')(x)
DE=Eq(4*Derivative(y,x,2) - 4*Derivative(y,x) -3*y , exp(2*x))
GS=dsolve(DE)
display(GS)
```

Output:-

$$y(x) = c_1 e^{\frac{3x}{2}} + c_2 e^{-\frac{x}{2}} + \frac{e^{2x}}{5}$$

9) b) Write the Python code and Execute :: $\frac{d^2y}{dx^2} - 2\frac{dy}{dx} + y = \cos 3x$

```
from sympy import *
x=symbols('x')
y=Function('y')(x)
DE=Eq(Derivative(y,x,2) - 2*Derivative(y,x) +y,  cos(3*x))
GS=dsolve(DE)
display(GS)
```

Output:-

$$y(x) = (c_1 + c_2 x)e^{x} + \frac{3\sin 3x}{50} - \frac{2\cos 3x}{25}$$

9) c) Write the Python code and Execute :: $\frac{d^2y}{dx^2} + y = \sin 2x$

```
from sympy import *
x=symbols('x')
y=Function('y')(x)
DE=Eq(Derivative(y,x,2)+y, sin (2*x))
GS=dsolve(DE)
display(GS)
```

Output:-

$$y(x) = (c_1 \sin x + c_2 \cos x) - \frac{\sin 2x}{3}$$

9) d) Write the Python code and Execute :: $\frac{d^2y}{dx^2} - 2\frac{dy}{dx} + 3y = \cos 2x$

```
from sympy import *
x=symbols('x')
y=Function('y')(x)
DE=Eq(Derivative(y,x,2)-2*Derivative(y,x)+3*y, cos(2*x))
GS=dsolve(DE)
display(GS)
```

Output:-

$$y(x) = (c_1 sin\sqrt{2}x + c_2 cos\sqrt{2}x)e^x - \frac{4sin2x}{17} - \frac{cos2x}{17}$$

10) a) Solve the PDE, xp + yq = z, where z = f(x, y)

```
from sympy . solvers .pde import pdsolve
from sympy import Function , Eq ,cot , classify_pde , pprint
from sympy .abc import x, y, a
f = Function ('f')
z = f(x, y)
zx = z.diff (x)
zy = z.diff (y)
# Solve xp+yq=z
eq = Eq(x*zx+y*zy , z)
pprint (eq)
print ("\n")
soln = pdsolve (eq ,z)
 pprint( soln )
```

Output:-

$$x\cdot\frac{\partial}{\partial x}(f(x, y)) + y\cdot\frac{\partial}{\partial y}(f(x, y)) = f(x, y)$$

$$f(x, y) = x\cdot F\left(\frac{y}{x}\right)$$

10) b) Solve the PDE, x^2p + y^2 q = (x + y)z, where p = $\frac{\partial z}{\partial x}$ and q = $\frac{\partial z}{\partial y}$

```
from sympy . solvers .pde import pdsolve
from sympy import Function , Eq ,cot , classify_pde , pprint
from sympy .abc import x, y, a
f = Function ('f')
z = f(x, y)
zx = z. diff (x)
zy = z. diff (y)
# Solve x^2p+y^2q =(x+y)z
eq=Eq(x ** 2*zx+y ** 2*zy ,(x+y)*z)
pprint (eq)
print ("\n")
soln = pdsolve (eq ,z)
pprint ( soln )
```

Output:-

$$x^2 \cdot\frac{\partial}{\partial x}(f(x, y)) + y^2\cdot\frac{\partial}{\partial y}(f(x, y)) = (x+y)\cdot f(x, y)$$

$$f(x, y) = (x-y)\cdot F\left(\frac{-x+y}{x+y}\right)$$