Objectives of Session3a_Derivatives_and_Integration:

1-To understand how to take derivatives and integrals by using library functions of python.

2-To implement Numerical Differenciation methods to find values of f'(x) for a given table of x and f(x).

3-To implement Numerical Integration methods to find values of definite integrals and their error bound by a specific method.

How to take derivative in python:

```
from sympy import* #Call Library of sympy
x = symbols('x')   #Make x a symbol
f = 2*x**2+5       #Function to take derivative

df = diff(f, x,2)    #diff(f,x,1) is used to take first derivative of f w.r.t x
#df = diff(f, x,n)    #diff(f,x,1) is used to take nth derivative of f w.r.t x
print(df)
print(float(df.subs(x,3))) #df.subs(x,1) is used to substitute value of x=1 in above taken derivative
```

```
4
4.0
```

```
# another procedure for finding  derivative
y=sin(x)-x
derivative_y=y.diff(x) #differentiate y w.r.t x
print(derivative_y)
```

```
cos(x) - 1
```

How to convert a sympy symbolic expression into numpy function to evaluate it on a point or array.

```
f1=lambdify(x,f) #Now f1 is numpy function
print(f1(1)) #f1 at 1, you can also evaluate f1 at an array.
df1=lambdify(x,df)
print(df1(1))
```

```
7
4
```

Task1

```
a)Use above two procedures to find the second derivative of f(x)=x**2 exp(-x).
b)Convert symbolic expression in part (a) into numpy function.
c)Evaluate the numpy function (obtained in part b)at a single value and at an array.
```

Coding of some Numerical differentiation formulae

Code of forward difference formula

Forward Differnce

• $f'(x_0) =\dfrac{f(x_0+h)-f(x_0)}{h}$

Backward Differnce

• $f'(x_0) =\dfrac{f(x_0)-f(x_0-h)}{h}$

```
# code of forward difference formula.

import numpy as np
from tabulate import tabulate
```

```
def forward_diff(x, y):

    # Compute the step size h
    h = x[1] - x[0]
    data=[]

    # Compute the forward difference approximation
    fdf = np.zeros_like(y)
    fdf[-1] = (y[-1] - y[-2]) / h  # use backward difference at the end point
    for i in range(len(y) - 1):
        fdf[i] = (y[i+1] - y[i]) / h
        data.append([x[i],y[i],fdf[i]])
    data.append([x[-1],y[-1],fdf[-1]])

    print(tabulate(data,headers=['x','f(x)','df(x)/dx'],tablefmt="github"))

    return

# example to run above code
x=[0.2,0.4,0.6,0.8]
y=[3,3.9,3.98,4.2]
forward_diff(x, y)
```

x	f(x)	df(x)/dx
0.2	3	4.5
0.4	3.9	0.4
0.6	3.98	1.1
0.8	4.2	1.1

Task 2: Write a code for Backward difference approximation (apply forward difference approximation on first point)

Code of three point endpoint and three point midpoint formula

Three-Point Endpoint Formula

- $f'(x_0) =\dfrac{−3f(x_0) + 4f(x_0 + h) - f (x_0 + 2h)}{2h}$$f'(x_0) =\dfrac{−3f(x_0) + 4f(x_0 + h) - f (x_0 + 2h)}{2h}$

Three-Point Midpoint Formula

- $f'(x_0) =\dfrac{f(x_0+h)-f(x_0-h)}{2h}$$f'(x_0) =\dfrac{f(x_0+h)-f(x_0-h)}{2h}$

```
# code for three point endpoint and three point midpoint formulae for finding f'(x) for an array of x and f(x).
import numpy as np

def three_point(x, y):

    # Compute the step size h
    data=[]
    h = x[1] - x[0]

    # Compute the forward difference approximation
    tp = np.zeros_like(y)
    tp[0]=(-3*y[0]+4*y[1]-y[2])/(2*h) #three point endpoint (left end) formula
    tp[-1]=(3*y[-1]-4*y[-2]+y[-3])/(2*h) #three point endpoint (right end) formula

    data.append([x[0],y[0],tp[0]])
    for i in range(1,len(y)-1):
        tp[i] = (y[i+1] - y[i-1]) / (2*h)
        data.append([x[i],y[i],tp[i]])
    data.append([x[-1],y[-1],tp[-1]])

    print(tabulate(data,headers=['x','f(x)','df(x)/dx'],tablefmt="github"))


    return


# example to run above code
x=[0.2,0.4,0.6,0.8]
y=[3,3.9,3.98,4.2]
three_point(x, y)
```

```
|   x |   f(x) |   df(x)/dx |
|-----|--------|------------|
| 0.2 |   3    |       6.55 |
| 0.4 |   3.9  |       2.45 |
| 0.6 |   3.98 |       0.75 |
| 0.8 |   4.2  |       1.45 |
```

Task 3: Make a code for five point endpoint and midpoint formulae where possible in given table.:

How to take integral in python

```
x = symbols('x')   #Make x a symbol
f = 2*x/(x**2-4)      #Function to take integrate

I_actual = float(integrate(f, (x,1,1.6)))    #integrate(f,(x,l,u)) is used to take integral of f from l to u
print(I_actual)
```

```
-0.7339691750802008
```

Numerical Integration by using Composite Trapezoidal rule

Trapezoid Rule

$$\int_a^b f(x) dx \approx \frac{h}{2}[f(a)+ \sum_{i=1}^{n-1} {f(x_i)}+f(b)]$$$$\int_a^b f(x) dx \approx \frac{h}{2}[f(a)+ \sum_{i=1}^{n-1} {f(x_i)}+f(b)]$$

```
def comp_trapezoidal_rule(f, a, b, n=1):  #n=1 indicates simple trpezoidal rule
  h = (b - a) / n
  x = [a + i*h for i in range(n+1)]
  y = [f(xi) for xi in x]
  s = sum(y[1:-1])
  ans=h/2 * (y[0] + 2*s + y[-1])
  return ans
```

```
#Example for simple and composite Trapezoidal
def f(x):
  return(2*x/(x**2-4))
strap=comp_trapezoidal_rule(f,1,1.6)
print(strap) # gives ans of simple trapezoidal rule
ctrap=comp_trapezoidal_rule(f,1,1.6,4)
print(ctrap) # gives ans of composite trapezoidal rule with n=4
```

```
-0.866666666666667
-0.7435983879717899
```

Computing Actual Error for simple and composite trapezoidal rules

```
print(I_actual-strap)
print(I_actual-ctrap)
```

```
0.13269749158646627
0.009629212891589134
```

For calculating SimpleTrapezoidal error bound

Working for Question no 3e Exercise 4.3

```
from sympy import* #Call Library of sympy
def f(x):
  return(2*x/(x**2-4))

def Error_bound_trap(f,l,u):#l is the lower limit and u is the upper limit of integral
```

```
    ddf = diff(f, x,2)    #Evaluating second derivative of f
    abs_max_ddf=max(abs(ddf.subs(x,l)),abs(ddf.subs(x,u)))
    h=u-l
    Error_bound=h**3*abs_max_ddf/12
    return(Error_bound,abs_max_ddf)
```

```
Error_bound_trap(f,1,1.6)
```

```
(0.552960000000000, 30.7200000000000)
```

Task 4: Make a code of composite simpson's 1/3rd rule (set n=2 for simple simpson and raise exception when user enters n=odd value) and run on f(x) mentioned in exercise # 4.2, Question #5c and excercise # 4.3, Question 3e

For calculating simple Simpson's 1/3rd rule error bound

```
from sympy import* #Call Library of sympy
x = symbols('x')   #Make x a symbol
f = x**4       #Define your function here in 'x'
def Error_bound_simp(f,l,u):#l is the lower limit and u is the upper limit of integral
    d4f = diff(f, x,4)    #Evaluating second derivative of f
    abs_max_ddf=max(abs(d4f.subs(x,l)),abs(d4f.subs(x,u)))
    h=(u-l)/2
    Error_bound=h**5*abs_max_ddf/90
    return(Error_bound,abs_max_ddf)
```

Task 5: Find Error bound for Exercise 4.3 Qno 7 part(a) and (b)