Mariyam B Tharian
2409115

Numerical differentiation using Python

Aim:

To learn how to numerically differentiate the functions.

Conceptual Background:

The derivative of a function of a real variable measures the sensitivity to change of the function value (output value) with respect to a change in its argument (input value). Derivatives are a fundamental tool of calculus. For example, the derivative of the position of a moving object with respect to time is the object's velocity: a measure of how quickly the position of the object changes when time advances.

The derivative of a function of a single variable at a chosen input value, when it exists, is the slope of the tangent line to the graph of the function at that point. The tangent line is the best linear approximation of the function near that input value. For this reason, the derivative is often described as the "instantaneous rate of change", the ratio of the instantaneous change in the dependent variable to that of the independent variable.

In order to numerically calculate the derivative of the given function we employ *Taylor series* expansion to derive finite-divided-difference approximation of derivatives.

The forward Taylor series expansion of function f(x) around point x_i [predicting value of f(x) at x_{i+1}].

$$f\left(x_{i+1}\right) = f\left(x_i\right) + f'\left(x_i\right)h + \frac{f''\left(x_i\right)}{2}h^2 + \dots \qquad \text{--------------(1)}$$

where $h = x_{i+1} - x_i$

Which can be solved for

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{h} - \frac{f''(x_i)}{2}h + O(h^2) \qquad \text{---------------(2)}$$

where $O(h^2)$represents the remaining terms of the series.

To get the expression for $f''(x_i)$, we expand the function f(x) (for point x_{i+2}) around point x_i.

$$f(x_{i+2}) = f(x_i) + f'(x_i)2h + \frac{f''(x_i)}{2}(2h)^2 + \ldots \qquad \text{---------------(3)}$$

Now equation-1 can be multiplied by 2 and subtracted from equation-3 to give

$$f(x_{i+2}) - 2f(x_{i+1}) =- f(x_i) + f''(x_i)h^2 + \ldots \qquad \text{---------------(4)}$$

Which can be solved for $f''(x_i) = \frac{f(x_{i+2}) - 2f(x_{i+1}) + f(x_i)}{h^2} + \Omega(h)$ ---------------(5)

This is inserted in equation-2 to get

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{h} - \frac{f(x_{i+2}) - 2f(x_{i+1}) + f(x_i)}{h^2}h + \zeta(h^2) \quad \text{----------------(6)}$$

Rearranging the terms in the above equation,

$$f'(x_i) = \frac{-f(x_{i+2}) + 4f(x_{i+1}) - 3f(x_i)}{2h} + \psi(h^2) \text{----------------(7)}$$

The first term in equation 7 can be used as first derivative of f(x) at point x_i with error $\psi(h^2)$. For small h, $\psi(h^2)$ term can be ignored. Therefore we can write

$$f'(x_i) = \frac{-f(x_{i+2}) + 4f(x_{i+1}) - 3f(x_i)}{2h}$$

Example:

Calculate the derivative of function $f(x) = x^2$ in the range x=0 to x=50,

Algorithm:

1. Start
2. Define and declare the function f(x)=x^2 over a period 0 to 50.
3. Take range_min=1, range_max=50 and n=500. [range_min and range_max are limits of integration. n is number of intervals]
4. Step size: h=(b – a)/n.
5. Initialize: i=0.
6. derv(i)=$(-f(x_{i+2})+ 4f(x_{i+1}) - 3f(x_i))/2h$
7. i=i+1.
8. If i<=n then goto step 6.
9. Plot derv vs x.

Activities:

1. Differentiate the following any two functions and plot the graph of $f(x)\ vs\ x$. Take the number of steps as n = 5, 10, 50 and 500. Compare the results with analytical results.

   a $f(x) = \sqrt{x}e^{-x}$ (range = 0.2 to 2) b $f(x) = \frac{sin^2(\theta)}{cos(\theta)}$ (range = 0 to π)

   c $f(x) = \frac{x^2}{\sqrt{x^3+1}}$ (range = 0.2 to 2)

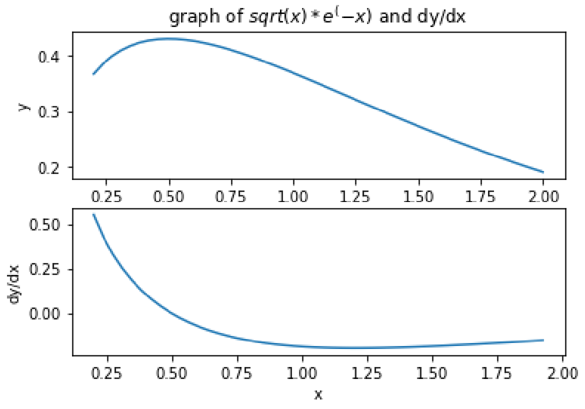
Observation table:

Sr. No.	No. of Steps (n)					Analytical Answer ($\frac{df(x)}{dx}$ at the endpoint)
	5	10	15	20	50	
1a	y'(x)= -0.2524	y'(x) =-0.1957	y'(x)=-0.1677	y'(x) = -0.1677	y'(x)= -0.1528	
1b	y'= -3.8987962 04816896e+ 16	1.7828	1.0115	0.7147	0.2627	
1c	0.7251	0.5391	0.4976	0.4793	0.4497	0.444

1. $f(x) = \sqrt{x}e^{-x}$ (range = 0.2 to 2)

Python code:

```
import numpy as np
import matplotlib.pyplot as plt
def functn(xf):
    return np.sqrt(xf)*np.exp(-xf)
a=0.2
b=2
n= 50
```

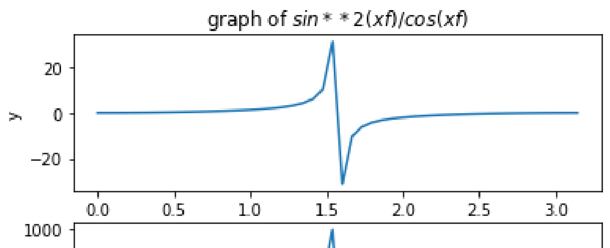


```
xi = np.linspace(a,b,n)
h=(b-a)/n
fn = functn(xi)
x=[]
derv=[]
i=0
while i<n-2:
    f=(-fn[i+2]+4*fn[i+1]-3*fn[i])/(2*h)
    x.append(xi[i])
    derv.append(f)
    i+=1

print("value of y'(x) is", round(derv[i-1],4),"at
x=",round(x[i-1],4))
plt.figure()
plt.subplot(2,1,1)
plt.plot(xi,fn)
plt.title('graph of $sqrt(x)*e^(-x)$ and dy/dx')
plt.xlabel('x')
plt.ylabel('y')
plt.subplot(2,1,2)
plt.plot(x,derv)
plt.xlabel('x')
plt.ylabel("dy/dx")
plt.show()
```

2. $f(x) = \frac{sin^2(\theta)}{cos(\theta)}$ (range = 0 to π)

Python code

```
import numpy as np
import matplotlib.pyplot as plt
def functn(xf):
    return np.sin(xf)**2/np.cos(xf)
a=0
b=np.pi
n= 50
```

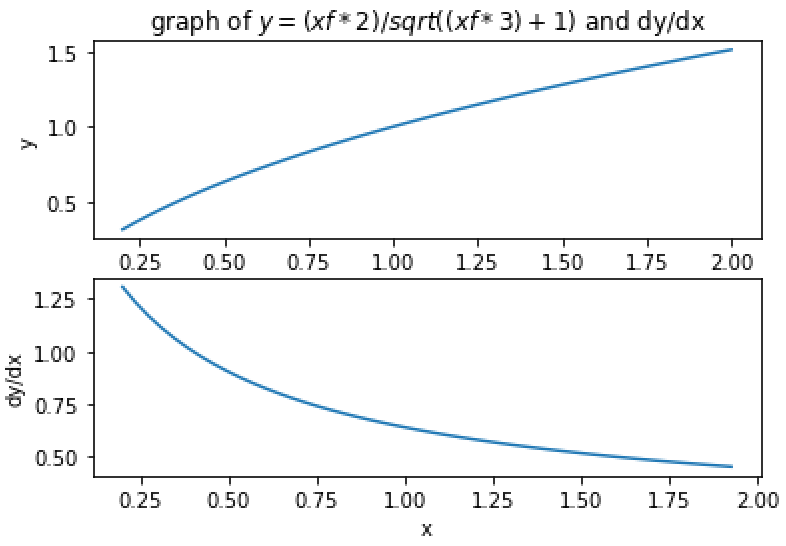


```
xi = np.linspace(a,b,n)
h=(b-a)/n
fn = functn(xi)
x=[]
derv=[]
i=0
while i<n-2:
    f=(-fn[i+2]+4*fn[i+1]-3*fn[i])/(2*h)
    x.append(xi[i])
    derv.append(f)
    i+=1

print("value    of    y'(x)    is",    round(derv[i-1],4),"at
x=",round(x[i-1],4))
plt.figure()
plt.subplot(2,1,1)
plt.plot(xi,fn)
plt.title('graph of $sin**2(xf)/cos(xf)$')
plt.xlabel('x')
plt.ylabel('y')
plt.subplot(2,1,2)
plt.plot(x,derv)
plt.xlabel('x')
plt.ylabel("dy/dx")
plt.show()
```

3. $f(x) = \frac{x^2}{\sqrt{x^3+1}}$ (range = 0.2 to 2)

```
import numpy as np
import matplotlib.pyplot as plt
```

```
def functn(xf):
    return (xf*2)/np.sqrt((xf*3)+1)
a=0.2
b=2
n= 50
xi = np.linspace(a,b,n)
h=(b-a)/n
fn = functn(xi)
x=[]
derv=[]
i=0
while i<n-2:
    f=(-fn[i+2]+4*fn[i+1]-3*fn[i])/(2*h)
    x.append(xi[i])
    derv.append(f)
    i+=1

print("value of y'(x) is", round(derv[i-1],4),"at
x=",round(x[i-1],4))
plt.figure()
plt.subplot(2,1,1)
plt.plot(xi,fn)
plt.title('graph of $y=(xf*2)/sqrt((xf*3)+1)$and dy/dx')
plt.xlabel('x')
plt.ylabel('y')
plt.subplot(2,1,2)
plt.plot(x,derv)
plt.xlabel('x')
plt.ylabel("dy/dx")
plt.show()
```



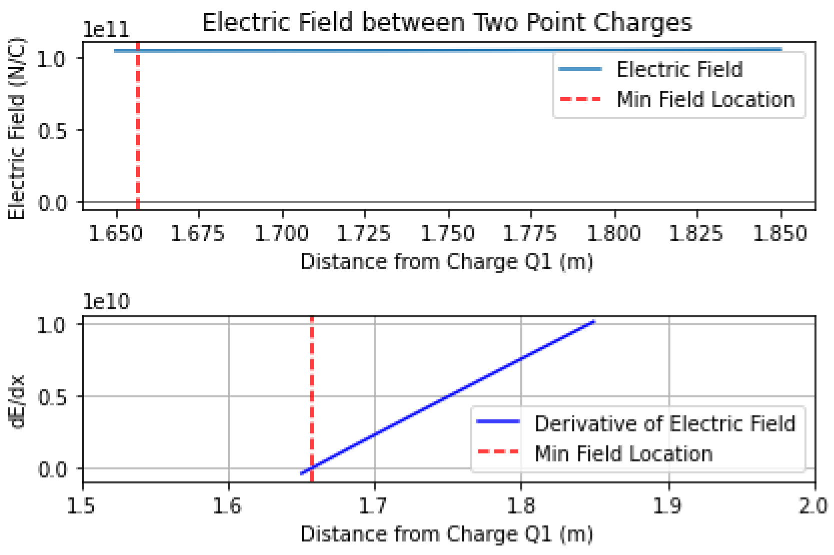
---

2. **Home Activity:** Consider two-point charges Q_1=4C & Q_2=2*Q_1 separated by distance d=4m. Using a numerical differentiation method, find the location of the minimum electric field between the charges.

```
import numpy as np

import matplotlib.pyplot as plt


# Constants

epsilon_0 = 8.854 * 10**-12

k = 1 / (4 * np.pi * epsilon_0)  # Coulomb constant


# Charges

Q1 = 4  # in Coulombs

Q2 = 2 * Q1  # in Coulombs

d = 4  # distance between the charges in meters
```

```
# Electric field function (function of x)

def electric_field(x):

    return k * Q1/x*2 + k*Q2/(d-x)*2


# Numerical differentiation

a = 1.65  # Start slightly away from 0 to avoid division by zero

b = 1.85  # End slightly away from d to avoid division by zero

n = 1000

xi = np.linspace(a, b, n)

h = (b - a) / n

ef = electric_field(xi)

x = []

derv = []

i = 0

while i < n - 2:

    f = (-ef[i + 2] + 4 * ef[i + 1] - 3 * ef[i]) / (2 * h)

    x.append(xi[i])

    derv.append(f)

    i += 1


# Find the minimum electric field location

min_index = np.argmin(ef)

min_x = xi[min_index]

min_ef = ef[min_index]


print("The location of the minimum electric field is approximately at x
  =", round(min_x, 4), "meters.")

print("The minimum electric field value is approximately E =",
  round(min_ef, 4), "N/C.")
```

```
# Plotting
plt.figure()
plt.subplot(2, 1, 1)
plt.plot(xi, ef, label='Electric Field')
plt.title('Electric Field between Two Point Charges')
plt.xlabel('Distance from Charge Q1 (m)')
plt.ylabel('Electric Field (N/C)')
plt.axhline(0, color='black', linewidth=0.5)
plt.axvline(min_x, color='r', linestyle='--', label='Min Field Location')
plt.legend()

plt.subplot(2, 1, 2)
plt.xlim(1.5, 2)
plt.plot(x, derv, label="Derivative of Electric Field", color='b')
plt.xlabel('Distance from Charge Q1 (m)')
plt.ylabel("dE/dx")
plt.axvline(min_x, color='r', linestyle='--', label='Min Field Location')
plt.legend()
plt.grid()
plt.tight_layout()
plt.show()
```



Conclusion:

The numerical differentiation using Python demonstrated the application and accuracy of the Taylor Series in evaluating the derivatives of the functions.

---×××××××---