MATHEMATICAL PHYSICS-2

ASSIGNMENT-3

Name: Aditya Pachar

Rollno:245673989

Course: BSc Physics Hons

Date:10-04-2025

Question-1

To approximate the elementary functions

1) exp(x)
2) sin(x)
3) cos(x)
4) ln(1+x)

by a finite number of Taylor's series and discuss the truncation error. To plot the function as well as the nth partial sum of its series for various values of n on the same graph and visualize the convergence of the series.

Program Code :

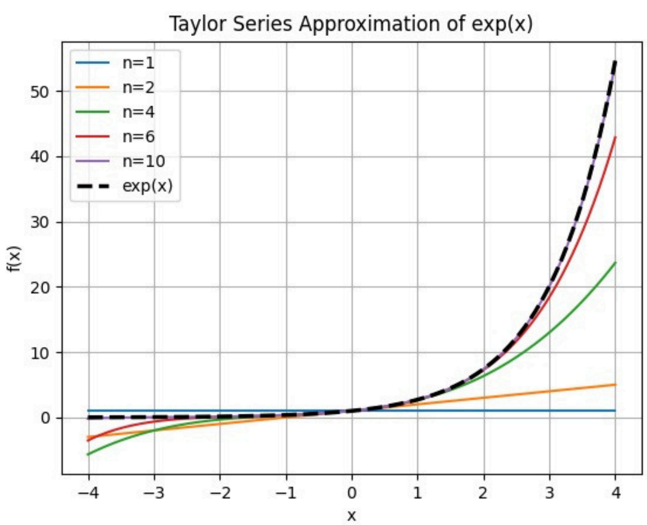
1) exp(x)

```
import numpy as np

import math

import matplotlib.pyplot as plt

def taylor_exexp(x,n):
```

```
    result=0
    for i in range(n):
        result+=x**i/math.factorial(i)
    return result

def truncation_error(x,a):
    error = np.exp(x) - taylor_exexp(x,a+1)
    return error

n=int(input("Upto which term of taylor series do you wish the function e^x to be approximated : "))
x=float(input("For which value of x do you want the function e^x to be estimated : "))
a=int(input("Upto which order do you wish to calculate the truncation error : "))



print(f'The Taylor expansion of e^x upto {n} terms of Taylor series for x={x} is {taylor_exexp(x,n):.8f} .')
print("")
print(f'The {a} order Truncation error is {truncation_error(x,a):.8f} .')



x_values=np.linspace(-4,4,100)
n_values=[1,2,4,6,10]
y_actual=np.exp(x_values)
for n in n_values:
```

```
    y_approx = [taylor_exexp(x, n) for x in x_values]
    plt.plot(x_values, y_approx, label=f'n={n}')

plt.plot(x_values, y_actual, 'k--', lw=2.5, label='exp(x)')
plt.xlabel('x')
plt.ylabel('f(x)')
plt.title("Taylor Series Approximation of exp(x)")
plt.legend()
plt.grid()
plt.show()
```

Output :



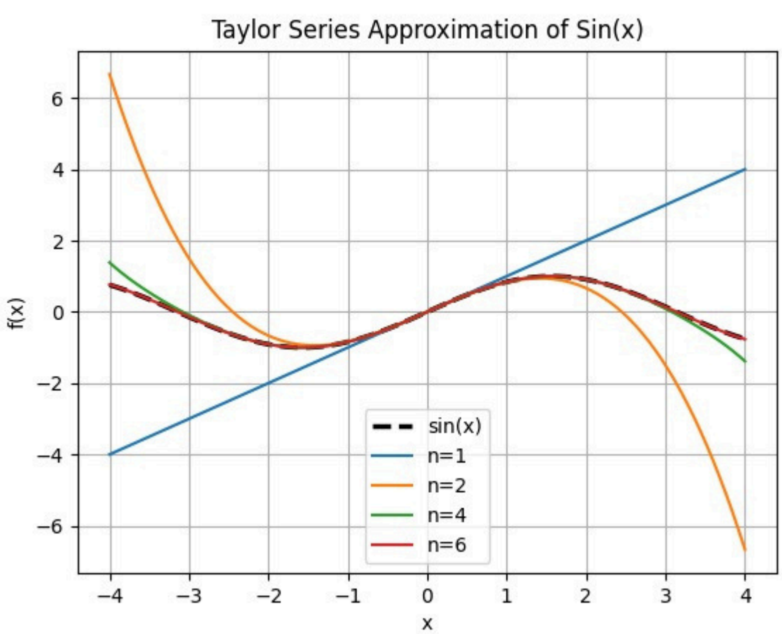

```
Upto which term of taylor series do you wish the function e^x to be approximated : 6
For which value of x do you want the function e^x to be estimated : 1
Upto which order do you wish to calculate the truncation error : 4
The Taylor expansion of e^x upto 6 terms of Taylor series for x=1.0 is 2.71666667 .

The 4 order Truncation error is 0.00994850 .
```

2)sin(x)

```
import numpy as np

import math

import matplotlib.pyplot as plt

def taylor_sinexp(x, a, n):

    expansion = 0

    for k in range(n):

        term = ((-1)**k * (x - a)**(2*k + 1)) / math.factorial(2*k + 1)

        expansion += term

    return expansion

def truncation_error(x, a, b):

    error = np.sin(x) - taylor_sinexp(x, a, b)

    return error

n = int(input("Up to which term of the Taylor series do you want to
approximate sin(x)? "))

x = float(input("For which value of x do you want to estimate sin(x)? "))

a =0

b = int(input("Up to which order do you wish to calculate the truncation
error? "))
print(f'The Taylor expansion of sin(x) up to {n} terms for x={x} is
{taylor_sinexp(x, a, n):.8f}.')
print("")

print(f'The {b}th order truncation error is {truncation_error(x, a, b):.8f}.')

x_values = np.linspace(-4, 4, 100)
```

```
n_values = [1, 2, 4, 6]

y_actual = np.sin(x_values)

plt.plot(x_values, y_actual, 'k--', lw= 2.5,label='sin(x)')

for n in n_values:

    y_approx = [taylor_sinexp(x, a, n) for x in x_values]

    plt.plot(x_values, y_approx, label=f'n={n}')


plt.xlabel('x')

plt.ylabel('f(x)')

plt.title("Taylor Series Approximation of Sin(x)")

plt.legend()

plt.grid()

plt.show()
```

Output :



```
Up to which term of the Taylor series do you want to approximate sin(x)? 6
For which value of x do you want to estimate sin(x)? 1
Up to which order do you wish to calculate the truncation error? 4
The Taylor expansion of sin(x) up to 6 terms for x=1.0 is 0.84147098.

The 4th order truncation error is 0.00000273.
```

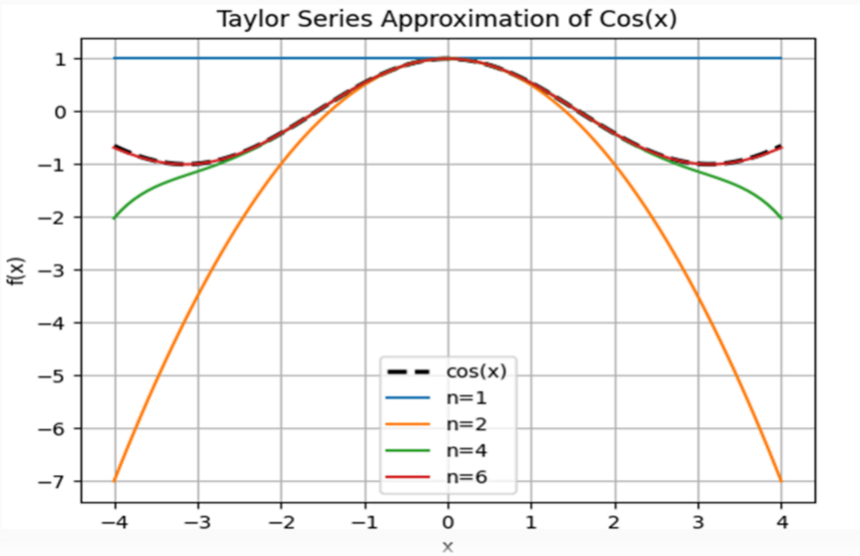
3)cos(x)

```
import numpy as np

import math

import matplotlib.pyplot as plt


def taylor_cosexp(x, a, n):

    expansion = 0

    for k in range(n):

        term = ((-1)**k * (x - a)**(2*k)) / math.factorial(2*k)

        expansion += term

    return expansion

def truncation_error(x, a, b):

    error = np.sin(x) - taylor_cosexp(x, a, b)

    return error

n = int(input("Up to which term of the Taylor series do you want to
approximate cos(x)? "))

x = float(input("For which value of x do you want to estimate cos(x)? "))

a =0
```

```
b = int(input("Up to which order do you wish to calculate the truncation error? "))

print(f'The Taylor expansion of cos(x) up to {n} terms for x={x} is {taylor_cosexp(x, a, n):.8f}.')
print("")

print(f'The {b}th order truncation error is {truncation_error(x, a, b):.8f}.')

x_values = np.linspace(-4, 4, 100)

n_values = [1, 2, 4, 6]

y_actual = np.cos(x_values)

plt.plot(x_values, y_actual, 'k--',lw=2.5, label='cos(x)')

for n in n_values:

    y_approx = [taylor_cosexp(x, a, n) for x in x_values]

    plt.plot(x_values, y_approx, label=f'n={n}')


plt.xlabel('x')

plt.ylabel('f(x)')

plt.title("Taylor Series Approximation of Cos(x)")

plt.legend()

plt.grid()

plt.show()
```

Output :

```
Up to which term of the Taylor series do you want to approximate cos(x)? 6
For which value of x do you want to estimate cos(x)? 1
Up to which order do you wish to calculate the truncation error? 4
The Taylor expansion of cos(x) up to 6 terms for x=1.0 is 0.54030230.

The 4th order truncation error is 0.30119321.
```



4)ln(1+x) OR log(1+x)

```
import numpy as np

import matplotlib.pyplot as plt


def taylor_log1p(x, a, n):

    if 1 + a <= 0:

        raise ValueError("The function log(1+a) is undefined for a ≤ -1.")


    expansion = np.log(1+a)

    for k in range(1, n+1):

        term = ((-1) ** (k+1) * (x - a) ** k) / (k * (1 + a) ** k)

        expansion += term

    return expansion


def truncation_error(x, a, n):
```

```
    if x <= -1:
        raise ValueError("log(1+x) is undefined for x ≤ -1.")
    return np.log(1+x) - taylor_log1p(x, a, n)


n = int(input("Up to which term of the Taylor series do you want to
approximate log(1+x)? "))
x = float(input("For which value of x do you want to estimate log(1+x)?
"))
a = float(input("Around which point a do you want to expand the Taylor
series? "))



print(f'The Taylor expansion of log(1+x) up to {n} terms for x={x} is
{taylor_log1p(x, a, n):.8f}.')
print("")
print(f'The {n}th order truncation error is {truncation_error(x, a, n):.8f}.')


x_values = np.linspace(-0.9, 5, 100)
n_values = [1, 2, 4, 6, 10]
y_actual = np.log(1+(x_values))


plt.plot(x_values, y_actual, 'k--',lw=2.5, label='log(1+x)')


for n in n_values:
    y_approx = [taylor_log1p(x, x+1, n) for x in x_values]
    plt.plot(x_values, y_approx, label=f'n={n}')
```

```
plt.xlabel('x')

plt.ylabel('f(x)')

plt.title("Taylor Series Approximation of Log(1+x)")

plt.legend()

plt.grid()

plt.show()


print(f"Actual log(1+{x}): {np.log1p(x):.8f}")
```

output :

```
Up to which term of the Taylor series do you want to approximate log(1+x)? 6
For which value of x do you want to estimate log(1+x)? 1
Around which point a do you want to expand the Taylor series? 0
The Taylor expansion of log(1+x) up to 6 terms for x=1.0 is 0.61666667.

The 6th order truncation error is 0.07648051.
```