Taylor Series Expansion
Lab Report for Assignment No. 1

College Roll No.	**:**	**2020PHY1140**
University Roll No.	**:**	**20068567043**
Name	**:**	**Preetpal Singh**
Unique Paper Code	**:**	**32221401**
Paper Title	**:**	**Mathematical physics III Lab**
Course and Semester	**:**	**B.Sc.(H) Physics Sem IV Lab**
Due Date	**:**	**January 15, 2022**
Date of Submission	**:**	**January 15, 2022**
Lab Report File Name	**:**	**2020PHY1140.pdf**
Partner's Name	**:**	
Partner's College Roll No.	**:**	

1 Theory

Taylor series[2] is used to express a real function $f(x)$ about the point x_0 as an infinite sum:

$$f(x) = f(x_0)+(x-x_0)f'(x_0)+\frac{(x-x_0)^2}{2!}f''(x_0)+\frac{(x-x_0)^3}{3!}f'''(x_0)+\frac{(x-x_0)^4}{4!}f^{(4)}(x_0)+\cdots$$

Here, we will always assume that these derivatives exist. If $x_0 = 0$, this is known as a **Maclaurin series**.

Radius of convergence[1] gives the number of values where the series converges.

$$R = \lim \left| \frac{a_n}{a_{n+1}} \right|$$

If R is the radius of convergence of the power series $\sum a_n x^n$, then the series is absolutely convergent if $|x| < R$ and is divergent if $|x| > R$.

Taylor series for a function of two variables[3]

For a function of two variables $f(x, y)$ whose first and second partials exist at the point (a, b), the 2^{nd}-degree Taylor polynomial of f for (x, y) near the point (a, b) is:

$$f(x,y) \approx Q(x,y) = f(a,b)+f_x(a,b)(x-a)+f_y(a,b)(y-b)+\frac{f_{xx}(a,b)}{2}(x-a)^2+f_{xy}(a,b)(x-a)(y-b)+\frac{f_{yy}(a,b)}{2}($$

If we have already determined $L(x, y)$, we can simplify this formula as:

$$f(x,y) \approx Q(x,y) = L(x,y) + \frac{f_{xx}(a,b)}{2}(x-a)^2 + f_{xy}(a,b)(x-a)(y-b) + \frac{f_{yy}(a,b)}{2}(y-b)^2$$

Maclaurin series representation and radius of convergence

Sinx

$$\sin x = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{(2n+1)!} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - + \cdots$$

Radius of Convergence

The ratio test gives us:

$$\begin{aligned}\lim_{k\to\infty}\left|\frac{(-1)^{k+1}}{(2(k+1)+1)!}x^{2(k+1)+1}/\frac{(-1)^k}{(2k+1)!}x^{2k+1}\right| &= \lim_{k\to\infty}\frac{(2k+1)!}{(2k+3)!}|x|^2 \\ &= \lim_{k\to\infty}\frac{1}{(2k+3)(2k+2)}|x|^2 \\ &= 0\end{aligned}$$

Cosx

$$\cos x = \sum_{n=0}^{\infty}(-1)^n\frac{x^{2n}}{(2n)!} = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - + \cdots$$

Radius of Convergence

The ratio test gives :

$$\begin{aligned}\lim_{k\to\infty}\left|\frac{(-1)^{k+1}}{(2(k+1))!}x^{2(k+1)}/\frac{(-1)^k}{(2k)!}x^{2k}\right| &= \lim_{k\to\infty}\frac{(2k)!}{(2k+2)!}|x|^2 \\ &= \lim_{k\to\infty}\frac{1}{(2k+1)(2k+2)}|x|^2 \\ &= 0\end{aligned}$$

$\mathbf{e}^x$

$$e^x = \sum_{n=0}^{\infty}\frac{x^n}{n!} = 1 + x + \frac{x^2}{2!} + \cdots$$

The ratio test gives :

$$\lim_{k\to\infty}\left|\frac{x^{k+1}}{(k+1)!}/\frac{x^k}{k!}\right| = \lim_{k\to\infty}\frac{|x|}{k+1} = 0$$

Because for all cases, limit is zero for all real values of x, the radius of convergence of the expansion is the set of all real numbers.

2 Algorithms

Algorithm 1 Taylor Expansion of Sin Function

function INPUT(x, n)

▷ *Here x is vector of values for which we want to calculate, n is number of terms in expansion* ◁

Define y ← an empty array

▷ *An empty to store values of Taylor Expansion* ◁

Set sol ← 0

▷ *Initialise sol variable to zero* ◁

for all $i \in \{1, \ldots, n\}$ **do**

sol+= $\sum_{n=0}^{\infty}(-1)^n \frac{x^{2n+1}}{(2n+1)!}$

append $sol \leftarrow y$

▷ *Append Output of sol variable in y array*

return y

Algorithm 2 Taylor Expansion of Cos Function

function INPUT(x, n)

▷ *Here x is vector of values for which we want to calculate, n is number of terms in expansion* ◁

Define y ← an empty array

▷ *An empty to store values of Taylor Expansion* ◁

Set sol ← 0

▷ *Initialise sol variable to zero* ◁

for all $i \in \{1, \ldots, n\}$ **do**

sol+= $\sum_{n=0}^{\infty}(-1)^n \frac{x^{2n}}{(2n)!}$

append $sol \leftarrow y$

▷ *Append Output of sol variable in y array*

return y

3 Programming Output and Analysis

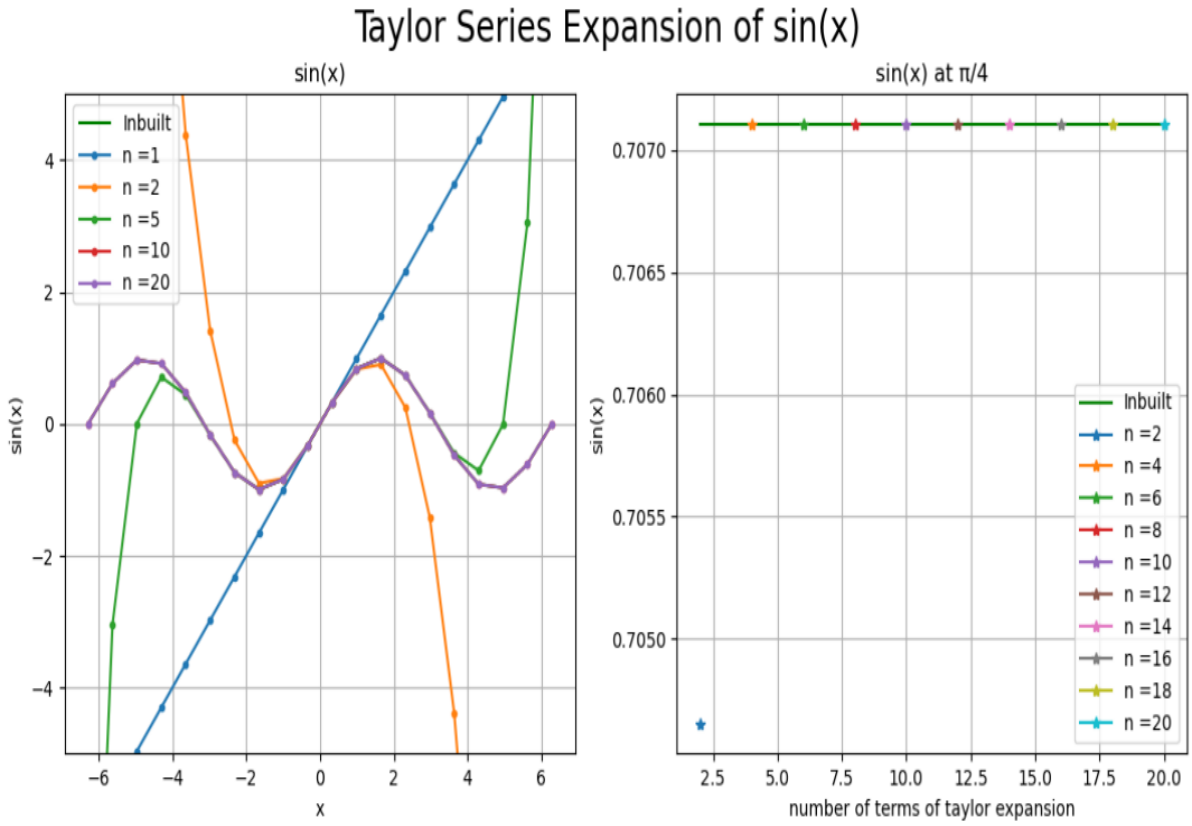

Figure 1: ***Left SubFig*** *shows the Taylor expansion of Sine Function within range of* $[-2\pi, 2\pi]$ *for* $n = [1,2,5,10,20]$ *terms. It explains that when we increase the number of terms for taylor expansion the results start converging with more efficient numpy's sine function.* ***Right SubFig*** *compares the value of* $\sin(\pi/4)$ *obtained by Taylor expansion for* $n = [1,2,5,10,20]$ *terms with original* sin *function(Green Straight Line labelled as inbuilt) In this figure we found that for* $m = 1$*, the value obtained by Taylor Series is far different from inbuilt function but when we start increasing the number of terms(n) the values start converging. The best results are obtained at* $n = 20$*. For* $n(4 \rightarrow 18)$ *values slightly deviates from original function(Deviation can be seen from Python Programme Output via Zooming)*

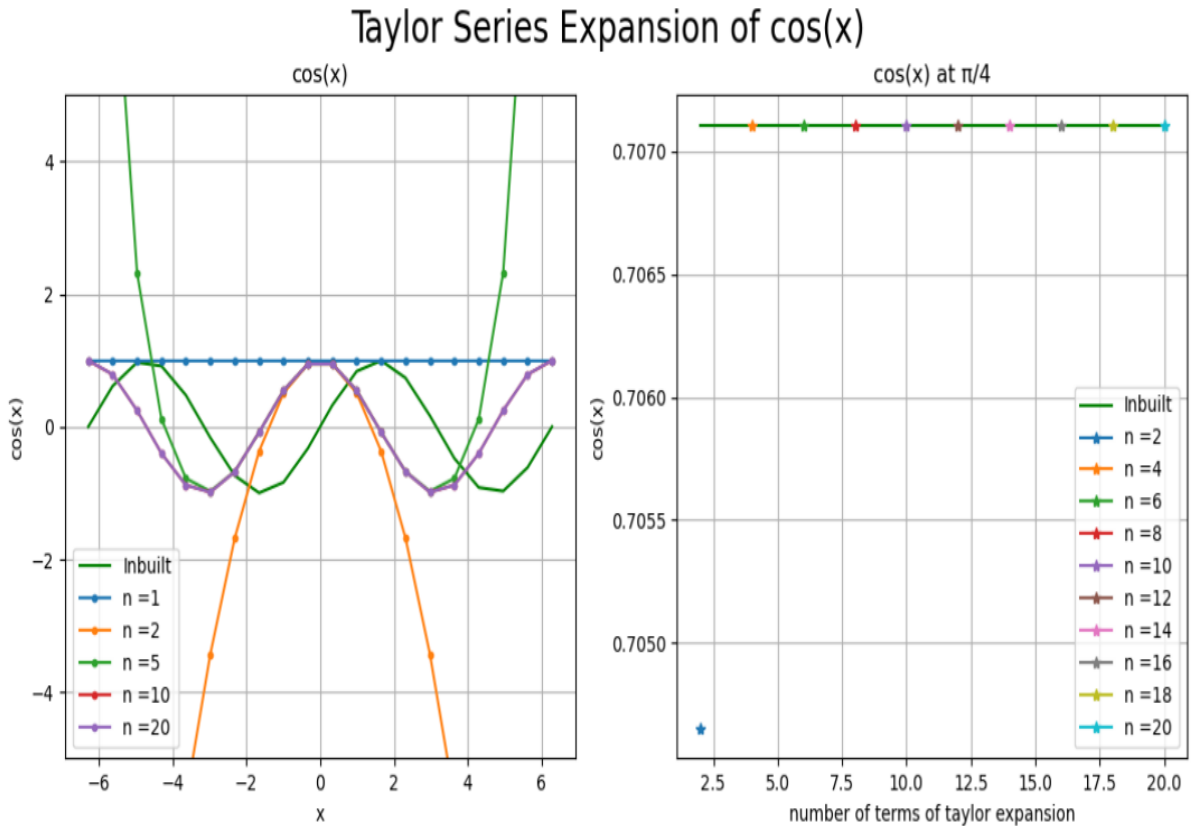

Figure 2: ***Left SubFig*** *shows the Taylor expansion of Cosine Function within range of* $[-2\pi, 2\pi]$ *for* $n = [1,2,5,10,20]$ *terms. It explains that when we increase the number of terms for taylor expansion the results start converging with more efficient numpy's Cosine function.* ***Right SubFig*** *compares the value of* $\cos(\pi/4)$ *obtained by Taylor expansion for* $n = [1,2,5,10,20]$ *terms with original* cos *function(Green Straight Line labelled as inbuilt) In this figure we found that for* $m = 1$*, the value obtained by Taylor Series is far different from inbuilt function but when we start increasing the number of terms(n) the values start converging. The best results are obtained at* $n = 20$*. For* $n(4 \rightarrow 18)$ *values slightly deviates from original function(Deviation can be seen from Python Programme Output via Zooming)*

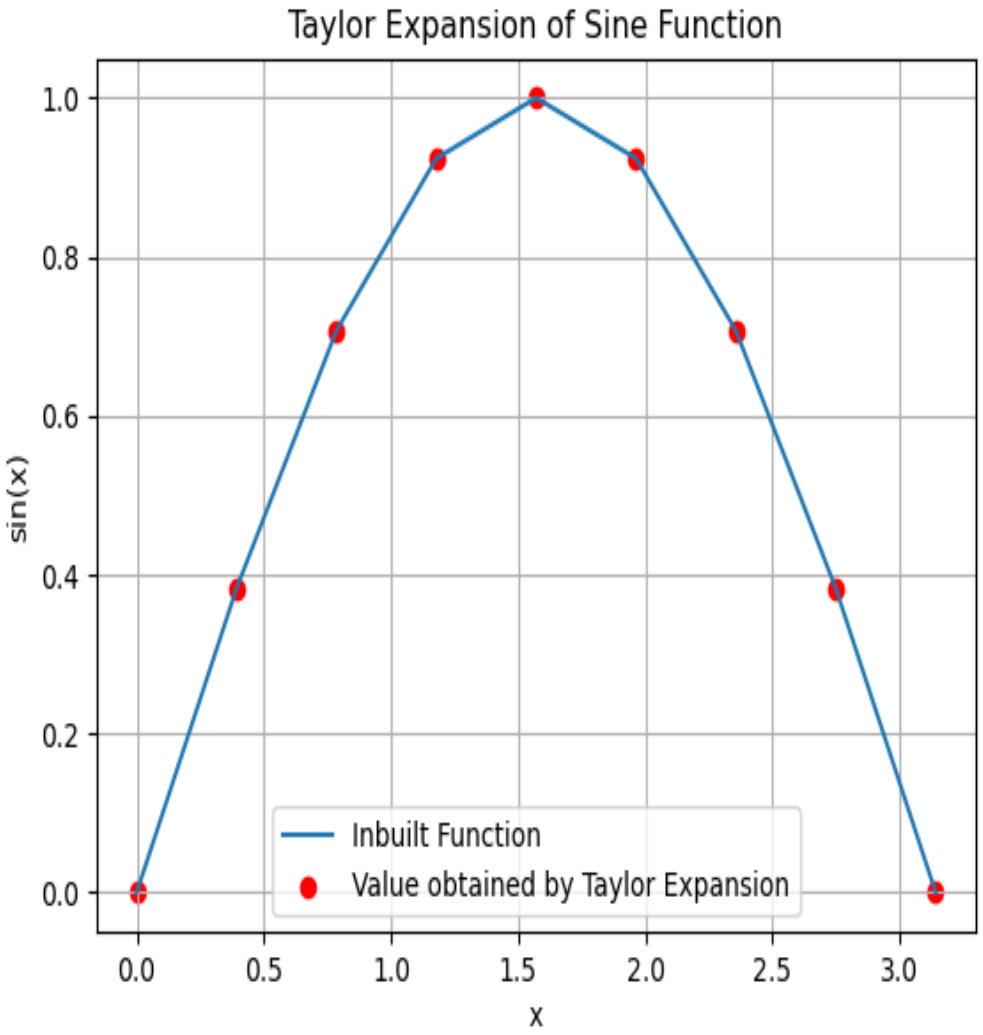

Figure 3: *Here, the red points are results obtained by taylor expansion for different number of taylor expansion terms within the tolerance of 0.000001. This graph verifies the output of taylor expansion converges with inbuilt Sine Function. As the angle increases the number of terms required for convergence also increases.*

```
Input tolerance value:0.000001
          x  sin(x) cal  n  sin_inbuilt
0  0.392699    0.382683  4     0.382683
1  0.785398    0.707107  5     0.707107
2  1.178097    0.923880  6     0.923880
3  1.570796    1.000000  7     1.000000
4  1.963495    0.923880  8     0.923880
5  2.356194    0.707107  8     0.707107
6  2.748894    0.382683  9     0.382683
```

Figure 4: *This is terminal output of main programme. This table compare the output of taylor series expansion with Inbuilt Sine Function for tolerance value = 0.000001. It clearly shows we require more terms(n) of taylor expansion when we increase the angle(x).*

4 Programmes

```
#Name Preetpal Singh
#Roll No. 2020PHY1140
import math
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
'''In MySinFunction user defined function, sin function is approximated
   with taylor expansion'''
def MySinFunction(x,n):
    y = []
    sol = 0
    for i in range(0,n,1):
        sol = sol + ((-1)**i * (x)**(2*i + 1))/math.factorial(2*i + 1)
        #y.append(sol)
    return sol
'''In MyCosFunction user defined function, cos function is approximated
   with taylor expansion'''
def MyCosFunction(x,n):
    y = []
    sol = 0
    for i in range(0,n,1):
        sol = sol + ((-1)**i * (x)**(2*i))/math.factorial(2*i)
        #y.append(sol)
    return sol
'''This user defined function take domain and tolerance as input and
   return the output within tolerance limit and number of terms of
   expansion'''
def my_fun(x,tol):
    fun_cal,n,lis = 0,0,[]
    while True:  #Loop iterates till we get output within tolerance
        fun_cal = fun_cal+(-1)**n*x**((2*n)+1)/math.factorial((2*n)+1)
        lis.append(fun_cal)
        n+=1
        if len(lis)>=2: #list should have atleast two elements to compare
            if lis[-2]==0: #if 2nd last element of list is zero then
   break
                break
            else:
                err = abs((lis[-1]-lis[-2])/lis[-2]) #Calculate relative
   error
            if err <= tol: #if relative error is within tolerance then
   break
                break
    return fun_cal,n

'''This is another function in which tolerance is taken as input from
   user and output from my_func is used to print the table that compares
    results with inbuilt function with number of terms of expansion'''
def pr_values(x_a):
```

```
    tol = float(input("Input tolerance value:"))
    fun_cal=[];n=[]
    for x in x_a:
        h=my_fun(x,tol)
        fun_cal.append(h[0]);n.append(h[1])
    return fun_cal,n

'''graph function is to automate plotting for sin and cosine function.
   This user defined function takes arguments xrange(domain of function)
   , inbuilt function(sinx,cosx) to compare, number of terms to expand
   taylor series, function(either MyCosFunction/MySinFunction), title,
   ylabel and title of subplot 1 and subplot 2 respectively'''
def graph(x_0,fun,m,req_func,title,ylabel,sub1_title,sub2_title,filename)
   :
    fig, (ax1, ax2) = plt.subplots(1,2)
    fig.suptitle(title, fontsize=20)
    ax1.plot(x_0,sin_x_0,color='green',label="Inbuilt")
    for j in m:
            apr_sin = [req_func(ang,j) for ang in x_0]
            ax1.plot(x_0,apr_sin,label="n ={}".format(j),marker='o',
   markersize=3)
    ax1.legend();ax1.set_ylim([-5,5]);ax1.set_xlabel("x");ax1.grid()
    ax1.set_ylabel("{}".format(ylabel));ax1.set_title("{}".format(
   sub1_title))
    m=np.arange(2,21,2)
    ax2.plot(m,np.sin(const_value(m)),color='green',label = "Inbuilt")

    for j in m:
            apr_sin = MySinFunction(const_value(j),j)
            ax2.plot(j,apr_sin,label="n ={}".format(j),marker='*',
   markersize=6)
    ax2.legend();ax2.set_xlabel("number of terms of taylor expansion");
   ax2.set_ylabel("{}".format(ylabel))
    ax2.grid();ax2.set_title("{}".format(sub2_title));plt.savefig("{}".
   format(filename))
    plt.show()

'''In const_value function @np.vectorize is used to return the same value
    if we iterate the function in loop'''
@np.vectorize
def const_value(m):
    return np.pi/4

if __name__ == "__main__":
    '''----------------------------------------Q1
   ----------------------------------------------------'''
    x_0=np.linspace(-2*np.pi,2*np.pi,20) #Domain to calculate
    sin_x_0 = np.sin(x_0)
    m = [1,2,5,10,20]
    graph(x_0,np.sin(x_0),m,MySinFunction,"Taylor Series Expansion of sin
   (x)","sin(x)","sin(x)","sin(x) at   /4","/home/hinton/Semester_4/MP3/
```

```
Practical/Programmes/sin.jpg")
 graph(x_0,np.cos(x_0),m,MyCosFunction,"Taylor Series Expansion of cos
(x)","cos(x)","cos(x)","cos(x) at   /4","//home/hinton/Semester_4/MP3
/Practical/Programmes/cos.jpg")

 '''----------------------------------------Q2
-----------------------------------------------'''
 x = np.arange(np.pi/8,np.pi,np.pi/8)
 sin_in = np.sin(x)
 output = pr_values(x)
 data = {"x":x,"sin(x) cal":output[0],"n":output[1], "sin_inbuilt":
sin_in}
 print(pd.DataFrame(data)) #print pandas dataframe
 plt.scatter(x,output[0],c = "r",label="Value obtained by Taylor
Expansion")
 plt.plot(x,sin_in,label="Inbuilt Function");plt.savefig("/home/hinton
/Semester_4/MP3/Practical/Programmes/final.jpg") #save the output in
image file
 plt.xlabel("x");plt.ylabel("sin(x)");plt.title("Taylor Expansion of
Sine Function");plt.grid(True);plt.legend()
 plt.show()
```

References

[1] (PDF) Advanced.Engineering.Mathematics.10th.Edition.By.ERWIN.KREYSZIG.pdf — Luis José Sa URL: https://www.academia.edu/33252267/Advanced_Engineering_Mathematics_10th_Edition_By_ERWIN_KREYSZIG_pdf (visited on 01/13/2022).

[2] Numerical Methods in Physics with Python. URL: https://www.cambridge.org/core/books/numerical-methods-in-physics-with-python/563DF013576DCC535668A100B8F7D2F9 (visited on 01/13/2022).

[3] Taylor Polynomials of Functions of Two Variables - Mathematics LibreTexts. URL: https://math.libretexts.org/Bookshelves/Calculus/Supplemental_Modules_(Calculus)/Multivariable_Calculus/3%3A_Topics_in_Partial_Derivatives/Taylor__Polynomials_of_Functions_of_Two_Variables (visited on 01/13/2022).