Mathematical Physics III

Lab Assignment #5

Ishmeet Singh

College Roll No	**:**	2020PHY1221
University Roll No	**:**	20068567051
Unique Paper Code	**:**	32221401
Paper Title	**:**	Mathematical physics III Lab
Course and Semester	**:**	B.Sc.(Hons.) Physics, Sem IV
Due Date	**:**	10 March 2022
Date of Submission	**:**	10 March 2022
Lab Report File Name	**:**	2020PHY1221_A5.pdf
Submitted to	**:**	Dr. Mamta Dahiya

Team Member :

Name	**:**	Sarthak Jain
Roll Number	**:**	2020PHY1201

Shri Guru Tegh Bahadur Khalsa College, University of Delhi
New Delhi-110007, India.

Contents

1 Theory

1.1 Hermite Gauss Quadrature method for Integration

Hermite Gauss quadrature is a Gaussian quadrature over the interval $(-\infty, \infty)$ with weighting function $W(x) = e^(- x^2)$.
The method is used for evaluating the integrals of the following kind:

$$\int_{-\infty}^{+\infty} e^{-x^2} f(x)\, dx$$

1.2 Hermite differential equation

The Hermite differential equation is given by:

$$\frac{d^2y}{dx^2} - 2x\frac{dy}{dx} + 2\lambda y = 0$$

The first five Hermite Polynomials are:

- $H_0 = 1$
- $H_1 = 2x$
- $H_2 = 4x^2 - 2$
- $H_3 = 8x^3 - 12x$
- $H_4 = 16x^4 - 48x^2 + 12$

1.3 Recurrence Relations

Recurrence Relation I:

$$H_{n+1}(x) = 2xH_n(x) - 2nH_{n-1}(x)$$

Recurrence Relation II:

$$H_n^{'}(x) = 2nH_{n-1}(x)$$

1.4 Orthogonal Properties

The orthogonal properties of Hermite Polynomials are expressed as,

$$\int_{-\infty}^{\infty} e^{-x^2} H_m(x) H_n(x)\, dx = \begin{cases} 0, & m \neq n \\ 2^n n! \sqrt{\pi}, & m = n \end{cases}$$

2 Algorithm

Algorithm 1 Algorithm for n-point Gauss Hermite Quadrature Method

```
function MyHermiteQuad($expression, n$)    ▷ Function to perform numerical
                        integration using n-point Gauss Hermite quadrature method
    x = var('x')                                                             ▷
                                                   expr = sympify(expression)
    func = lambdify(x,expr)
    xvals,weights = np.polynomial.hermite.hermgauss(n)                       ▷
    w_fx = [weights[i]*func(xvals[i])
    for i in range(len(weights)) do:                                         ▷
        result = sum(w_fx)
    end for
    return result
end function
```

3 Programming

3.1 2020PHY1221_A5.py

```python

import numpy as np
import matplotlib.pyplot as plt
import MyIntegration as mi
import pandas as pd

"Name: Ishmeet Singh, 2020PHY1221"
"Partner Name: Sarthak Jain, 2020PHY1201"

print("My Roll No.: 2020PHY1221")

def integral_simp(I1_exact,I2_exact):
    i = 2
    I1_simp = []
    I2_simp = []
    count1_simp = []
    count2_simp = []
    I1_exact_S = []
    I2_exact_S = []

    while True:
        if abs(I1_exact - mi.MySimp("exp(-x**2)/(1+x**2)",1000,-1000,i))
   <= 10**(-2):
            I1_simp.append(mi.MySimp("exp(-x**2)/(1+x**2)",1000,-1000,i))
            I1_exact_S.append(I1_exact)
            count1_simp.append(i)
            break
        else:
            I1_simp.append(mi.MySimp("exp(-x**2)/(1+x**2)",1000,-1000,i))
            I1_exact_S.append(I1_exact)
            count1_simp.append(i)
            i += 2

    i = 2

    while True:
        if abs(I2_exact - mi.MySimp("1/(1+x**2)",1000,-1000,i)) <=
   10**(-2):
            I2_simp.append(mi.MySimp("1/(1+x**2)",1000,-1000,i))
            I2_exact_S.append(I2_exact)
            count2_simp.append(i)
            break
        else:
            I2_simp.append(mi.MySimp("1/(1+x**2)",1000,-1000,i))
            I2_exact_S.append(I2_exact)
            count2_simp.append(i)
            i += 2

    return I1_simp,I2_simp,count1_simp,count2_simp,I1_exact_S,I2_exact_S

def graph(I1,I2,n,I1_simp,I2_simp,count1_simp,count2_simp,I1_exact_S,
    I2_exact_S):
    fig1,ax1 = plt.subplots(1, 2)
    fig2,ax2 = plt.subplots(1, 2)
    ax1[0].plot(n,I1,label = "MyHermiteQuad")
    ax1[0].plot(n,I1_exact_LL,label = "Analytic Value")
    ax1[1].plot(count1_simp,I1_simp,label = "MySimp")
    ax1[1].plot(count1_simp,I1_exact_S,label = "Analytic Value")
    ax2[0].plot(n,I2,label = "MyHermiteQuad")
    ax2[0].plot(n,I2_exact_LL,label = "Analytic Value")
    ax2[1].plot(count2_simp,I2_simp,label = "MySimp")
    ax2[1].plot(count2_simp,I2_exact_S,label = "Analytic Value")
    for i in range(2):
        if i == 0:
            ax1[i].set(xlabel = "Nodal Points (n)",ylabel = "Value of
    Integration (I)",title = "Gauss Hermite Quadrature")
            ax2[i].set(xlabel = "Nodal Points (n)",ylabel = "Value of
    Integration (I)",title = "Gauss Hermite Quadrature")
        elif i == 1:
            ax1[i].set(xlabel = "Nodal Points (n)",ylabel = "Value of
    Integration (I)",title = "Simpson 1/3 Method")
            ax2[i].set(xlabel = "Nodal Points (n)",ylabel = "Value of
    Integration (I)",title = "Simpson 1/3 Method")
        ax1[i].grid(ls = "--")
        ax2[i].grid(ls = "--")
        ax1[i].legend()
        ax2[i].legend()
    fig1.suptitle("INTEGRAL 1")
    fig2.suptitle("INTEGRAL 2")
    plt.show()

if __name__ == "__main__":
```

```

# PART B I

count = 0
func = []
Exact
=[1.7724538509,0,0.886226925,0,1.329340388,0,3.32335097044,0,11.63172839,0]


for count in range(len(Exact)):
    f = input("\nEnter Function: ")
    func.append(f)
    if count < (len(Exact) - 1):
        ans = input("Do you want to enter more function (Y/N) ?\t")
        if ans == "N" or ans == "n":
            break

for j,m in zip(func,Exact):
    for k in range(2,6,2):
        print("\nValue of integration of",j,"for n =",k,"is: ",mi.
MyHermiteQuad(j,k))
    print("\nExact Value of integration of",j,"is: ",m)
    print("
---------------------------------------------------------------")


# PART B II

I1 = []
I2 = []
I1_exact = 1.343293421646735
I2_exact = 3.141592653589793
I1_exact_LL = []
I2_exact_LL = []
n = []

for i in range(2,130,2):
    n.append(i)
    I1_exact_LL.append(I1_exact)
    I2_exact_LL.append(I2_exact)
    i1 = mi.MyHermiteQuad("1/(1+x**2)",i)
    I1.append(i1)
```

```
        i2 = mi.MyHermiteQuad("exp(x**2)/(1+x**2)",i)
        I2.append(i2)

    data1 =  np.column_stack([n,I1,I2])
    file1 = np.savetxt("quad-herm-1221.txt",data1,header = ("n,I1,I2"))

    df1 =  pd.DataFrame({"n": n, "I1": I1, "I2": I2})
    print("\nGAUSS HERMITE QUADRATURE:\n",df1)

    # PART B III & IV

    I1_simp,I2_simp,count1_simp,count2_simp,I1_exact_S,I2_exact_S = 
   integral_simp(I1_exact,I2_exact)

    df2 =  pd.DataFrame({"n": count1_simp, "I1": I1_simp})
    print("\nTOLERNACE LIMIT = 10**(-2)")
    print("\nSIMPSON FOR INTEGRAL 1:\n",df2)
    data2 =  np.column_stack([count1_simp,I1_simp])
    file2 = np.savetxt("Simpson-Integral_1(H)-1221.txt",data2,header = ("n
   ,I1"))

    print("\n
   -----------------------------------------------------------------")

    df3 =  pd.DataFrame({"n": count2_simp, "I2": I2_simp})
    print("\nTOLERNACE LIMIT = 10**(-2)")
    print("\nSIMPSON FOR INTEGRAL 1:\n",df3)
    data3 =  np.column_stack([count2_simp,I2_simp])
    file3 = np.savetxt("Simpson-Integral_2(H)-1221.txt",data3,header = ("n
   ,I2"))

    graph(I1,I2,n,I1_simp,I2_simp,count1_simp,count2_simp,I1_exact_S,
   I2_exact_S)
```

4 Results and Discussion

According to Part b) i. of the assignment, we have verified in our code that n-point quadrature formula gives exact result when $f(x)$ is a polynomial of order $2n-1$ taking $n=2$ and $n=4$

Also, according to Part b) ii. of the assignment, we were asked to compute numerically the integration values for two different functions, first, by using n - point Gauss Hermite Quadrature method of integration and second, by using the Simpson $\frac{1}{3}$ method of integration.

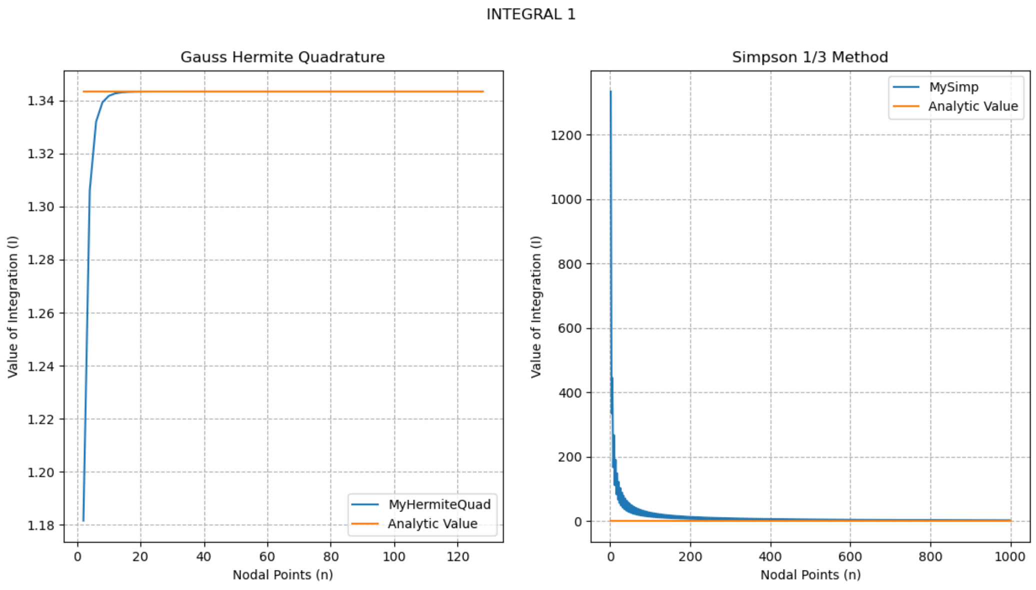

Figure 1: Comparing Gauss Hermite Quadrature with Simpson $\frac{1}{3}$ for First Integral

In the above graph (1), we have shown the comparison between the two numerical methods while simultaneously comparing each of them with the analytical values. It can be inferred from the above graph (1) that the n - point Gauss Hermite quadrature method starts approaching the analytical value for fewer nodal points (≈ 12) while for the Simpson $\frac{1}{3}$ method minimum nodal points required for the numerical value to converge with the analytical values were around ≈ 200.

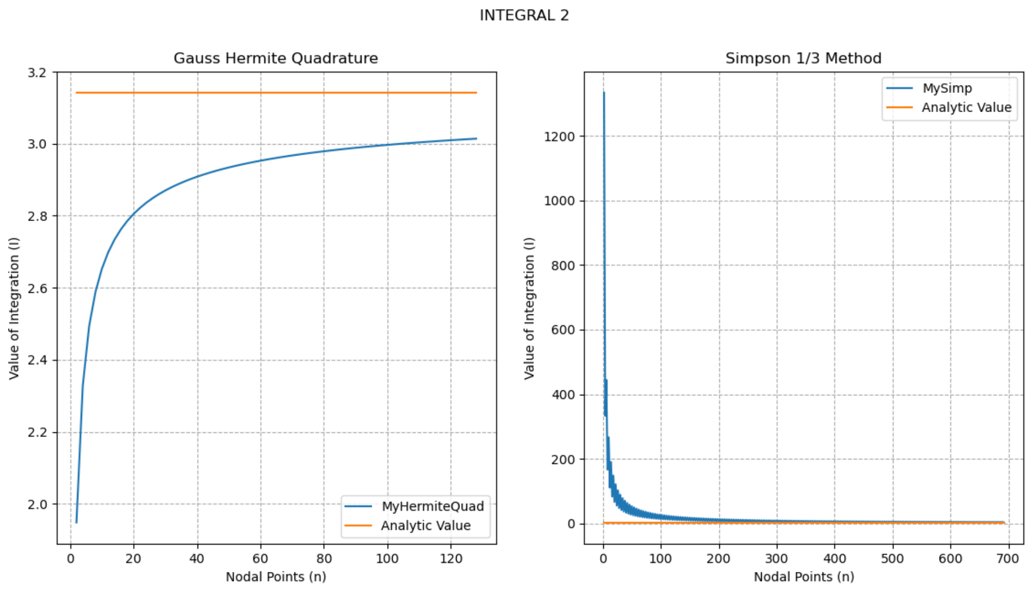

Figure 2: Comparing Gauss Hermite Quadrature with Simpson $\frac{1}{3}$ for Second Integral

For the second integral, however, it can be seen in the above graph (2) that the numerical value computed by the n - point Gauss Hermite quadrature method does not seem to approach the analytical value even for $n = 128$, whereas for the Simpson $\frac{1}{3}$ method, the numerical value does start to approach more for fewer nodal points than the n - point Gauss Hermite quadrature method.

This is purely due to the fact that for the Simpson $\frac{1}{3}$ method of integration, we chose the limits of integration as $(-1000, 1000)$, which is in no way comparable to $(-\infty, +\infty)$. Therefore, the Simpson $\frac{1}{3}$ method seems to approximate the function better than the n - point Gauss Hermite quadrature method for this particular case but in general, we believe that an accurate comparison between the two can not be made.

NOTE: As can be seen from both the graphs (1) and (2), the numerical values of integration for the Simpson method seem to oscillate for each alternate nodal point. As of now, we can't seem to figure out the actual reason behind this but we do intend to improve our computations and find a resolution to this problem.