Hermite gauss quadrature
Lab Report for Assignment No. 5

Kabir Sethi (2020PHY1097)
Pawanpreet Kaur (2020PHY1162)

S.G.T.B. Khalsa College, University of Delhi, Delhi-110007, India.

March 10, 2022

Submitted to Dr. Mamta
"32221401 - MATHEMATICAL PHYSICS III"

Contents

1 Theory

1.1 Hermite-Gauss Quadrature

As we know that Gauss Quadrature can be used for several different polynomials. One such type of polynomial is Hermite Polynomials which are known to be the solution of Hermite second order linear differential equation.

$$y" + -2xy' + \lambda y = 0 \qquad \text{where} \lambda \in \mathbb{R} \tag{1}$$

Whenever $\lambda \in \mathbb{N}$ the solution of this differential equation gives the Hermite polynomial.
The weighting function and integration interval of these polynomials is -

$$w(x) = e^{-x^2} \qquad \text{on} \qquad [-\infty, \infty) \tag{2}$$

Or we can write it as

$$\int_{-\infty}^{\infty} f(x)dx \tag{3}$$

With the weighting function e^{-x^2}.

$$\int_{-\infty}^{\infty} f(x)e^{-x^2}dx = \sum_{1}^{n} w_i f(x_i)$$

where n is the n point quadrature used.We can write

$$\int_{-\infty}^{\infty} f(x)dx = \int_{-\infty}^{\infty} f(x)e^{x^2}e^{-x^2}dx = \int_{-\infty}^{\infty} g(x)e^{-x^2}dx$$
$$g(x) := f(x)e^{x^2}$$

1.2 Hermite polynomials

The Solution of above differential equation results in the following recurrence relation.

$$2xH_n(x) = 2nH_{n-1}(x) - H_{n+1}(x) \tag{4}$$

With the knowlegde of $H_0 \text{and} H_1$ we can determine all the other polynomials

$$H_n(x) = \sum_{k=0}^{n} \frac{(-1)^k(n!)}{(n-2k)!k!}(2x)^{n-2k} \tag{5}$$

From these relations we can derive first five Hermite polynomials

$$H_0 = 1 \qquad H_1 = 2x \tag{6}$$
$$H_2 = 4x^2 - 2 \tag{7}$$
$$L_3 = 8x^3 - 12x \tag{8}$$
$$L_4 = 16x^4 + 48x^2 + 12 \tag{9}$$
$$L_5 = 32x^5 - 160x^3 + 120x \tag{10}$$

Orthogonality relation

$$\int_{-\infty}^{\infty} H_m(x)H_n(x)e^{-x^2}dx = 2^n n!\sqrt{\pi}\delta_{mn} \tag{11}$$

Where δ_{mn} is the kronecker delta function

1.3 2-point Hermite gauss quadrature

Since this formula is to have degree of precision equal to 3,Then the weights and abcissas must satisfy-that for constant,linear,quadratic and cubic functions the integral must be exatly equal to the weights multiplied by the function evaluation at these decided abcissa.

1. $f(x) = 1$

$$\int_{-\infty}^{\infty} 1e^{-x}dx = w_1f(x_1) + w_2f(x_2)$$

$$w_1 + w_2 = \sqrt{\pi} \tag{12}$$

2. $f(x) = x$

$$\int_{-\infty}^{\infty} xe^{-x^2}dx = w_1f(x_1) + w_2f(x_2) \tag{13}$$

$$w_1x_1 + w_2x_2 = [\frac{-e^{-x^2}}{2}]_{-\infty}^{\infty} \tag{14}$$

$$w_1x_1 + w_2x_2 = 0 \tag{15}$$

3. $f(x) = x^2$

$$\int_{-\infty}^{\infty} x^2e^{-x^2}dx = w_1f(x_1) + w_2f(x_2) \tag{16}$$

$$\tag{17}$$

$$w_1x_1^2 + w_2x_2^2 = \frac{\sqrt{\pi}}{2} \tag{18}$$

4. $f(x) = x^3$

$$\int_{-\infty}^{\infty} x^3e^{-x^2}dx = w_1f(x_1) + w_2f(x_2) \tag{19}$$

$$w_1x_1^3 + w_2x_2^3 = [-(x^2+1)e^{-x^2}]_{-\infty}^{\infty} \tag{20}$$

$$w_1x_1^3 + w_2x_2^3 = 0 \tag{21}$$

After solving the following equations we get the following weights-

x_i	w_i
-0.707106781186547524400 8	0.886226925452758013649 1
0.707106781186547524400 8	0.886226925452758013649

Table 1: Roots and weights for 2-point formula

2 Algorithm

Algorithm 1 MyHermiteQuad

function MYHERMITEQUAD:(f, n) MyHermiteQuad function takes the parameter f and n f: function n: no.of points

Integral=0 ▷ Declaring variable to store the value of integral

xi, wi=xi,wi=np.polynomial.hermite.hermgauss(n) ▷ Inbuilt function xi,wi=np.polynomial.hermite.hermgauss(n) takes n as a parameter and it returns two arrays of weights and points.

for all (Xi, Wi) in zip (xi, wi): **do**

Integral+=$Wi * f(Xi)$ ▷ n-point gauss-hermite quadrature Integration formula is the sum of the product of weight and points

return Integral ▷ Returns the value of integral

3 Discussion

```
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*

METHOD USED : Gauss Hermite quadrature (TWO POINT)
   f(x)  Calculated     Exact
0      1    1.772454  1.772454
1      x    0.000000  0.000000
2   x**2    0.886227  0.886227
3   x**3    0.000000  0.000000
4   x**4    0.443113  1.329340
5   x**5    0.000000  0.000000

*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
```

Figure 1: *Two-Point Gauss Hermite Quadrature for different functions.*

From the above table , it can be understood that two point quadrature gives exact results for the polynomials of degree 3 and less but for the polynomials of degree 4 or more , it does not give exact results.

```
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*

METHOD USED : Gauss Hermite quadrature (FOUR POINT)
   f(x)     Calculated       Exact
0      1  1.772454e+00   1.772454
1      x  2.775558e-17   0.000000
2   x**2  8.862269e-01   0.886227
3   x**3  0.000000e+00   0.000000
4   x**4  1.329340e+00   1.329340
5   x**5  1.110223e-16   0.000000
6   x**6  3.323351e+00   3.323351
7   x**7  0.000000e+00   0.000000
8   x**8  8.973048e+00  11.631728
9   x**9  0.000000e+00   0.000000

*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
```

Figure 2: *Four-Point Gauss Hermite Quadrature for different functions.*

From the above table , it can be understood that four point quadrature gives exact results for the polynomials of degree 7 and less but for the polynomials of degree 8 or more , it does not give exact results.

So, it can be concluded that npoint quadrature formula gives exact result when $f(x)$ is a polynomial of order 2n 1 or less.

#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#

	n	I1	I2
0	2	1.181636	1.948188
1	4	1.306019	2.328295
2	8	1.339187	2.588464
3	16	1.343129	2.761919
4	32	1.343292	2.879063
5	64	1.343293	2.958943
6	128	1.343293	3.013879

#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#

Figure 3: *Gauss Hermite Quadrature for different values of n for integral I1 and I2.*

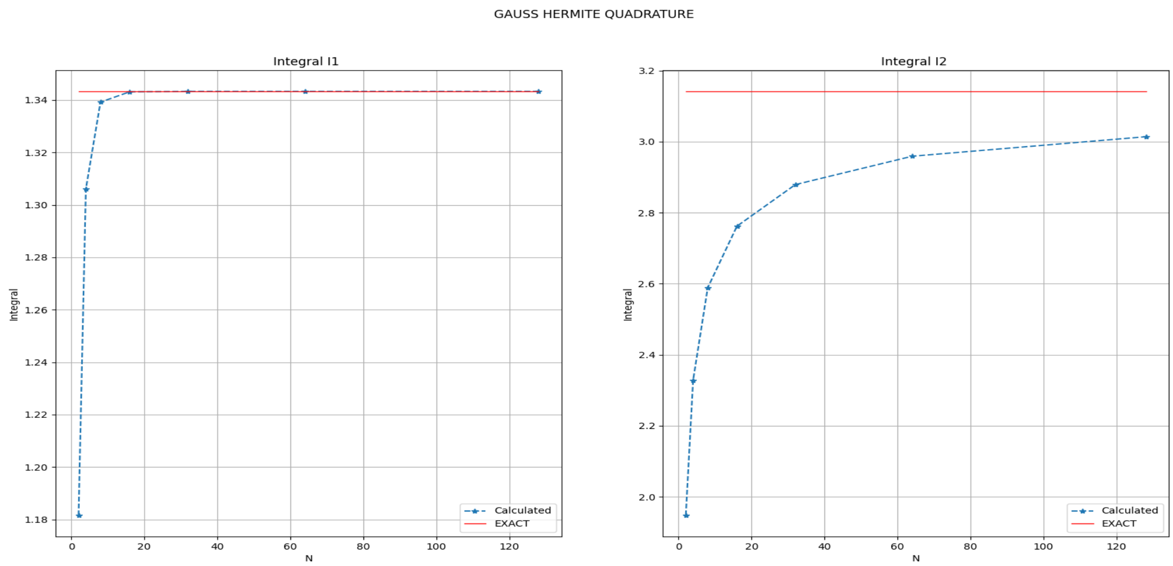

Figure 4: *Integral vs n plot for Gauss Hermite Quadrature.*

From the graph we can see that the integral value I1 approaches to true value very rapidly i.e it converges for increasing value of n. The graph overlap over the exact value line for n=20.
For integral value I2 graph we can see that the graph approaching towards true value as n increases but it doesn't overlap over the exact value line and the integral I2 graph line shows exponential growth due to presence of exponential term in the integrand.

```
RESULTS USING SIMPSON METHOD
Tolerance for MYSimp defined in MyIntegration Module = 0.1e-5
Tolerance for the value of Integral with respect to value of b(upper limit) = 0.1e-8

INTEGRAL I1

Limit -R to R
   a(lower limit)  b(upper limit)  Integral I1
0             -10              10     1.343293
1            -100             100     1.343293
INTEGRAL I2

Limit -R to R
   a(lower limit)  b(upper limit)  Integral I2
0             -10              10     2.942255
1            -100             100     3.121593
2           -1000            1000     3.139593
3          -10000           10000     3.141393
4         -100000          100000     3.141573
5        -1000000         1000000     3.141591
```

Figure 5: *Calculation of Integral I1 and I2 using Simpson Method*

It can be seen that required tolerance is achieved for R=1000000

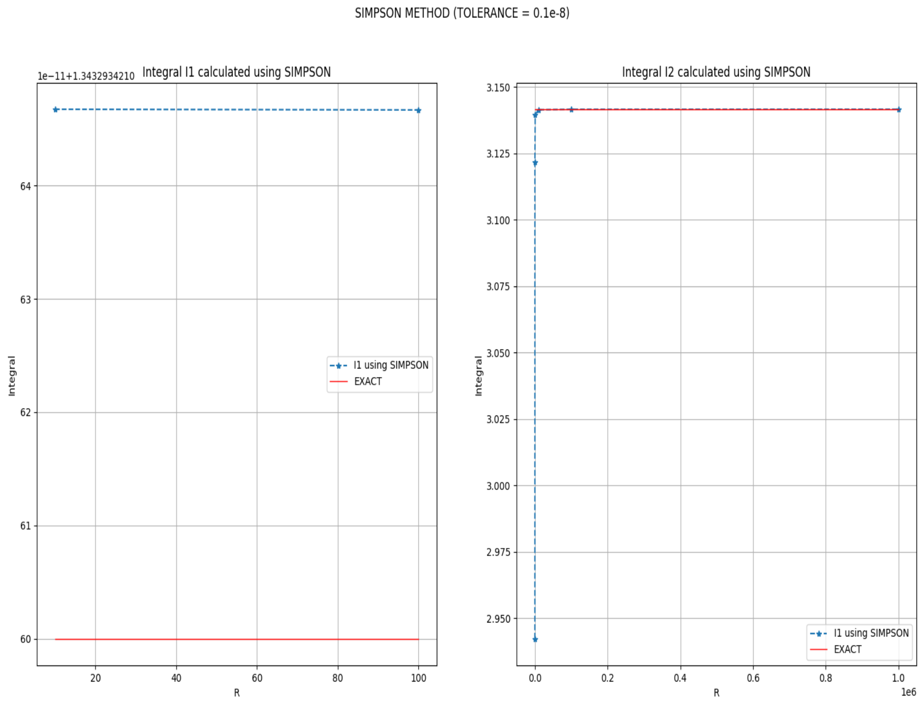

Figure 6: *Integral vs R plot for Integral I1 and I2 calculated using Simpson Method.*

It is the graph between integral value vs the limits that we are using instead of infinity to check for a given tolerance what is the limit that gives us a value correct upto certain significant digits.
For I1 we can see that the integral starts converging for R=1000 . And I2 converges for the value of R=0.1*10

4 Program

```python
'''
Name-Kabir Sethi
Roll No. - 2020PHY1097


Partner -
Name-Pawanpreet Kaur
Roll No. - 2020PHY1092

'''

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import math
from scipy import integrate
from sympy import *
from sympy import simplify
import scipy
from MyIntegration import MySimp
from MyIntegration import MyHermiteQuad

print("Name-Kabir Sethi  \n Roll No. - 2020PHY1097")
#(c)

def new_simp(f,a,R0,R_max,tol):
    lis=[]
    R_a=[]
    w=0
    a_a=[]
    while R0<=R_max:
        j=MySimp(f,-R0,R0,2,key1=True,N_max=10**8,key2=True,tol=0.1e-5)
        lis.append(j[0])
        R_a.append(R0)
        a_a.append(-R0)
        if len(lis)>=2:
            if lis[-1]<=0.1e-5:
                err=abs(lis[-1]-lis[-2])
            else:
                err=abs((lis[-1]-lis[-2])/lis[-1])
            if err<=tol:
                w=1
                break
            else:
                pass
        R0=10*R0
    if w==0:
            s=("R_max reached without achieving required tolerance")
    elif w==1:
             s="Given tolerance achieved with R=",R_a[-1]
    return lis[-1],R_a[-1],s,lis,R_a,a_a     #returning integral,number of intervals and message


#Q3(b)
#(i)
n=2
f_x=["1","x","x**2","x**3","x**4","x**5"]
Calc=[]
Exact=[1.7724538509,0,0.88622692545,0,1.329340388,0]
```

```
for i in range(0,6):
    print(i+1,"th function")
    f=eval("lambda x:"+input("Enter the value of the FUNCTION F(x): "))
    Calc.append(MyHermiteQuad(f, n))


data={"f(x)":f_x,"Calculated":Calc,"Exact":Exact}
print()
print("*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*")
print()
print("METHOD USED : Gauss Hermite quadrature (TWO POINT)")
print(pd.DataFrame(data))
print()
print("*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*")
print()


n=4
f_x=["1","x","x**2","x**3","x**4","x**5","x**6","x**7","x**8","x**9"]
Calc=[]
Exact=[1.7724538509,0,0.886226925,0,1.329340388,0,3.32335097044,0,11.63172839,0]
for i in range(0,10):
    print(i+1,"th function")
    f=eval("lambda x:"+input("Enter the value of the FUNCTION F(x): "))
    Calc.append(MyHermiteQuad(f, n))

data={"f(x)":f_x,"Calculated":Calc,"Exact":Exact}
print()
print("*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*")
print()
print("METHOD USED : Gauss Hermite quadrature (FOUR POINT)")
print(pd.DataFrame(data))
print()
print("*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*")
print()






#(ii)
I1_a=[]
I2_a=[]
n_a=[2,4,8,16,32,64,128]
f1=lambda x : 1/(1+x**2)
f2=lambda x : np.exp(x**2)/(1+x**2)
for n in n_a:
    I1_a.append(MyHermiteQuad(f1, n))
    I2_a.append(MyHermiteQuad(f2, n))

DataOut = np.column_stack((n_a,I1_a,I2_a))
np.savetxt("quad-herm-1092", DataOut,delimiter=',')

print("
    #-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#
    ")
print()
data={"n":n_a,"I1":I1_a,"I2":I2_a}
print(pd.DataFrame(data))
```

```
q1=np.array([1.3432934216]*len(I1_a))
q2=np.array([np.pi]*len(I2_a))
fig, (ax1, ax2) = plt.subplots(1, 2)
fig.suptitle('GAUSS HERMITE QUADRATURE')
ax1.plot(n_a,I1_a,marker="*",label="Calculated",linestyle='dashed')
ax1.plot(n_a,q1,label="EXACT",c="red",linewidth=1)
ax2.plot(n_a,I2_a,marker="*",label="Calculated",linestyle='dashed')
ax2.plot(n_a,q2,label="EXACT",c="red",linewidth=1)
ax1.grid()
ax1.legend()
ax1.set(xlabel="N",ylabel="Integral",title="Integral I1 ")
ax2.set(xlabel="N",ylabel="Integral",title="Integral I2 ")
ax2.grid()
ax2.legend()
plt.show()


a=0
R0=10
R_max=10**6
tol=0.1e-8
F1=lambda x : np.exp(-1*x**2)/(1+x**2)
F2=lambda x : 1/(1+x**2)


s2=new_simp(F1,-R0,R0,R_max,tol)
s3=new_simp(F2,-R0,R0,R_max,tol)

print()
print("
    #-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#
    ")
print()
print("RESULTS USING SIMPSON METHOD")
print("Tolerance for MYSimp defined in MyIntegration Module = 0.1e-5")
print("Tolerance for the value of Integral with respect to value of b(upper limit) =
    0.1e-8")
print()
print("INTEGRAL I1")
print()
print("Limit -R to R")
data={"a(lower limit)":s2[5],"b(upper limit)":s2[4],"Integral I1":s2[3]}
print(pd.DataFrame(data))


print("INTEGRAL I2")
print()
print("Limit -R to R")
data={"a(lower limit)":s3[5],"b(upper limit)":s3[4],"Integral I2":s3[3]}
print(pd.DataFrame(data))

q1=np.array([1.3432934216]*len(s2[3]))
q2=np.array([np.pi]*len(s3[3]))
fig, (ax1, ax2) = plt.subplots(1, 2)
fig.suptitle('SIMPSON METHOD (TOLERANCE = 0.1e-8)')
ax1.plot(s2[4],s2[3],marker="*",label="I1 using SIMPSON",linestyle='dashed')
ax1.plot(s2[4],q1,label="EXACT",c="red",linewidth=1)
ax2.plot(s3[4],s3[3],marker="*",label="I1 using SIMPSON",linestyle='dashed')
ax2.plot(s3[4],q2,label="EXACT",c="red",linewidth=1)
ax1.grid()
ax1.legend()
ax1.set(xlabel="R",ylabel="Integral",title="Integral I1 calculated using SIMPSON")
```

```python
ax2.set(xlabel="R",ylabel="Integral",title="Integral I2 calculated using SIMPSON")
ax2.grid()
ax2.legend()
plt.show()
```