Gauss-Hermite

SGTB Khalsa College, University of Delhi

Preetpal Singh(2020PHY1140)(20068567043)

Ankur Kumar(2020PHY1113)(20068567010)

Unique Paper Code: 32221401

Paper Title: Mathematical Physics III

Submitted on: February 8, 2022

B.Sc(H) Physics Sem IV

Submitted to: Dr. Mamta

Contents

1 Theory

Hermite Gauss Quadrature method for Integration

Hermite-Gauss quadrature is an extension of Gaussian quadrature over the interval $(-\infty, \infty)$. It fits all polynomials of degree $2m-1$ exactly. This method approximates the value of integrals of the following kind:

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

here x_i is the i-th root of Hermite polynomial $H_n(x)$ and the weight w_i is,

$$w_i = \frac{2^{n-1} n! \sqrt{\pi}}{n^2 \left[H_{n-1}(x_i)\right]^2}$$

Hermite differential equation

The Hermite polynomials are solutions of second-order linear Hermite differential equation:

$$y'' - 2xy' + 2ky = 0$$

where constant k can be any real number.

The Hermite polynomials are defined by given formula,

$$H_n(x) = (-1)^n e^{x^2} \frac{d^n}{dx^n}\left(e^{-x^2}\right)$$

The first five Hermite polynomials are denoted by H_0, H_1, H_2, H_3, H_4, which are following,

$$H_0(x) = 1$$

$$H_1(x) = 2x$$

$$H_2(x) = 4x^2 - 2$$

$$H_3(x) = 8x^3 - 12x$$

$$H_4(x) = 16x^4 - 48x^2 + 12$$

Recursion Formulae for Hermite Polynomials

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

Orthogonality properties for Hermite Polynomials

$$\int_{-\infty}^{\infty} e^{-x^2} H_m(x)H_n(x)dx = 0 \quad whenm \neq n$$
$$\int_{-\infty}^{\infty} e^{-x^2} \left[H_n(x)\right]^2 dx = 2^n n!\sqrt{\pi} \quad whenm = n$$

2-point Quadrature Formula for Gauss Hermite Polynomials

As we know for N point method,

$$I_n = \sum_{i=1}^{N} w_i f\left(x_i\right)$$
$$I_n = \int_{-\infty}^{\infty} g(x)e^{-x^2} dx = \omega_0 g\left(x_0\right) + \omega_1 g\left(x_1\right) + \cdots + \omega_n g\left(x_n\right)$$

For 2 point formula,

$$I(g) = \int_{-\infty}^{\infty} g(x)e^{-x^2} dx = \omega_0 g\left(x_0\right) + \omega_1 g\left(x_1\right) = I_2(g)$$

As there are 4 unknowns $x_0, x_1, \omega_0, \omega_1$ so approximating these with 4 polynomials of degree less than or equal to 3.

$$g(x) = 1, x, x^2, x^3$$

$$g(x) = 1 \quad \int_{-\infty}^{\infty} g(x)e^{-x^2} dx = \int_{-\infty}^{\infty} e^{-x^2} dx = \omega_0 g\left(x_0\right) + \omega_1 g\left(x_1\right) = \omega_0 + \omega_1 = \sqrt{\pi}$$
$$g(x) = x \quad \omega_0 x_1 + \omega_1 x_2 = 0$$
$$g(x) = x^2 \quad \omega_0 x_1^2 + \omega_1 x_2^2 = \frac{\sqrt{\pi}}{2}$$
$$g(x) = x^3 \quad \omega_0 {x_1}^3 + \omega_1 {x_2}^3 = 0$$

Solving these 4 equations, we'll get

$$x_1 = \frac{1}{2}\sqrt{2}$$
$$x_2 = -\frac{1}{2}\sqrt{2}$$
$$\omega_1 = \frac{1}{2}\sqrt{\pi}$$
$$\omega_2 = \frac{1}{2}\sqrt{\pi}$$

2 Algorithm

Algorithm 1 n-point Gauss Hermite Quadrature rule

function MYHERMITEQUAD(f, n)
  $[herm\_zer, w] = h\_roots(n)$ ▷ Store the x values and weights in two lists
  herm = 0 ▷ initialize the summation

  for i in range(1, n+1):
    $herm+ = f(herm\_zer[i-1]) * w[i-1]$ ▷ loop to sum all values for the integral
  return $herm$ ▷ Returns the value of integral

3 Programming

```
#Ankur Kumar 2020PHY1113
#Preetpal Singh 2020PHY1140


from IntegrationModule import *
from scipy.integrate import quad
import numpy as np
import pandas as pd
from prettytable import PrettyTable

print("2020PHY1113")


ver_f = eval("lambda x:" + input("function to be integrated by Gauss Hermite , f(x)
    = "))

x = PrettyTable()

x.field_names = ["n-point", "Inbuilt Function", "My Function"]

x.add_row(["2", quad(lambda x: np.exp(-(x**2))*ver_f(x), -np.inf, np.inf)[0],
    MyHermiteQuad(ver_f, 2)])
x.add_row(["4", quad(lambda x: np.exp(-(x**2))*ver_f(x), -np.inf, np.inf)[0],
    MyHermiteQuad(ver_f, 4)])

print(x)

def Int_1(x):
    return 1/(1+x**2)

def Int_2(x):
    return np.exp(x**2)/(1+x**2)

I1 = []
I2 = []
n = []
for i in range(1, 8):
    val_1 = MyHermiteQuad(Int_1, 2**i)
    I1.append(val_1)


for i in range(1, 8):
    val_2 = MyHermiteQuad(Int_2, 2**i)
    I2.append(val_2)
    n.append(2**i)

data = {

```

```
  'n': n,
  'I_1': I1,
  'I_2': I2

}

df = pd.DataFrame(data)
numpy_array = df.to_numpy()
np.savetxt("quad-herm-1113.out.txt", numpy_array, fmt = "%f")
print(df)


#Comparsion With Simpson Method

func_1 = lambda x: np.exp(-(x**2))/(1+x**2)
func_2 = lambda x: 1/(1+x**2)

simp1 = []
simp2 = []
upperLimit = []
lowerLimit = []

for i in range(0,5):

    temp1 = My_Simp(func_1, -10**i, 10**i, 100000)
    simp1.append(temp1)

    temp2 = My_Simp(func_2, -10**i, 10**i, 100000)
    simp2.append(temp2)

    temp3 = 10**i
    upperLimit.append(temp3)

    temp4 = -10**i
    lowerLimit.append(temp4)


data1 = {
    'Lower Limit': lowerLimit,
    'Upper Limit': upperLimit,
  'I_1(Simpson)': simp1,
  'I_2(Simpson)': simp2

}

df = pd.DataFrame(data1)
print(df)
```

4 Discussion

Verification of n-point Gauss Hermite

We know that for n-point Gauss Hermite the method will give exact results upto $2n - 1$ order of polynomials.
For $n = 2$ the integral is exact upto 3rd order polynomials.
For $n = 4$ the integral is exact upto 7th order polynomials.

function to be integrated by Gauss Hermite , f(x) = x**2

n-point	Inbuilt Function	My Function
2	0.8862269254527599	0.8862269254527577
4	0.8862269254527599	0.8862269254527573

Figure 1: Polynomial of Order 2

For the given 2nd order polynomial it can be seen that both 2-point and 4-point quadrature gives exact results.

function to be integrated by Gauss Hermite , f(x) = x**6

n-point	Inbuilt Function	My Function
2	3.3233509704478426	0.22155673136318937
4	3.3233509704478426	3.323350970447833

Figure 2: Polynomial of Order 6

For the given 6th order polynomial it can be seen that 2-point quadrature fails to give an exact result and 4-point is still accurate as predicted.

function to be integrated by Gauss Hermite , f(x) = x**8

n-point	Inbuilt Function	My Function
2	11.63172839656745	0.11077836568159467
4	11.63172839656745	8.973047620209144

Figure 3: Polynomial of Order 8

For the given 8th order polynomial it can be seen that both 2-point and 4-point quadrature fail to give exact results.

Evaluating I_1 and I_2

	n	I_1	I_2
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 4: Value of Integrals with change in n

It can be seen that as value of n increases the value of integral approaches the true value, but the second integral does not reach the true value of π even with 128 point quadrature.

Computation using two-point quadrature

Integral I

$$f(x) = \frac{1}{1+x^2}$$

$$I_1 = \sum_{i=1}^{N} w_i f(x_i)$$
$$I_1 = w_1 f(x_1) + w_2 f(x_2)$$
$$I_1 = \frac{\sqrt{\pi}}{2} f\left(\frac{1}{\sqrt{2}}\right) + \frac{\sqrt{\pi}}{2} f\left(\frac{-1}{\sqrt{2}}\right)$$
$$I_1 = (0.886227)(0.666667) + (0.886227)(0.666667)$$
$$I_1 = 1.181636$$

Integral II

$$f(x) = \frac{e^{x^2}}{1+x^2}$$

$$I_1 = \sum_{i=1}^{N} w_i f(x_i)$$
$$I_1 = w_1 f(x_1) + w_2 f(x_2)$$
$$I_1 = \frac{\sqrt{\pi}}{2} f\left(\frac{1}{\sqrt{2}}\right) + \frac{\sqrt{\pi}}{2} f\left(\frac{-1}{\sqrt{2}}\right)$$
$$I_1 = (0.886227)(1.099148) + (0.886227)(1.099148)$$
$$I_1 = 1.948188$$

Comparison with Simpson Method

	Lower Limit	Upper Limit	I_1(Simpson)	I_2(Simpson)
0	-1	1	1.237644	1.570796
1	-10	10	1.343293	2.942255
2	-100	100	1.343293	3.121593
3	-1000	1000	1.343293	3.139593
4	-10000	10000	1.343293	3.141393

Figure 5: Comparison with Simpson Method

We change the values of both upper limit and lower limit towards infinity.
It can be seen that as the value of upper limit and lower limit goes on increasing both the integrals approach their true value.
With this method the value of second integral can be approximated to π.