

UNIVERSITY OF OSLO
COMPUTATIONAL PHYSICS

Project 3



UiO : **University of Oslo**

Authors:

Birgitte Madsen

Magnus Isaksen

Soumya Chalakkal

Autumn 2015



UiO : University of Oslo

Department of Physics

University of Oslo

Sem Sælands vei 24

0371 Oslo, Norway

+47 22 85 64 28

<http://www.mn.uio.no/fysikk/english/>

Course:

Computational Physics

Project number:

3

Link to GitHub folder:

https://github.com/birgimad/Project3/tree/master/Project3_BM_MI_SC

Hand-in deadline:

Friday, October 23, 2015

Project Members:

Birgitte Madsen

Magnus Isaksen

Soumya Chalakkal

Copies: 1

Page count: 19

Appendices: 0

Completed: Friday, October 23, 2015

The content of the report is freely available, but publication (with source) may only be made with the agreement of the authors.



ABSTRACT

The project solves a multidimensional integral function involving quantum mechanical expectation value of the correlation energy between two electrons which repel each other via the classical Coulomb interaction using Gaussian quadrature methods and Monte Carlo methods. In the Gaussian quadrature methods Legendre and Laguerre polynomials are used to solve the integral. It is observed that Laguerre polynomials gives better results compared to Legendre polynomials. A brute force Monte Carlo and improved Monte Carlo method using importance sampling are employed to evaluate the integral. The latter is found to be more more efficient in computing the integral in terms of computational speed and accuracy of the result among all the four methods that are used.



TABLE OF CONTENTS

Chapter 1	Introduction	1
Chapter 2	Method	3
2.1	Nature of the problem	3
2.1.1	Integral Written in Cartesian and Spherical Coordinates	4
2.2	Gauss-Legendre Method for Computing the Integral	4
2.3	Generation of Legendre Polynomials	6
2.3.1	Test of the Legendre Polynomial Generation	7
2.4	Laguerre Method for Computing the Integral	7
2.5	Monte Carlo Method	8
2.5.1	Brute force method	8
2.5.2	Improved Monte Carlo Method	9
Chapter 3	Results and Discussion	13
3.1	Solving the Integral by the Gauss-Legendre Method	13
3.2	Solving the Integral by the Laguerre Method	13
3.3	Monte Carlo method	14
3.3.1	Brute force	14
3.4	Computational Time for each Method	14
Chapter 4	Conclusion	17
	Bibliography	19

INTRODUCTION

The project mainly discusses about various integration methods namely Gaussian quadrature methods and Monte carlo methods that can be employed to solve an integral function. The integral we are interested in is the quantum mechanical expectation value of the correlation energy between two electrons which repel each other via the classical Coulomb interaction. In the Gaussian quadrature method two important orthogonal polynomials Legendre and Laguerre are used to evaluate the integral. Since Legendre polynomial is defined in the interval $[-1, 1]$, solving the integral gave unsatisfactory results. To improve the results the integral which was in cartesian coordinate is converted to spherical coordinates with different integration limits and Laguerre polynomial is introduced which is defined in the interval $[0, \infty]$. A comparison of the results using the two different polynomial is made.

In order to employ Monte Carlo methods we assumed the system can be described as a probability density function and random numbers were generated to cover uniformly in the interval $[0, 1]$. First a brute force approach using cartesian coordinates is used to find the solution then the method is improved with importance sampling in which the coordinates are changed to spherical coordinates as in the Laguerre method with the radial component being exponentially distributed. An error estimation is made in both cases. A comparison of all the method that we used to solve the integral is done in Sec. 3.4.

This report mainly consists of two sections. First section gives a short introduction on the nature of the problem and then move on to a detailed description of both the Gaussian quadrature and Monte Carlo methods for solving the integral. Chap. 3 is for the discussion and interpretation of results.

METHOD

In this chapter a short introduction of the nature of problem is given together with a brief analysis of the function whose integral is to be calculated is made. Furthermore, the Legendre and Laguerre method for computing the integral are presented. In these methods the Legendre and Laguerre polynomials are used to find the roots of the n 'th polynomial and the corresponding weights of these roots. From the polynomial roots, the function is evaluated, and all of these function values are added up, taking the weights into account. The Monte Carlo method solves the integral using probability density functions (PDF) to evaluate the function in random numbers.

The source codes for the algorithms described in this chapter can be found in the Github folder https://github.com/birgimad/Project3/tree/master/Project3_BM_MI_SC.

2.1 Nature of the problem

Even though the Schrödinger equation cannot be solved exactly for the helium atom or more complicated atomic or ionic species due to electron-electron interaction, the ground state energy of the helium atom can be calculated using approximate methods. One method is to assume that the electrons in helium atom occupies scaled hydrogen 1s orbital so that the product of the wave function of the two electrons can be given as

$$\Phi(r_1, r_2) = \exp[-\alpha(r_1 + r_2)] \quad (2.1)$$

in which $\Phi_{1s}(r_i) = \exp(-\alpha(r_i))$ is the single particle wave function for a particle at position r_i . α is a parameter that corresponds to the charge of helium atom and $r_i = \sqrt{x_i^2 + y_i^2 + z_i^2}$ is the cartesian coordinate of particle. The ground state correlation energy between these two electrons in the helium atom can be calculated by solving the integral

$$\left\langle \frac{1}{|\mathbf{r}_1 - \mathbf{r}_2|} \right\rangle = \int_{-\infty}^{\infty} \frac{e^{-2\alpha(r_1+r_2)}}{|\mathbf{r}_1 - \mathbf{r}_2|} d\mathbf{r}_1 d\mathbf{r}_2 \quad (2.2)$$

which is the quantum mechanical expectation value of the correlation energy between two electrons which repel each other via the classical coulomb interaction. It turns out that the closed-form solution of Eq. (2.2) is $5\pi^2/16^2$.

2.1.1 Integral Written in Cartesian and Spherical Coordinates

Writing out Eq. (2.2) in cartesian coordinates in which

$$\mathbf{r}_i = x_i \hat{\mathbf{i}} + y_i \hat{\mathbf{j}} + z_i \hat{\mathbf{k}} \quad (2.3)$$

with $\hat{\mathbf{i}}$, $\hat{\mathbf{j}}$ and $\hat{\mathbf{k}}$ being unit vectors in the x-, y- and z-direction, respectively and $x_i, y_i, z_i \in (-\infty; \infty)$, yields

$$\left\langle \frac{1}{|\mathbf{r}_1 - \mathbf{r}_2|} \right\rangle = \int_{-\infty}^{\infty} \frac{e^{-2\alpha(\sqrt{x_1^2+y_1^2+z_1^2} + \sqrt{x_2^2+y_2^2+z_2^2})}}{\sqrt{(x_1-x_2)^2 + (y_1-y_2)^2 + (z_1-z_2)^2}} dx_1 dy_1 dz_1 dx_2 dy_2 dz_2 \quad (2.4)$$

with the integral being from $-\infty$ to ∞ over all six variables. This representation of the integral will be used in the Gauss-Legendre method for computing the integral. The following lines of code shows how to compute the function to be integrated in c++.

```
double int_function(double x1, double x2, double x3, double y1, double y2, double y3)
{
    int alpha = 2.0;
    double denominator = pow((x1-y1),2)+pow((x2-y2),2)+pow((x3-y3),2);
    double exponent = exp(-2.0*alpha*(sqrt(x1*x1+x2*x2+x3*x3)+sqrt(y1*y1+y2*y2+y3*y3)));
    if (denominator < pow(10,-6)){return 0;}
    else return exponent/sqrt(denominator);
}
```

A problem arises if the denominator becomes very small, when computing the fraction, since this would cause the function value to become very large. To avoid this problem, the function value is set equal to zero if this is the case. For the Laguerre method, spherical coordinates with $r_i \in [0; \infty)$, $\theta_i \in [0; \pi]$, and $\phi_i \in [0; 2\pi]$ are used. When writing the considered integral in spherical coordinates, the problem to solve becomes

$$\left\langle \frac{1}{|\mathbf{r}_1 - \mathbf{r}_2|} \right\rangle = \int \frac{e^{-2\alpha(r_1+r_2)}}{\sqrt{r_1^2 + r_2^2 - 2r_1 r_2 \cos(\beta)}} r_1^2 r_2^2 \sin(\theta_1) \sin(\theta_2) dr_1 dr_2 d\theta_1 d\theta_2 d\phi_1 d\phi_2 \quad (2.5)$$

with the integral having the limits as described above. $\cos(\beta)$ is then given by

$$\cos(\beta) = \cos(\theta_1)\cos(\theta_2) + \sin(\theta_1)\sin(\theta_2)\cos(\phi_1 - \phi_2) \quad (2.6)$$

2.2 Gauss-Legendre Method for Computing the Integral

To be able to use the Gauss-Legendre method to compute the integral in Eq. (2.2), the limits of the integral must be made finite. Since the wave function

$$e^{-2\alpha x} \quad (2.7)$$

rapidly goes toward zero as x is increased (see Fig. 2.1), the integral can be approximated by the same integral with finite limits.

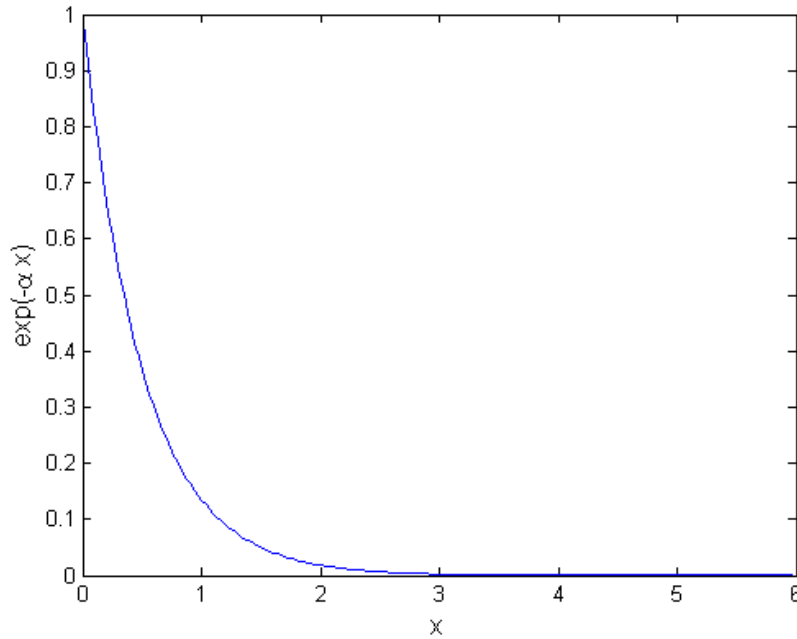


Figure 2.1. Plot of $e^{-2\alpha x}$ with $\alpha = 2$.

In this project, if the value is equal or smaller than 10^{-6} it is close enough to zero to neglect contributions from the part of the wave function when the wave function gives a value of this order. For $x = 4$ the value of the wave function is $e^{-2 \cdot \alpha \cdot 4} \approx 1.1 \cdot 10^{-6}$, when $\alpha = 2$. Hence, the considered integral that is to be solved by the Gauss-Legendre method is given by

$$\left\langle \frac{1}{|\mathbf{r}_1 - \mathbf{r}_2|} \right\rangle = \int_{-4}^4 \frac{e^{-2\alpha(r_1+r_2)}}{|\mathbf{r}_1 - \mathbf{r}_2|} d\mathbf{r}_1 d\mathbf{r}_2 \quad (2.8)$$

To be able to use the Gauss-Legendre method, the limits have to be -1 and 1 . This is, however, easily obtained by a change in variables using the following quantity

$$\int_a^b f(t) dt = \int_{-1}^1 f\left(\frac{b-a}{2}x + \frac{b+a}{2}\right) dx \quad (2.9)$$

The first step of the Gauss-Legendre method is then to compute the roots of the n 'th Legendre polynomial. The i 'th root z_i is approximated by

$$z_i = \cos\left(\frac{\pi(4 \cdot i - 1)}{4n + 2}\right) \quad (2.10)$$

for large n . The n 'th Legendre polynomial $L_n(x)$ can then be computed using the recursive relation

$$(j+1)L_{j+1}(z) + jL_{j-1}(z) - (2j+1)zL_j(z) = 0 \quad (2.11)$$

with $L_{-1}(x) = 0$ and $L_0(x) = 1$. The code for generation the Legendre polynomial can be found in Sec. 2.3 together with a test of the Legendre polynomial generating algorithm.

The derivative of the Legendre polynomial can then be computed as

$$L'_n(z) = \frac{-n \cdot z \cdot L_n(z) + n \cdot L_{n-1}(z)}{1 - z^2} \quad (2.12)$$

Newton's method is then applied to find the best estimation of the roots by considering the expression

$$z_1 = z - \frac{L_n(z)}{L'_n(z)} \quad (2.13)$$

in which z is the first approximation of the root given by Eq. (2.10), and z_1 is a better approximation for the root. The algorithm for generation of the roots and the Legendre polynomials is run until the difference between z_1 and the root approximated by Eq. (2.10) is ultimately zero, which in this project is set to 10^{-6} .

The weight w_i of the i 'th root z_i is then obtained by

$$w_i = \frac{2}{(1 - z_i^2)(L'_n(z_i))^2} \quad (2.14)$$

2.3 Generation of Legendre Polynomials

The following lines of code generates the n 'th order Legendre polynomial from the recursive relation given in Eq. (2.11) and its roots. Since the roots are symmetric about 0, the for-loop is run from $i = 1$ to $i < m$ for $m = n + 1$ with n being the order of the computed polynomial and hence the length of the array containing the roots and the array containing the respective weights of the roots.

```
//Code for computing the Legendre polynomials to determine the roots of the n'th
Legendre polynomial
for (int i = 1; i<=m;i++){
    root = cos(pi * (4*i-1) / (4*n + 2 ));
    //approximation of the root of the n'th polynomial
do{
    // This uses the recursive relation to compute the Legendre polynomials
    double L_plus = 1.0 , L = 0.0, L_minus;
    for (int j = 0; j < n; j++)
    {
        L_minus = L;
        L = L_plus;
        L_plus = (2.0*j +1)*root*L - j*L_minus ;
        L_plus /= j+1;
    }
    //derivative of the Legendre polynomial (L_plus)
    dev_L = (-n*root*L_plus + n*L)/(1-root*root);
    // Newton's method
    z = root;
    root = z - L_plus/dev_L;
} while(fabs(root - z) > pow(10,-6));
//compute values of x (array containing the roots) and w (array containing the
    roots)
    * w_temp1 = 2/((1-root*root)*(dev_L*dev_L));
    *(x_temp1++) = root;
    *(x_temp2--) = -root;
    *(w_temp2--) = *(w_temp1++);
}
```

The algorithm starts with an approximation of the i 'th root of the n 'th order Legendre polynomial as described in Eq. (2.10). After computation of the Legendre polynomial, the approximation is improved by Newton's method. If the difference between the improved approximation and the initial approximation is greater than zero (that is 10^{-6}), the n 'th order Legendre polynomial is computed using this new approximation, and with this new Legendre polynomial the previously improved approximation of the root is again improved, and once again, if the difference between the approximated roots is greater than zero, new Legendre polynomials are computed. This procedure is run until the difference between the approximation of the i 'th root and the improved approximation of the i 'th root is zero. When this is achieved, the root z_i is put into the i 'th and $(n - i)$ 'th entry of the array containing the roots, and the corresponding weights are computed by Eq. (2.14). This is done m , in which m is the lowest integer value of $(n + 1)/2$ times, and hence all the entrances of the arrays $x[n]$ and $w[n]$ will be filled with the roots and corresponding weights

2.3.1 Test of the Legendre Polynomial Generation

Tab. 2.1 provides the roots and corresponding weights of the n 'th Legendre polynomial. [1] For $n = 5$, these are

Table 2.1. Given roots and weights for the Legendre polynomial

Roots	Weights
0	0.568889
± 0.538469	0.478629
± 0.90618	0.236927

which is exactly what is obtained up to the same accuracy as in Fig. 2.1 by running the above lines of code for $n = 5$. (see https://github.com/birgimad/Project3/blob/master/Project3_BM_MI_SC/Data/LegendrePol_n_5.txt)

2.4 Laguerre Method for Computing the Integral

In the Gauss-Laguerre method integrals of the form

$$I = \int_0^{\infty} x^{\alpha} e^{-x} g(x) dx \quad (2.15)$$

can be solved using Laguerre polynomials and a weight function given by

$$W(x) = x^{\alpha} e^{-x} \quad (2.16)$$

Hence, if the variables of the integral in Eq. (2.2) are changed from cartesian coordinates to spherical coordinates with $r_i \in [0; \infty)$, $\theta_i \in [0; \pi]$, and $\phi_i \in [0; 2\pi]$, the integral can be solved by using the roots and corresponding weights of Legendre polynomials for θ_i and ϕ_i and use roots and corresponding weights of Laguerre polynomials for r_i . The algorithm for finding the roots and their corresponding weights of the Laguerre polynomials will, however, not be discussed here. An important note is, though, that since the

weight function includes x^α and e^{-x} , the function that is to be evaluated in the roots of the n 'th Laguerre polynomial is

$$g(r_1, r_2, \theta_1, \theta_2, \phi_1, \phi_2) = \frac{e^{-2\alpha(r_1+r_2)+r_1+r_2}}{\sqrt{r_1^2 + r_2^2 - 2r_1r_2\cos(\beta)}} \sin(\theta_1)\sin(\theta_2) \quad (2.17)$$

with $\cos(\beta)$ described by Eq. (2.6).

2.5 Monte Carlo Method

Monte Carlo methods are used for problems that can be described using probability distribution functions. This part will look at a brute force approach using two different probability distribution functions (PDF). With the probability distribution function the approach is to characterizing a problem using random numbers. Then getting a sum out of the function evaluated and considering the error made by finding the variance using this method.

2.5.1 Brute force method

The brute force method uses the uniform probability distribution function. Where the stochastic variables are approximated using random numbers. Where the probability distribution function gives a probability or the relative frequency for the stochastic variables. The probability function $Prob(X)$ is found by Eq. (2.18), where $p(x)$ is the probability for one of the variables.

$$Prob(a < X < b) = \int_a^b p(x)dx \quad (2.18)$$

The method consist in choosing a random number generator to make the stochastic values that goes in to the probability distribution function. This gives out the function values that make the integral sum. This also gives the possibility to generate the variance for estimating the error.

The integral sum is made using Eq. (2.19).

$$I = \frac{1}{N} \sum_{i=1}^N f(x) \approx \langle f(x) \rangle \quad (2.19)$$

The variance using Eq. (2.20)

$$\sigma^2 = \langle x^2 \rangle - \langle x \rangle^2 \quad (2.20)$$

where $\langle x^2 \rangle$ is the expectation value that is found from the given PDF using Eq. (2.21).

$$\langle x^k \rangle = \frac{1}{N} \sum_{i=1}^N x_i^k p(x) \quad (2.21)$$

And for $\langle f(x) \rangle$ it is found using Eq. (2.22)

$$\langle f(x) \rangle = \frac{1}{N} \sum_{i=1}^N f(x) p(x) \quad (2.22)$$

Part of the code is included below, and it shows the for loop that generates the integral. It does this by generating a random number for all of the 6-dimensions that exists. These variables it sends to the function that is integrated, returning a function value that goes in to the sum generating the end integral. This is done N times, preferably with N as high as possible. After the for loop the sum is multiplied with the Jacobi determinant giving the end integral. Also the variance is generated by the function value and used to produce the standard deviation sigma.

```
for (int i=0;i<N;i++){
    for (int j=0;j<6;j++){
        x[j] = - length +2*length*((double) rand() / (RAND_MAX)); //random
            numbers generated in the interval(-length,length)

        //cout<<x[j]<<endl;

    }

    fx = func(x[0],x[1],x[2],x[3],x[4],x[5]);
    Montec += fx;
    Montesqr += fx*fx;

}

double MC = Montec/((double) N );
double Montesqr1= Montesqr/((double) N);
double variance= Montesqr1-MC*MC;
double volume = pow(length,6);
cout << " Integral = " << jacobidet*MC << endl;
cout << "sigma = " << jacobidet*sqrt(variance/ ((double) N)) << endl;
```

2.5.2 Improved Monto Carlo Method

In the improved brute force Monto Carlo method the coordinates of the function to be integrated are changed from cartesian coordinates to spherical coordinates with r_i being exponentially distributed on the interval $[0; \infty)$, θ_i being uniformly distributed on the interval $[0; \pi]$, and ϕ_i being uniformly distributed on the interval $[0, 2\pi]$. That means that the PDF's of θ_i and ϕ_i are given by

$$p(\theta_i) = \frac{1}{\pi} \quad \text{and} \quad p(\phi_i) = \frac{1}{2\pi} \quad (2.23)$$

whilst the PDF for the radial component is given by

$$p(r_i) = \exp(-r_i) \quad (2.24)$$

Since the considered function in spherical coordinates is given by Eq. (2.5), it is an advantage to do a change of the radial variables into

$$u_i = 2\alpha r_i \quad (2.25)$$

which is also exponentially distributed in the interval $[0; \infty)$ with the PDF given by Eq. (2.24). This change in variables could of course also have been applied in the Laguerre method described in Sec. 2.4. This gives

$$du_i = 2\alpha dr_i \quad (2.26)$$

and hence the function that is to be evaluated becomes

$$f(u_1, u_2, \theta_1, \theta_2, \phi_1, \phi_2) = \frac{1}{(2\alpha)^5} \frac{e^{-u_1-u_2}}{\sqrt{u_1^2 + u_2^2 - 2u_1u_2\cos(\beta)}} u_1^2 u_2^2 \sin(\theta_1) \sin(\theta_2) \quad (2.27)$$

When dividing this function by the PDF's for the radial components u_1 and u_2 , it is evident that the exponential $e^{-u_1-u_2}$ will disappear. That means that when evaluating the integral with the Monte Carlo method, it becomes

$$I = \frac{1}{N} \sum_{i=1}^N \frac{\tilde{f}(u_i, \tilde{u}_i, \theta_i, \tilde{\theta}_i, \phi_i, \tilde{\phi}_i)}{p(\theta_i)p(\tilde{\theta}_i)p(\phi_i)p(\tilde{\phi}_i)} = \frac{1}{N} \sum_{i=1}^N \frac{\tilde{f}(u_i, \tilde{u}_i, \theta_i, \tilde{\theta}_i, \phi_i, \tilde{\phi}_i)}{4\pi^4} \quad (2.28)$$

in which $\tilde{f} = f/e^{-u_1-u_2}$.

The source code below shows the for loop for the improved Monte Carlo method can be seen below. The difference between this and the for loop for the brute force Monte Carlo method given in Subsec. 2.5.1, is the definition of the random variables and explicit statement of the total PDF px .

```

for (int i=0; i<N; i++){
    for (int j=0; j<6; j++){
        x[j] = ((double) rand() / (RAND_MAX));
        //random numbers generated in the interval(0,1)
    }
    //making the random numbers for u_i, theta_i and phi_i on respective interval.
    double u1 = -log(1-x[0]);
    double u2 = -log(1-x[1]);
    double theta1 = x[2]*pi;
    double theta2 = x[3]*pi;
    double phi1 = x[4]*2*pi;
    double phi2 = x[5]*2*pi;
    //evaluating the function in the random u_i, theta_i and phi_i
    fx = func(u1,u2,theta1,theta2,phi1,phi2);
    //computing the product of the PDF's of theta_i and phi_i
    px = 1/(4*pi*pi*pi*pi);
    //computing the parts to be summed up
    Montec += fx/px;
    Montesqr += (fx/px)*(fx/px);
}
//computing integral value and variance from which the standard deviation is
    calculated

```

```

double integral_value = Montec/((double) N );
double Montesqr1 = Montesqr/((double) N);
double variance = Montesqr1-integral_value*integral_value;
double standard_deviation = sqrt(variance/ ((double) N));

```

The function *func* that is called in the for loop is in spherical coordinates and with the change of variable $u_i = 2\alpha r_i$ as described in this section, unlike the function called in the for loop for the brute force Monte Carlo method, which is in cartesian coordinates. The random radial number u_i is computed from the uniformly distributed random number x_j in $[0; 1]$ by the formula

$$u_i = -\ln(1 - x_j) \quad (2.29)$$

since the probability has to be conserved, yielding

$$p(u)du = \exp(-u)du = dx = p(x)dx \quad (2.30)$$

in which $p(x) = 1$ since x is uniformly distributed in $[0; 1]$. Integration of Eq. (2.30) from 0 to u yields

$$x = 1 - \exp(-u) \quad (2.31)$$

from which Eq. (2.29) follows.

RESULTS AND DISCUSSION

The results gained by solving the integral presented in Eq. (2.2) using the four different methods presented in Chap. 2 can be found in this chapter. The error of the computed integral value from the closed-form solution $5\pi^2/16^2$ is furthermore calculated for the methods, and in Sec. 3.4 the computational time and accuracy of the four methods are compared.

The results from running the codes for the Gauss-Legendre, Laguerre, brute force Monto Carlo and improved Monte Carlo method can be found in the GitHub folder https://github.com/birgimad/Project3/tree/master/Project3_BM_MI_SC.

3.1 Solving the Integral by the Gauss-Legendre Method

When solving the integral in Eq. (2.8) by Gauss-Legendre quadrature presented in Sec. 2.2 with the limits from -4 to 4 , $\alpha = 2$, and number of mesh points $n = 40$, the value of the integral becomes

$$GLd = 0.177118 \quad (3.1)$$

The closed-form solution to the integral with infinite limits, is given in Sec. 2.1, and has a value of $5\pi^2/16^2$. The percentage deviation between the computed estimation of the integral value and the closed-form solution, is then

$$\text{percentage deviation} = \frac{0.177118 - 5\pi^2/16^2}{5\pi^2/16^2} \cdot 100\% \approx -8.1\% \quad (3.2)$$

This deviation is quite large, and since it is difficult to run with a higher number of mesh points, it shows that this method is quite inaccurate.

3.2 Solving the Integral by the Laguerre Method

When solving the the integral with spherical coordinates with r_i being the root of the n 'th Laguerre polynomial and θ_i and ϕ_i being roots of the n 'th Legendre polynomial, the integral value with $n = 12$ is found to be

$$GLr_{12} = 0.192155 \quad (3.3)$$

which deviates from the closed-form solution by

$$\text{percentage deviation} = \frac{0.192155 - 5\pi^2/16^2}{5\pi^2/16^2} \cdot 100\% \approx -0.32\% \quad (3.4)$$

which is a great deal smaller than the deviation when using the Gauss-Legendre method for $n = 40$.

However, if the computed Laguerre method is run for $n = 30$, the computed value for the integral is

$$GLr_{30} = 0.195069 \quad (3.5)$$

which deviates from the closed form solution by

$$\text{percentage deviation} = \frac{0.195069 - 5\pi^2/16^2}{5\pi^2/16^2} \cdot 100\% \approx 1.2\% \quad (3.6)$$

and hence the deviation from the closed-form solution is found to be larger for $n = 30$ than for $n = 12$, which is intuitively not what was to be expected.

3.3 Monte Carlo method

3.3.1 Brute force

Monte Carlo method is used on the same problem as before. It uses random numbers and a uniform probability distribution. It gives the result Tab. 3.1. Where the Integral value is close to but not exactly the same as the closed form solution. Also it looks like there is something not working correctly with the standard deviation, it gives values that are too small, when comparing the result with the closed form solution.

Table 3.1. Results from the brute force method.

Number of integration points	Integral value	standard deviation
10^7	0.19988	0.000282091
10^8	0.175551	0.000126166
10^8	0.183602	0.000168347

3.4 Computational Time for each Method

The Tab. 3.2 and Tab. 3.3 shows computational time for some selected runs of the Gauss-Legendre, Laguerre, brute force Monte Carlo and improved Monte Carlo methods for different N 's together with the gained result by running the codes with the selected N 's.

There is a big difference between the methods, both in run time and accuracy. Both the Gaussian-Legendre and Gaussian-Laguerre methods are slow and inaccurate compared to the Monte Carlo methods. They also require small N values to be able to run, which also limits their accuracy. The Gaussian-Laguerre method produces more accurate results than the Gaussian-Legendre method (see Tab. 3.2, but the computational time is almost twice as big.

Both the brute force Monte Carlo method and the improved gives more accurate results than the Gaussian methods when the N value is high. At low N values they are giving worse results than the Gaussian

Table 3.2. Computational time for selected runs of the Gauss-Legendre and Laguerre method.

Method	N	Result	Time (sec)
Legendre	12	0.0548094	0
	20	0.127513	15
	30	0.163743	179
Laguerre	12	0.192155	1
	20	0.195636	29
	30	0.195069	339

Table 3.3. Computational time for selected runs of the brute force Monte Carlo (MC) and improved Monte Carlo method.

Method	N	Integral	Standard deviation	Time (sec)
Brute force MC	10^6	0.0975576	0.0170543	0
	10^7	0.229248	0.0444382	6
	10^8	0.19671	0.0192389	65
Improved MC	1000	0.219422	0.033404	0
	10^4	0.19491	0.0124651	0
	10^5	0.188323	0.00347509	0
	10^6	0.192977	0.000980611	0
	10^7	0.192412	0.00031521	8
	10^8	0.19286	0.000104762	82

methods as seen in Tab. 3.2 and Tab. 3.3. But the run time is much better, they can do large N values quicker than the Gaussian. If comparing the two Monte Carlo methods listed in Tab. 3.3, the improved one gives better results with a small N than the brute force one, but the brute force method is quicker as it has fewer calculations to make. But as the improved one gives the same accuracy with a smaller N value, it can get the same accuracy just as fast or faster than the brute force one.

CONCLUSION

Gaussian quadrature methods and Monte Carlo methods were used to solve a multidimensional integral representing quantum mechanical expectation value of the correlation energy between two electrons which repel each other via the classical Coulomb interaction. In Gaussian quadrature method at first the integral was calculated using Legendre polynomial. The weights and roots of the polynomial for different mesh points were found and the integral was calculated by setting the limit as $[-4, 4]$. About 40 mesh points were needed to get a result with less percentage deviation. Since Legendre polynomials are defined for x in $[-1, 1]$, and the integral has limits $(-\infty, \infty)$ in cartesian coordinates, the results obtained was unsatisfactory. For betterment of results, the coordinate frame of the function was changed to spherical coordinate system with r_i in $[0, \infty)$, θ in $[0, \pi]$, and ϕ in $[0, 2\pi]$. The roots and weights of the Laguerre polynomials was used to find r_i whereas both θ and ϕ were found using Legendre polynomials due to their finite limits. This gave improved results.

A different approach was employed using brute force Monte Carlo method by considering the system as a probability distribution function which ended up in good results for larger mesh points. Brute force Monte Carlo calculation was improved using importance sampling in which a transformation to spherical coordinates is done and exponential distribution is introduced. Now the results became far more better with less standard deviation. This study shows the efficiency of the Monte Carlo methods compared to Gauss-Legendre and Laguerre methods for solving multidimensional integrals.



BIBLIOGRAPHY

- [1] “Wolfram mathworld: Legendre-gauss quadrature.” <http://mathworld.wolfram.com/Legendre-GaussQuadrature.html>. Accessed: 2015-10-23.