Numerical integration
FYS4150

Written by:

Marius Enga, Patryk Krzyzaniak
Mohamed Ismail and Kristoffer Varslott

Department of Physics UiO

UiO : University of Oslo

October 21, 2019

Contents

Abstract

The correlation energy between two electrons is a regularly occurring integral that must be solved multiple times in many quantum mechanical applications. The overarching goal of this report is to evaluate the ground state correlation energy of the helium atom using different numerical methods, and comparing these results with the analytical result. Properties of the orthogonal Legendre and Laguerre polynomials will be used when we are solving the integral using Gauss Quadrature method. Monte Carlo integration method utilizes statistical properties to approximate the integral. Our results show how even powers of Legendre polynomials underestimates the exact solution while odd powers start with an overestimation until they both converges a bit under the analytical result. Change of coordinate system from Cartesian to spherical utilizes the orthogonal properties of Laguerre, resulting in faster and better convergence than with Legendre polynomials. Results from Monte Carlo method shows very high numerical precision, far above Gaussian quadrature, but becomes time consuming for very high numbers of iterations. Parallelization of the Monte Carlo integration program gives a fast and reliable method for numerical integration with very high accuracy converging to the exact value.

I. Introduction

In quantum physics when dealing with many-particle systems, the Hamiltonian of the system depends on the kinetic motion of the nuclei and electrons, repulsion and attraction between the particles dependent on their charges and any other external potentials and perturbations on the system. For a system with N_e electrons and N_n nuclei, using the Born-Oppenheimer approximation [1] where the nuclei are approximated as point-like charges with fixed positions, the full Hamiltonian of the system can be split into a nuclear Hamiltonian and an electronic Hamiltonian. Most often we are interested in the energies of the electrons of the system, so we use the electronic Hamiltonian as shown below.

$$\hat{H} = -\sum_{j=1}^{N_e} \frac{\hbar^2 \nabla_j^2}{2m_e} + \frac{q^2}{4\pi\varepsilon_0} \sum_{i=1}^{N_e-1} \sum_{j=i+1}^{N_e} \frac{1}{|\boldsymbol{r_i} - \boldsymbol{r_j}|} - \frac{q^2 Z_\alpha}{4\pi\varepsilon_0} \sum_{i=1}^{N_e} \sum_{\alpha=1}^{N_n} \frac{1}{|\boldsymbol{r_i} - \boldsymbol{R_\alpha}|}$$

This Hamiltonian involves the kinetic energy of all electrons, the electron-electron repulsion, and the electron-nuclei attraction. The first and last term is easily calculated as they only involve the electrons movement and their positions relative to the fixed nuclei in the system. Repulsion between the electrons however is not so straight forward, and can not be calculated analytically as it stands.

Our aim is to solve the repulsion integral using different numerical methods for integration, for a system containing two electrons. The second approximation we do after Born-Oppenheimer, are Hartree's method [2] where we write the full wave function as a product of single electron wave functions. This approximation does not respect Pauli's exclusion principle, but it simplifies calculations significantly.

$$\Psi(\boldsymbol{r_1}, \boldsymbol{r_2}) \approx \psi_1(\boldsymbol{r_1})\psi_2(\boldsymbol{r_2})$$

For single electron wave functions we will be using the results from solving the Schrödinger equation for the hydrogen atom and look at each electrons wave function as hydrogen-like [3].

$$\psi_{nlm}(r, \theta, \phi) = \sqrt{\left(\frac{2}{na_0}\right)^3 \frac{(n-l-1)!}{2n[(n+l)!]^3}} e^{-r/na_0} \left(\frac{2r}{na_0}\right)^l \mathcal{L}_{n-l-1}^{2l+1}\left(\frac{2r}{na_0}\right) \cdot Y_l^m(\theta, \phi)$$

For the ground state of helium, both electrons are in the $1s$ state, with both having different spin, but we neglect the spin part for our calculations. Each electron will interact with both protons in the core, resulting in a factor $\frac{1}{2}$ in the Bohr radius. The resulting wave functions to be used are then given by equation 1 below.

$$\psi_{100} = \frac{1}{\sqrt{\pi(\frac{a_0}{2})^3}} e^{-2r/a_0} \implies \Psi(\boldsymbol{r_1}, \boldsymbol{r_2}) \approx \frac{8}{\pi {a_0}^3} e^{-2(r_1+r_2)} \tag{1}$$

$$\left\langle \frac{1}{|\boldsymbol{r_1} - \boldsymbol{r_2}|} \right\rangle = \int_{-\infty}^{\infty} d\boldsymbol{r_1} d\boldsymbol{r_2} e^{-4(r_1+r_2)} \frac{1}{|\boldsymbol{r_1} - \boldsymbol{r_2}|} \tag{2}$$

Our main goal as explained earlier are the numerical integration of this kind of functions, so we will do one final approximation where we use Hartree atomic units [2] i.e. setting the four fundamental constants $m_e, e, \hbar$ and $\frac{1}{4\pi\varepsilon_0}$ to unity, and removing the normalization constant $8/\pi {a_0}^3$. This simplifies the integral into equation 2, with $r_i = \sqrt{{x_i}^2 + {y_i}^2 + {z_i}^2}$, which we will solve in Cartesian and spherical coordinates. Note: There exists analytical solutions, although with approximations as described above, on the full wave function for the helium atom, containing repulsion of only two electrons. One of these solutions are shown in appendix A, with the result $\frac{5\pi^2}{256}$, which we will compare our numerical results with. The change of integral from Cartesian to spherical coordinates will be included in appendix B. The methods to be compared for solving this integral numerically are Gaussian Quadrature, using both Legendre polynomials and Laguerre polynomials and Monte Carlo integration.

II. Method

i. *Gaussian Quadrature*

Gaussian Quadrature (GQ) is a well known method, which is suitable for many applications. This method is used to evaluate an arbitrary integral of a function, with the help of orthogonal polynomials. The approximation of the integrals may be written as such:

$$I = \int_a^b f(x)dx = \int_a^b W(x)g(x) \approx \sum_{i=0}^{N-1} w_i g(x_i) \tag{3}$$

Where $W(x)$ is the weight-function, w_i is the associated weights, and the function $g(x_i)$ is evaluated in the mesh-points x_i. The weight-function is extracted from the function $f(x)$, yielding a product of the weights-function and $g(x)$. It is possible to write the function $g(x)$ as a set of arbitrary polynomials of degree $2N-1$.

$$I = \int_a^b f(x)dx \approx \int_a^b P_{2N-1}(x)dx = \sum_{i=0}^{N-1} w_i P_{2N-1}(x_i) \tag{4}$$

Here we can see that we are able to represent the function with a higher order polynomial of degree $2N-1$. This is due to the fact that we have 2N functions. N for number of mesh-points and N for the weight-function. We will see that the mesh-points are the zeros of the chosen orthogonal polynomial of order N.

We can see that the function is evaluated in the mesh-points, and to be able to find the roots of the chosen orthogonal polynomial we can use various root-finding methods. In this article newtons method is being used. The use of the orthogonal polynomials is vital for this method, and is a key component for determining the weights. The weights are determined from the inverse of a matrix consisting of orthogonal polynomials evaluated in x_i.

i.1. *Gauss-Legendre Quadrature*

Still, our main goal is to evaluate the arbitrary integral of $f(x)$. The Gauss-Legendre Quadrature (GQ-Legendre) takes advantage of the Legendre polynomials. In this method all the weigths and mesh-points is being determined from these Legendre polynomials. These kind of polynomials are well studied and are the solutions to the following differential equation defined for $x = [-1, 1]$:

$$(1 - x^2)y'' - 2xy' + n(n + 1)y = 0 \qquad\qquad n = 0, 1, 2, 3, ...,$$

We know that the Legendre polynomials inhabits the orthogonality relation.

$$\int_{-1}^{1} L_i(x)L_j(x)dx = \frac{2}{2i + 1}\delta_{ij} \tag{5}$$

This relation is vital for further calculations. We can rewrite the Legendre polynomials as a recursion relation for simplification:

$$L_{j+1}(x) = \frac{(2j + 1)xL_j(x) - jL_{j-1}}{j + 1} \tag{6}$$

We choose the normalization condition for $L_N(1) = 1$. This and the orthogonality relation yields the four first Legendre polynomials.

$$\begin{aligned} L_0(x) &= 1 \\ L_1(x) &= x \\ L_2(x) &= \frac{1}{2}(3x^2 - 1) \\ L_3(x) &= \frac{1}{2}(5x^3 - 3x) \end{aligned}$$

In order to obtain the desired mesh-points and weights we need to define an arbitrary polynomial Q_{N-1}, where we can represent Q_{N-1} with Legendre polynomials.

$$Q_{N-1}(x) = \sum_{k=0}^{N-1} \alpha_k L_k(x) \tag{7}$$

We can now implement the orthogonality property of the Legendre polynomials. Multiplying both sides of equation 7 with $\int_{-1}^{1} L_N(x)$, gives us:

$$\int_{-1}^{1} L_N(x)Q_{N-1}(x)dx = \sum_{k=0}^{N-1} \alpha_k \int_{-1}^{1} L_N(x)L_k(x)dx = 0$$

Where the right-hand side is always zero because we have chosen a summation which goes from $k = 0 \to N - 1$, and this summation does not give $L_k = L_N$. The integral on the right-hand side therefore always becomes zero.

$$\int_{-1}^{1} L_N(x)Q_{N-1}(x)dx = 0$$

Further on we can now describe P_{2N-1} through polynomial division:

$$P_{2N-1}(x) = L_N(x)P_{N-1}(x) + Q_{N-1}(x)$$

We stated earlier in equation 4, that the integral of the function may be approximated with a set of polynomials of degree $2N-1$:

$$\int_{-1}^{1} f(x)dx \approx \int_{-1}^{1} P_{2N-1}(x)dx = \int_{-1}^{1} L_N(x)P_{N-1} + Q_{N-1}dx$$

$$\int_{-1}^{1} P_{2N-1}(x)dx = \int_{-1}^{1} L_N(x)P_{N-1}(x)dx + \int_{-1}^{1} Q_{N-1}(x)dx$$

We can express any polynomial function with orthogonal Legendre polynomials, Legendre series. We then get an expression for P_{N-1}:

$$P_{N-1} = \sum_{k=0}^{N-1} \alpha_k L_k(x)$$

Inserting this into the equation above gives:

$$\int_{-1}^{1} P_{2N-1}(x)dx = \sum_{k=0}^{N-1} \alpha_k \int_{-1}^{1} L_N(x)L_k(x)dx + \int_{-1}^{1} Q_{N-1}(x)dx$$

Where we have as before, the first integral on the right-hand side of the equation above gives us zero due to orthogonality property. We then get:

$$\int_{-1}^{1} P_{2N-1}(x)dx = \int_{-1}^{1} Q_{N-1}(x)dx \tag{8}$$

We need to evaluate these integrals in the zeros of L_N, which we define as x_k. By this we see that

$$P_{2N-1}(x_k) = L_N(x_k)P_{N-1}(x_k) + Q_{N-1}(x_k) \tag{9}$$

Where $L_N(x_k) = 0$

$$P_{2N-1}(x_k) = Q_{N-1}(x_k) \qquad\qquad k = 0, 1, 2, \ldots, N-1$$

Now we have showed that the arbitrary polynomial P_{2n-1} of degree $2N-1$ is equal to the polynomial Q_{N-1} for a set of x_k. The values of x_k comes from the roots of $L_N(x)$. We now need to find α_0, this can be done by multiplying equation 7 with the correct integral of Legendre polynomial. We choose therefore $L_0(x)$ as our Legendre polynomial:

$$\int_{-1}^{1} L_0(x)Q_{N-1}(x)dx = \sum_{i=0}^{N-1} \alpha_i \int_{-1}^{1} L_0(x)L_i(x)dx$$

We know from earlier that $L_0(x) = 1$, and if we take a closer look at the summation on the right-hand side we see that only $i = 0$ survives. All other terms is zero due to the orthogonality properties. This means that we are left with:

$$\int_{-1}^{1} Q_{N-1}(x)dx = 2\alpha_0 \tag{10}$$

Since the orthogonality property showed in equation 5 gives us 2 for $i = j = 0$. Now we need to find a general expression for α_k which can be used to identify the weights. We can start by confirming that our Legendre polynomials are indeed orthogonal, which means that they are linearly independent of each other. By this we know that if we set up a matrix consisting of our Legendre polynomials, this matrix has an inverse.

$$\mathbf{L} = \begin{bmatrix} L_0(x_0) & L_1(x_0) & \dots & L_{N-1}(x_0) \\ L_0(x_1) & L_1(x_1) & \dots & L_{N-1}(x_1) \\ \dots & \dots & \dots & \dots \\ L_0(x_{N-1}) & L_1(x_{N-1}) & \dots & L_{N-1}(x_{N-1}) \end{bmatrix}$$

Where we can see that each column is linear independent, because of the orthogonality of Legendre polynomials. The set $x_0, x_1 \dots x_{N-1}$ is the zeros in L_N, which we define as the vector $\hat{\mathbf{x}}_k$. The fact that the Legendre polynomials is linear independent gives us the inverse property as stated earlier:

$$\mathbf{L^{-1}L} = \mathbf{I}$$

As said before, we know that we can express $Q_{N-1}(x_k)$ as a Legendre series. At the same time we know the value of Q_{N-1} at the zeros of L_N. We can then write an expression in matrix form.

$$Q_{N-1}(\hat{\mathbf{x}}_k) = \mathbf{L}\hat{\alpha}$$

Where $\hat{\alpha}$ now is a vector consisting of the values determined from the zeros of L_N, ranging from $k = 0, 1, 2 \dots N-1$. Further on we use the inverse property of $\mathbf{L}$:

$$\mathbf{L^{-1}}Q_{N-1}(\hat{\mathbf{x}}_k) = \hat{\alpha}$$

which is equivalent to:

$$\sum_{i=0}^{N-1} (L^{-1})_{ki} Q_{N-1}(x_i) = \alpha_k$$

Now we have a general expression for α, this can be inserted in to equation 10, with k = 0. At the same time we have already proved that equation 8 is true because of the orthogonality property.

$$\int_{-1}^{1} P_{2N-1}(x)dx = \int_{-1}^{1} Q_{N-1}(x)dx = 2a_0 = 2\sum_{i=0}^{N-1} (L^{-1})_{0i} Q_{N-1}(x_i)$$

We know also by now that $Q_{N-1}(x_i) = P_{2N-1}(x_i)$, where x_i is the zeros of L_0, which finally gives us:

$$\int_{-1}^{1} f(x)dx \approx \int_{-1}^{1} P_{2N-1}(x)dx = 2\sum_{i=0}^{N-1} (L^{-1})_{0i} P_{2N-1}(x_i)$$

Where we end up with the final equation that approximate the integral:

$$\int_{-1}^{1} f(x)dx \approx \sum_{i=0}^{N-1} \omega_i P_{2N-1}(x_i) \tag{11}$$

Here weights is $\omega_i = 2(L^{-1})_{0i}$, and the mesh-points is defined by the zeros of the orthogonal Legendre polynomial L_N.

i.2. *Gauss-Laguerre Quadrature*

The Gaussian Quadrature can also be used with Laguerre polynomials (GQ-Laguerre).

$$x\mathscr{L}_n''(x) + (1-x)\mathscr{L}_n'(x) + n\mathscr{L}_n(x) = 0$$

These polynomials are the solutions of the above differential equation with non-negative integers n, and they have different properties than the Legendre polynomials, with different weight-functions as

well as mesh-points. For the Legendre polynomials we saw that the weight-function $W(x)$ was one, however, for the Laguerre polynomials, the weigth-function is:

$$W(x) = x^{\alpha}e^{-x} \tag{12}$$

Where α is a number chosen depending on the function. That means that the function $f(x)$ need to match the given weight-function. If that is the case, a better approximation is being established. The integration limits is altered as well, since Laguerre is defined on the interval $[0, \infty)$. We then get the desired integral in the correct integral limits if we have a integral on the form:

$$\int_0^{\infty} f(x)dx = \int_0^{\infty} W(x)g(x)dx = \int_0^{\infty} x^{\alpha}e^{-x}g(x)dx \tag{13}$$

The first set of Laguerre polynomials is defined as:

$$\begin{aligned}\mathscr{L}_0(x) &= 1\\ \mathscr{L}_1(x) &= 1 - x\\ \mathscr{L}_2(x) &= 2 - 4x + x^2\\ \mathscr{L}_3(x) &= 6 - 18x + 9x^2 - x^3\end{aligned}$$

As the Legendre polynomials these orthogonal polynomials has an orthogonality relation. They do on the other hand, differ somewhat from Legendre, in that a factor of e^{-x} is included.

$$\int_0^{\infty} e^{-x}\mathscr{L}_i(x)\mathscr{L}_j(x)dx = \delta_{ij} \tag{14}$$

We would like to write these Laguerre polynomials as a recursion relation:

$$\mathscr{L}_{n+1}(x) = \frac{(2n + 1 - x)\mathscr{L}_n(x) - n\mathscr{L}_{n-1}(x)}{n + 1} \tag{15}$$

We need to evaluate the roots of these polynomials as we did for Legendre. We know that both the weights and mesh-points is different as well as the integration limit.

ii. *Monte Carlo*

Monte Carlo (MC) integration [4] is based on the mean value theorem, which states that the integral of a function, f, over a volume, V, is equal to the mean of f times V.

$$\int f(V)dV = \mu_f V$$

Where μ_f is the mean of the function f.

In a one dimensional space, the expression above can be expressed as

$$\int_a^b f(x)dx = \mu_f(b - a)$$

Thus, the target is to find a good estimate to the mean or the expected value of the function you are integrating. This is done by calculating the sample mean and using this as the estimate for the true mean value of the function.

$$\langle f \rangle = \frac{1}{N} \sum_{i=1}^{N} f(x_i) p(x_i)$$
$$\langle f \rangle \approx \mu_f$$

Where $\langle f \rangle$ is the sample mean and $p(x)$ is a probability distribution function (PDF).
This means that the sample mean $\langle f \rangle$, also known as the average, can be used to approximate the integral above.

$$I = \int_a^b f(x)dx \approx \langle f \rangle (b-a)$$

Other quantities that are important for MC calculations in addition to the average value $\langle f \rangle$ are the variance σ^2 and the standard deviation σ.
The variance is defined as

$$\sigma_f^2 = \langle f^2 \rangle - \langle f \rangle^2$$
$$\sigma_f^2 = \frac{1}{N} \sum_{i=1}^{N} f(x_i)^2 - \left(\frac{1}{N} \sum_{i=1}^{N} f(x_i) \right)^2$$

and is a measure of how much f deviates from its average over the region of integration while standard deviation is defined as the square root of the variance.

$$\sigma_f = \sqrt{\sigma_f^2}$$

Assuming that the above is a result acquired for a fixed value of N to be a measurement, would make it possible to recalculate this for different measurements. Each such measurement produces a set of averages for the integral I denoted $\langle f \rangle_l$ and for M measurements the integral is then given by,

$$\langle I \rangle_M = \frac{(b-a)}{M} \sum_{l=1}^{M} \langle f \rangle_l$$

we can then rewrite the variance of these series of measurements as

$$\sigma_M^2 = \frac{1}{M} \Big(\langle f^2 \rangle - \langle f \rangle^2 \Big) = \frac{\sigma_f^2}{M}$$

Then the standard deviation is defined as

$$\sigma_M = \sqrt{\sigma_M^2} = \sqrt{\frac{\sigma_f^2}{M}} = \frac{\sigma_f}{\sqrt{M}}$$

From this it is possible to see that the standard deviation is proportional to the inverse square root of the amount of measurements.

$$\sigma_M \sim \frac{1}{\sqrt{M}}$$

In the case where we sample infinitely many times, $M \to \infty$, the standard deviation would go towards zero and the integral would in theory converge to the exact answer.

III. Implementation

Programs used in this project can be found on https://github.com/patrykpk/FYS4150/tree/master/Project_3 and the "README.md" explains how to run the scripts. All calculations are done in C++, while the plotting is done in Python. Our C++ program is based on the codes found in the course's github repository https://github.com/CompPhysics/ComputationalPhysics/tree/master/doc/Projects/2019/Project3 and the content found in "Computational Physics Lecture Notes 2015" on numerical integration and Monte Carlo methods [4]. This section will include how we implemented the code and include some of the code:

i. *Gaussian Qadrature*

i.1. *Implementation of roots and weights*

When implementing Gauss-Legendre quadrature for solving our integral, there are some additional methods we use that are not described in the methods section. The integration limits in our integral goes over all real space, while the Legendre polynomials only are defined from -1 to 1. We want to be able to approximate the function we want to integrate by a Legendre polynomial of N'th degree over a suitable interval $[a, b]$, where a and b are chosen in such a way so that the function goes to zero, since $\lim_{x\to\pm\infty} f(x) = 0$. By doing a mapping to the new integration limits a and b through change of variable from x to t, the Legendre polynomials can be used for any limit.

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

Doing Gaussian quadrature we are mainly interested in finding the N roots of the N'th Legendre polynomial and its corresponding weights so that $I = \int_a^b f(x)dx \approx \sum_{i=0}^{N-1} w_i f(x_i)$ can be calculated.

The i'th root of the N'th degree polynomial lies in the interval $\left[\frac{2i-1}{2N+1}\pi, \frac{2i}{2N+1}\pi\right]$ as proved by H. Bruns [5]. By taking the initial guess $x_i = \cos(\pi(i-0.25)/(N-0.25))$ [6] we get a value inside this interval, that can be improved using Newton's method: $\tilde{x_i} = x_i - \frac{f(x_i)}{f'(x_i)}$. To be able to use Newton's method we need the polynomials at the point x_i and its derivative at the same point given by the following recursion relation in equation 16.

$$\frac{dL_n(x)}{dx} = \frac{n}{x^2-1}\big(xL_n(x) - L_{n-1}(x)\big) \tag{16}$$

$$w_i = -\frac{2\cdot\left(\frac{b-a}{2}\right)}{\left(1-x_i^2\right)\left(\frac{dL(x_i)}{dx}\right)^2} \tag{17}$$

For the weights, equation 17 will be used, where the factor $\left(\frac{b-a}{2}\right)$ accounts for the new weights after performing a mapping of the function. One important thing to notice are that the weights and roots are symmetric in the interval, so only the right half of roots and weights are calculated and then reflected over midpoint in interval $[a, b]$.

Listing 1: C++ Finding weights and roots of n'th degree of Legendre polynomials

```
/*
Pseudocode for finding weights and roots of n'th degree Legendre polynomial.
*/

EPSILON = 1.0E-10 // Desired approximation
m  = (n + 1)/2;   // Gives right half of roots in for loop

// Mapping of integration limits
bpa = 0.5 * (b + a);
bma = 0.5 * (b - a);

//Loops over desired roots
for(i = 1; i <= m; i++){
    xi = cos(pi * (i - 0.25)/(n + 0.5)); // Initial guess for the i'th root

    do{             // Find desired polynomial at point xi with derivative
        L1 = 1.0;
        L2 = 0.0;
        for(j = 1; j <= n; j++){
            L3 = L2;
            L2 = L1;
            L1 = ((2.0 * j - 1.0) * xi * L2 - (j - 1.0) * L3)/j; // Recursion
            }

        LDer = n * (xi * L1 - L2)/(xi * xi - 1.0); // Derivative at xi
        xi_new = xi;
        xi  = xi_new - L1/LDer; // Newton's method for better approximation to root

        } while(fabs(xi - xi_new) > EPSILON); // Check if root converges

    x[i]=bpa-bma*xi;                          // Scale the root to mapping
    x[n-1-i]=bpa+bma*xi;                      // Symmetric root
    w[i]=2.0*bma/((1.0-xi*xi)*LDer*LDer);     // Compute weight
    w[n-1-i]=w[i];                            // Symmetric weight
    }
```

The above pseudocode is an illustration on how we implemented Legendre polynomials in our program based on the book, Numerical recipes [7]. The same fundamental approach is also done for Laguerre polynomials, but the start guess for the roots and the calculations of the weights are a bit more complex, and without the need of a mapping of the integration limits.

i.2. *Evaluating integral using Gaussian Quadrature*

Now that we have a program that finds the roots and weights of the two orthogonal polynomials, we must implement GQ as equation 3. When calculating the Cartesian function 2, we implement this by doing a mapping from $[-\infty, \infty]$ to $[-\lambda, \lambda]$ using GQ-Legendre for every differential in the integral, giving us the correct weights. The roots are then inserted into the function and the integral is calculated using the sum of $w_i g(x_i)$. In appendix B we show how we change coordinate system from Cartesian to spherical system, and utilize the properties of Laguerre polynomials (equation 13) to end up with the integral in the subsection; Gaussian Quadrature Laguerre in this appendix. GQ-Laguerre are used for the radial part, while GQ-Legendre are used for the angular parts, where the pseudocode given in Listing 2 shows the basic steps for how calculate the integral in spherical coordinates.

Listing 2: C++ Multidimensional GQ

```
/*
Pseudocode for calculating integral using GQ.
*/

// Calls GQ-Laguerre to calculate the roots and weights.
gauss_laguerre(r, W_r, N, alpha);

// Calls GQ-Legendre to calculate the roots and weights.
gauss_legendre(0, PI, Theta, W_Theta, N);
gauss_legendre(0, 2*PI, Phi, W_Phi, N);

for (i = 1; i <= N; i++){    // r_1
for (j = 1; j <= N; j++){    // r_2
for (k = 0; k < N; k++){     // Theta_1
for (l = 0; l < N; l++){     // Theta_2
for (p = 0; p < N; p++){     // Phi_1
for (q = 0; q < N; q++){     // Phi_2

// Calculate products of all weights
Weights = W_r[i]*W_r[j]*W_Theta[k]*W_Theta[l]*W_Phi[p]*W_Phi[q];

//QC method for integral;
I += Weights*Sphere_Function(r[i],r[j],Theta[k],Theta[l],Phi[p],Phi[q]);
}}}}}}
```

ii. *Monte Carlo*

ii.1. *Brute Force*

The brute force method (BF) is a known approach used in MC integration [4], where one does not take any considerations concerning the specific integral one is trying to solve. One simply uses the uniform probability distribution to generate the random numbers needed to do the calculation. As specified earlier, the integral we are trying to solve is on the form

$$\begin{aligned} I &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(\boldsymbol{r_1},\boldsymbol{r_2})d\boldsymbol{r_1}d\boldsymbol{r_2}, \qquad f(\boldsymbol{r_1},\boldsymbol{r_2}) = e^{-4(r_1+r_2)}\frac{1}{|\boldsymbol{r_1}-\boldsymbol{r_2}|} \\ &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} e^{-4(r_1+r_2)}\frac{1}{|\boldsymbol{r_1}-\boldsymbol{r_2}|}d\boldsymbol{r_1}d\boldsymbol{r_2} \\ &\approx \int_{a}^{b}\int_{a}^{b} e^{-4(r_1+r_2)}\frac{1}{|\boldsymbol{r_1}-\boldsymbol{r_2}|}d\boldsymbol{r_1}d\boldsymbol{r_2} \end{aligned}$$

where we have approximated the infinite integration limits, $[-\infty,\infty]$ to finite limits $[a,b]$

Thus, the integral can then be approximated to:

$$\begin{aligned} I &\approx (b-a)^6\langle f\rangle \\ &= (b-a)^6\frac{1}{N}\sum_{i=1}^{N} f(\boldsymbol{r_{1_i}},\boldsymbol{r_{2_i}}) \end{aligned}$$

with $\boldsymbol{r_{1_i}}$ and $\boldsymbol{r_{2_i}}$ being random numbers uniformly distributed in $[a,b]$.

The random numbers $\boldsymbol{r_{1_i}}$ and $\boldsymbol{r_{2_i}}$ are implemented by using a random number generator (RNG) that distributes the values in the interval we are integrating over. In this project we chose the Mersenne Twister pseudorandom number generator (mt19937_64)[1] found in the <random> library of C++ due to its high sample size.

The pseudocode below is an illustration on how we implemented the BF method with minor changes made here to make it easier to understand. All the approximations discussed above are included our program.

Listing 3: C++ Multidimensional Monte Carlo: Brute Force Integration

```
/*
Pseudocode for calculating integral using BF MC.
*/

RNG; // Random number generator

for (int i=1; i <= N; i++){
// p(x) is the uniform distribution

    x1 = RNG*(b-a)+a; // transformation to [a,b]
    x2 = RNG*(b-a)+a; // transformation to [a,b]
    y1 = RNG*(b-a)+a; // transformation to [a,b]
    y2 = RNG*(b-a)+a; // transformation to [a,b]
    z1 = RNG*(b-a)+a; // transformation to [a,b]
    z2 = RNG*(b-a)+a; // transformation to [a,b]

//  Function takes in the variables x1,x2,y1,y2,z1,z2 and evaluates them in each
    point which is distributed by the uniform distribution p(x)

    // Calculating the average of the function
    Integrand += Function(x1,x2,y1,y2,z1,z2);
    }

Integrand = (Integrand/ N);
Volume = pow((b-a),6); // Volume of the integral
Integral = Integrand*Volume; // Calculating the integral
```

ii.2. *Importance Sampling*

Importance sampling (IS) is another well known approach used in MC integration. If a probability density function (PDF) in an interval $[a, b]$ resembles the behavior of the function to be integrated, the integral can be simplified with this distribution. This results in random values that fit the function better than the uniform distribution used in BF. From Appendix B subsection Importance Sampling, we know that the integral can be approximated into

$$I = \int_0^\infty dr_1 \int_0^\infty dr_2 \int_0^\pi d\theta_1 \int_0^\pi d\theta_2 \int_0^{2\pi} d\phi_1 \int_0^{2\pi} d\phi_2 \frac{e^{-4(r_1+r_2)} {r_1}^2 {r_2}^2 \sin\theta_1 \sin\theta_2}{\sqrt{{r_1}^2 + {r_2}^2 - 2r_1 r_2 \cos\beta}}$$
$$\approx \frac{4\pi^4}{16} \frac{1}{N} \sum_{i=1}^{N} f(r_{1_i}, r_{2_i}, \theta_{1_i}, \theta_{2_i}, \phi_{1_i}, \phi_{2_i})$$

with the radial parts r_{1_i} and r_{2_i} being random numbers distributed by the exponential distribution in the interval $[0, \infty)$. The angular parts θ_{1_i} and θ_{2_i} are random numbers uniformly distributed in $[0, \pi]$, while ϕ_{1_i} and ϕ_{2_i} are random numbers uniformly distributed in $[0, 2\pi]$.

[1] http://www.cplusplus.com/reference/random/mt19937_64/

The random numbers $r_{1_i}, r_{2_i}, \theta_{1_i}, \theta_{2_i}, \phi_{1_i}$ and ϕ_{2_i} are implemented by using a RNG (mt19937_64) that distributes the values in the interval we are integrating over.

Pseudocode shown below contains minor changes compared to how we implemented the IS method in order to make it easier to understand here. It illustrates how the method was implemented and the approximations discussed above are also included in our program.

Listing 4: C++ Multidimensional Monte Carlo: Importance Sampling

```
/*
Pseudocode for calculating integral using IS Monte Carlo.
*/

RNG; // Random number generator

for (int i=1; i <= N; i++){


// Looping over the radial parts, r_1 r_2 following the exponential distribution

    y = RNG;
    r1 = -lambda*log(1. - y);
    r2 = -lambda*log(1. - y);

// Looping over the angular parts theta and phi following the uniform distribution

    theta1 = PI*RNG;
    theta2 = PI*RNG;

    phi1 = 2*PI*RNG;
    phi2 = 2*PI*RNG;


// Spherical_Function takes in the variables (r1, r2, theta1, theta2, phi1, phi2)
    and evaluates them in each point

    Integrand += Spherical_Function(r1, r2, theta1, theta2, phi1, phi2); //
        Calculating the average of the function

    }

Integrand = Integrand/N;

Volume = 4*pow(PI,4)/16 // Volume of the integral

Integral = Volume*Integrand; // Calculating the integral
```

iii. *Avoiding division by zero*

Evaluating the integral for this function can cause problems when two almost alike numbers are being subtracted from each other. In this case the problem occurs due to the denominator, where this subtraction can lead to values near zero and thus division by zero leading to inaccurate results. This is a problem regardless of coordinate system used and in order to avoid this we set a tolerance for the denominator. When the denominator approaches a value near zero it is then neglected. A snippet of the code showing how we are avoiding this division is shown below for the function with cartesian coordinates,

Listing 5: C++ Evaluation of the function for cartesian coordinates

```
double Cartesian_Function(double x_1, double y_1, double z_1, double x_2, double y_2
    , double z_2){
    // Evaluating the denominator
    double D = (x_1-x_2)*(x_1-x_2)+(y_1-y_2)*(y_1-y_2)+(z_1-z_2)*(z_1-z_2);

    // Excluding values near zero to avoid zero division
    if (D > ZERO){
        // Evaluating different terms of the exponential
        double Denominator = sqrt(D);
        double r_1 = sqrt((x_1*x_1)+(y_1*y_1)+(z_1*z_1));
        double r_2 = sqrt((x_2*x_2)+(y_2*y_2)+(z_2*z_2));

        double Exponent = exp(-4*(r_1+r_2));    // Resulting exponent

        double Function = Exponent/Denominator;
        return Function;
    }
    else{

        return 0;
    }
}
```

IV. Numerical results

Numerical results is being presented in this section. Both GQ-Legendre and GQ-Laguerre results is shown during the next subsection, some of the results given is the approximated solution to the integral that we are evaluating, relative error for both methods and the CPU-time. The next subsection will dive into the Monte Carlo method, results presented there will be approximated solution of the integral, relative error, standard deviation and also the effect that parallelization has on the CPU-time. When evaluating the integral in equation 2 with Legendre polynomials we need to approximate infinity with some λ. We say that we can approximate infinity when the probability that the electron is located at a point away from the core goes to zero. A approximation of the three dimensional case as a function of two variables is shown and plotted below.

$$|\psi(x,y)|^2 = e^{-4\left(\sqrt{x^2+y^2}\right)} \tag{18}$$

As we can see from figure 1, $|\psi(x,y)|^2$ rapidly goes to zero as the distance from the nucleus increases. Therefore appropriate limits of [-3,3] is chosen. This limit will be used when presenting Gaussian-Legendre quadrature results and discussion and also Monte Carlo BF.

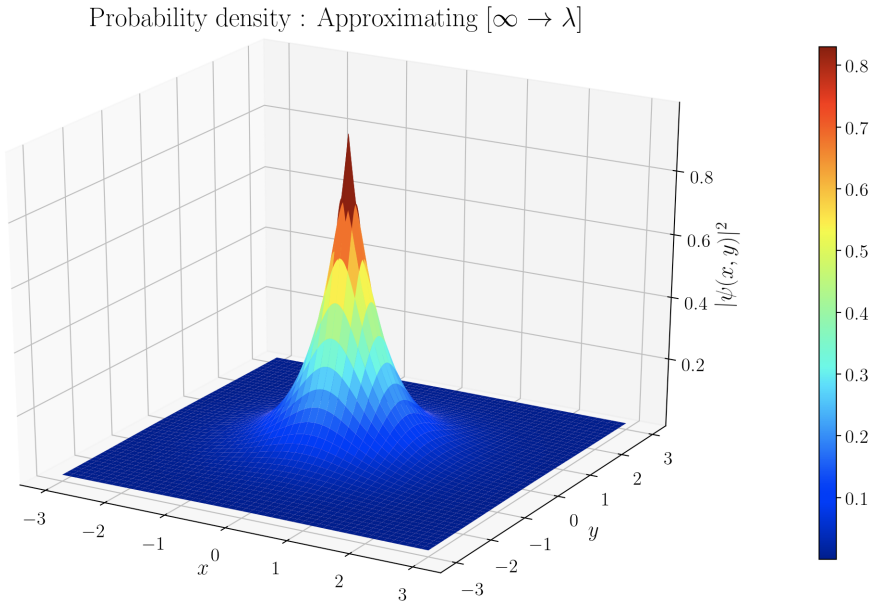
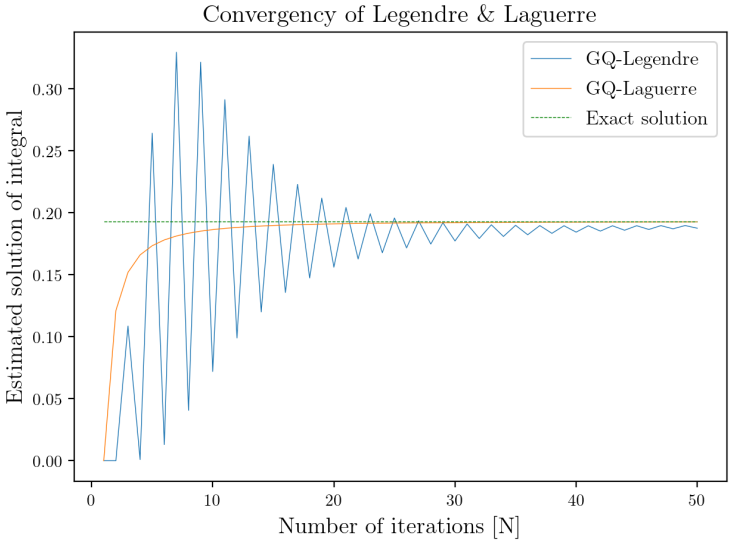


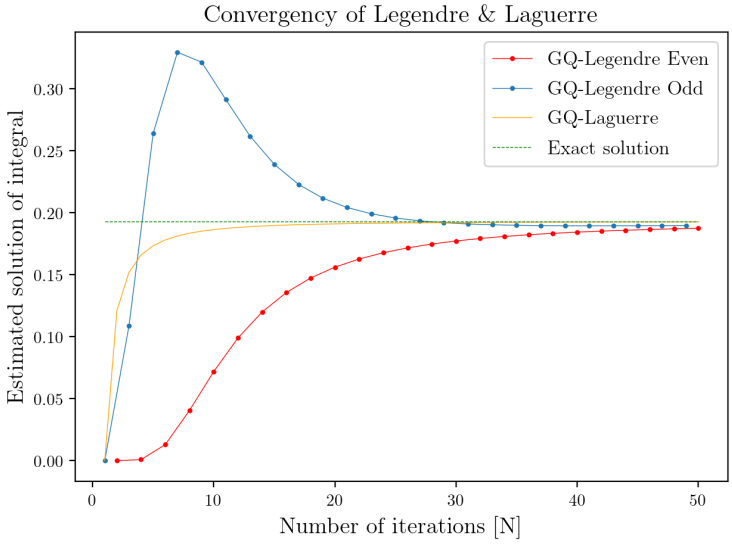
Figure 1: 3D-plot of the probability density, as a function of position r_i. λ set to [-3,3]

i. *Gaussian Quadrature*

Figure 2 shows two plots on how the two methods converges to the exact answer. The exact answer of the given integral is derived in Appendix A. Figure 2a), shows the convergence of both Legendre and Laguerre GQ for increasing number of iterations N. Where we do not make a difference of odd and even numbers for Legendre. We can clearly see that GQ-Laguerre converge more rapidly towards the exact answer than GQ-Legendre, and that GQ-Legendre has jumps back and forth between overestimating and underestimating the exact answer for N up to 50. In figure 2b) we dissect the Legendre solutions into two parts, odd and even. Where the odd part is blue and the even part is red. We can see by the plot that the odd part converges more quickly towards the exact answer than the even part.

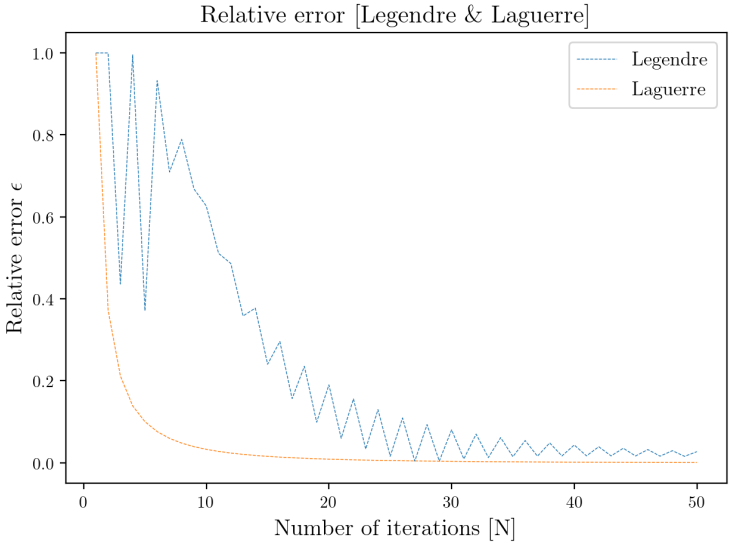


(a) Plot of integral solution with increasing N. Both Legendre & Laguerre polynomials is being used.

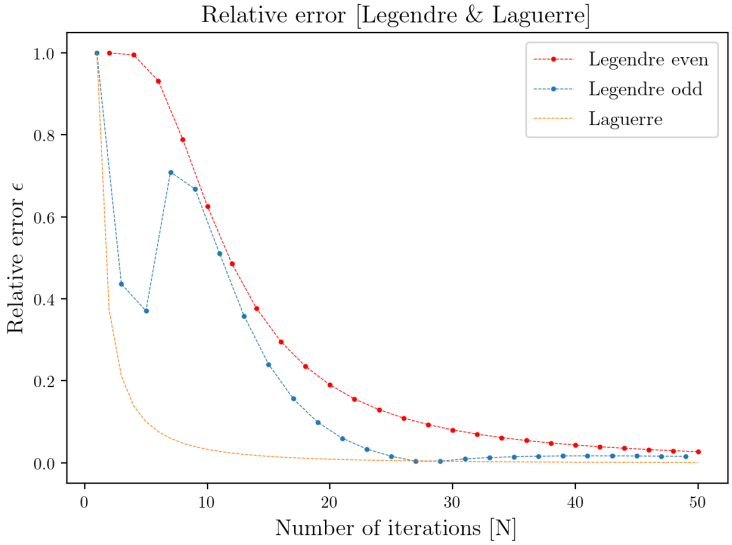


(b) Laguerre and exact remains the same as figure (a), Legendre is divided into an even and odd part.

Figure 2: (a) and (b) shows how the Gaussian Quadrature with Legendre and Laguerre polynomials is converging to the exact answer as number of iterations N increase. (b) divides the solutions given by Legendre into an even and odd part. Important to note that λ is set to [-3,3]



(a) Plot of relative error for both Legendre & Laguerre, for N up to 50



(b) Laguerre remains the same as figure (a), Legendre is divided into an even and odd part.

Figure 3: a) and b) shows the relative error as a function of N. b) Legendre is dived into even and odd part, where the red plot shows how the relative error goes for even numbers of N and blue plot for odd numbers of N.

Figure 3 displays how the relative error progress as N is increased. In figure 3a) we look at both Legendre and Laguerre, we clearly see that the relative error rapidly goes to zero for Laguerre. For Legendre we see some abnormal spikes as we did in figure 2a). These spikes gives us an unusual behavior for the relative error. In figure 3b) we have as in figure 2b) divided the Legendre into an even and odd part. We can more clearly see the behavior this way.

In table 1 we have chosen to present values of the approximated integrals for both Legendre and Laguerre as well as the relative error, as displayed in figure 2 & 3. Values ranging from [5,50]N with interval of 5. As we can see, the relative error for Laguerre reaches a four decimals precision for $N \geq 45$. For Legendre we can see that the relative error jumps up and down as shown in 3a), and the relative error is smallest for odd numbers.

Table 1: Legendre & Laguerre values for number of iterations N = [0,50], with interval of 5, and relative error as measured with the exact value $\frac{5\pi^2}{16^2}$.

N	**Legendre**	**Relative error ϵ**	**Laguerre**	**Relative error ϵ**
5	0.2643	0.3708	0.1735	0.1002
10	0.0720	0.6266	0.1865	0.0327
15	0.2391	0.2403	0.1898	0.0156
20	0.1561	0.1900	0.1911	0.0087
25	0.1958	0.0158	0.1917	0.0053
30	0.1773	0.0803	0.1921	0.0034
35	0.1899	0.0148	0.1923	0.0022
40	0.1844	0.0433	0.1925	0.0014
45	0.1896	0.0165	0.1926	0.0009
50	0.1876	0.0270	0.1927	0.0005

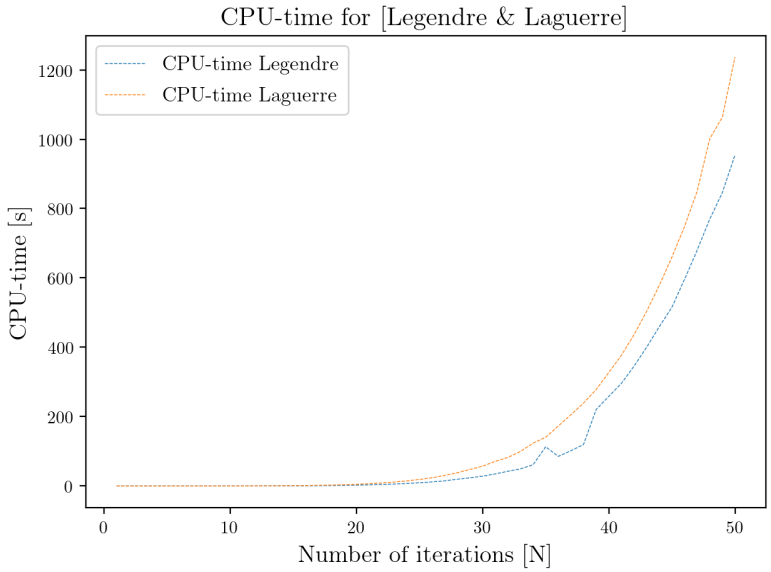


Figure 4: CPU-time as a function of N for both Legendre & Laguerre. An exponential behavior as N is increasing for both methods is observed. CPU-time messured in seconds.

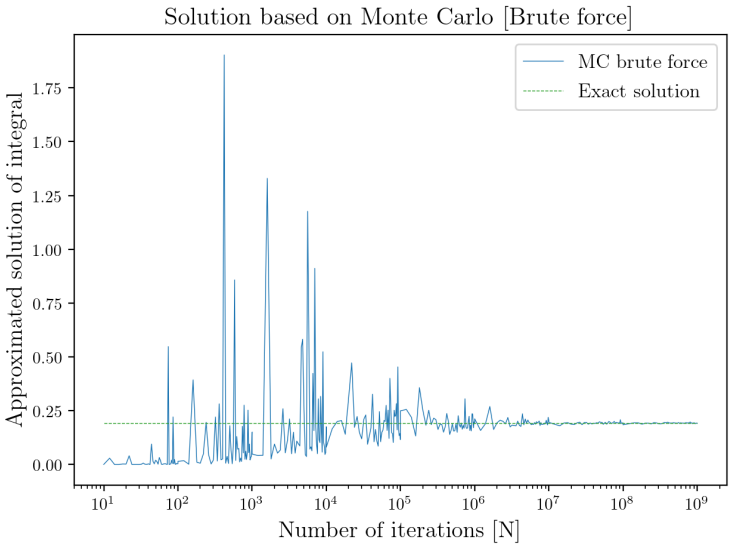
Table 2: Legendre & Laguerre CPU-time for N = [0,50] with interval of 5. CPU-time measured only over integration-loops.

CPU-time [s]		
N	**Legendre**	**Laguerre**
5	0.001	0.003
10	0.039	0.081
15	0.457	0.925
20	2.447	4.907
25	9.311	18.834
30	28.231	57.106
35	113.105	140.532
40	258.274	327.090
45	515.142	659.291
50	954.434	1236.827

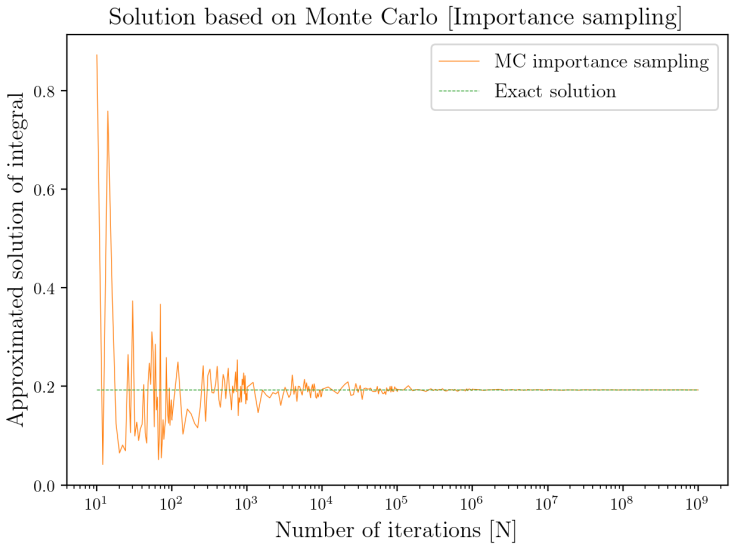
The last results presented for GQ is the CPU-time for both Legendre and Laguerre. Figure 4 shows a plot of the CPU-time as a function of N. In addition, for low N we see in table 2 that the difference between the two method is minimal, in the order of 10^{-3} seconds. We can see that there is an increasing difference in CPU-time between Legendre and Laguerre for $N \geq 25$. There is an exponential behavior for both methods, which reaches values of roughly 950 and 1240 seconds at N = 50 for Legendre and Laguerre, respectively.

ii. *Monte Carlo*

Introducing the results of MC is displayed throughout this subsection. In Figure 5, we see two plots of BF and IS. Figure 5 is an approximated solution of our integral in equation 2.



(a) Convergency of Monte Carlo brute force, with increasing N.



(b) Convergency of Monte Carlo importance sampling, with increasing N.

Figure 5: a)Plot of MC-solution based on BF, starting to converge for for high N. b) Plot of MC-solution based on IS, starting to converge for lower N than BF. 370 sampling points.

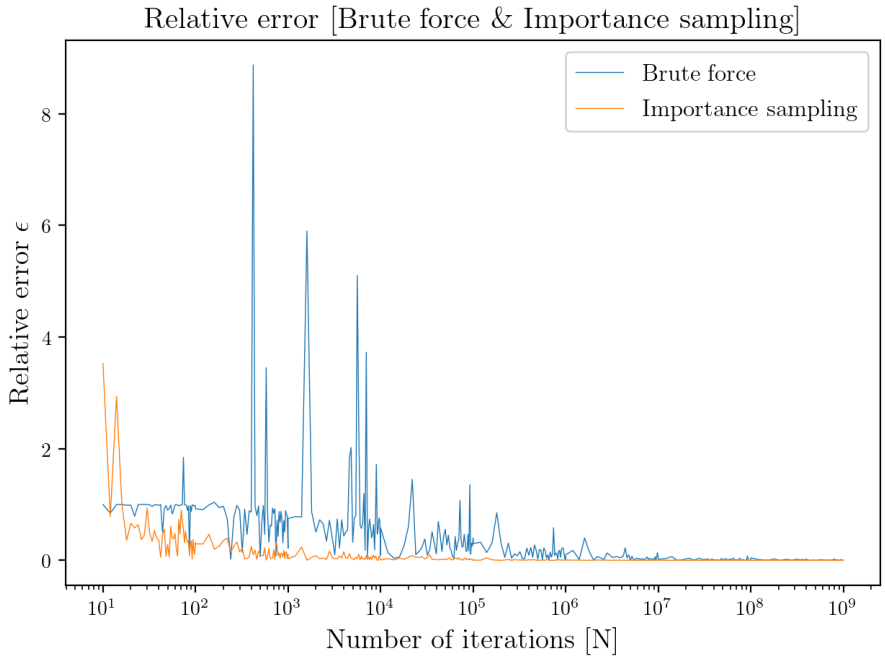


Figure 6: Plot of relative error for both brute force & importance sampling. Plotted with N increased up to 10^9. Also with 370 sampling points in the interval $[10, 10^9]$.

The main differences in-between figure 5a) and 5b) is the methods, and the way they converges towards the exact answer. IS converges more rapidly towards the exact answer, than BF. It becomes clear that both methods converges to a relatively exact answer as N is in the order of 10^9.

Figure 6 shows a plot over relative error as a function of N. We can easily see that the relative error goes more rapidly towards zero as N increases fir SI than BF. This shows the clear advantage the IS has over the BF-method.

Table 3 takes into account approximated solution of the integral for both methods as well as relative error [ϵ] and standard deviation [σ]. We can see that the relative error [IS] is approximately zero for $N = 10^9$. Where $\epsilon_{SI} = 1.6 \times 10^{-5}$, for $N = 10^9$. BF-method is also fairly accurate for high N, but fails considerably more for low values of N.

Table 3: Values of Brute force & Importance sampling for N up to 10^9. Relative error & Standard deviation is denoted as RE & STD respectively.

N	**BF**	**RE ϵ**	**STD σ**	**IS**	**RE ϵ**	**STD σ**
10^3	0.1509	0.2172	0.0067	0.1724	0.1057	0.0436
10^4	0.0808	0.5808	0.0808	0.1921	0.0033	0.0081
10^5	0.1158	0.3995	0.0319	0.1939	0.0060	0.0034
10^6	0.1900	0.0143	0.0177	0.1939	0.006	0.0010
10^7	0.1961	0.0171	0.0178	0.1925	0.0017	0.0003
10^8	0.1934	0.0033	0.0030	0.1927	0.0004	0.0001
10^9	0.1917	0.0057	0.0011	0.1928	1.6E-05	3.3E-05

Comparing CPU-times for the two methods of interest is not the most interesting thing. We see by figure 7 that both BF and IS overlaps relatively well, however, if we parallelize our code we see a clear decrease in time consumption. Table 4 displays various CPU-times for 1, 2 and 4 CPUs. We can clearly see a decrease in time as number of CPUs is increased[2].

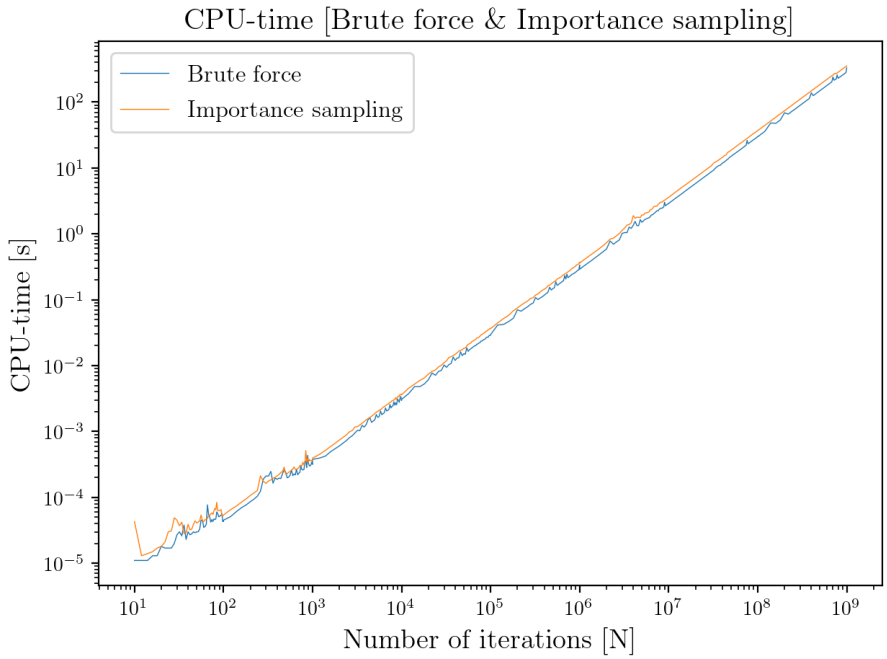


Figure 7: CPU-time for both brute force & importance sampling.

Table 4: Parallel computing of Monte Carlo brute force and importance sampling, without vectorization. Comparing CPU-time [s] for 1, 2 and 4 CPUs, for N = $[10^4, 10^9]$

CPU-time [s]						
#CPU	1 CPU		2 CPU		4 CPU	
N	**BF**	**IS**	**BF**	**IS**	**BF**	**IS**
10^4	0.003	0.004	0.002	0.003	0.001	0.001
10^5	0.030	0.037	0.015	0.019	0.009	0.012
10^6	0.336	0.372	0.152	0.192	0.086	0.095
10^7	2.902	3.580	1.727	1.904	0.757	0.980
10^8	29.857	37.005	14.664	18.930	8.630	9.764
10^9	330.617	355.212	146.609	186.959	82.327	109.022

[2]Important to be aware of the fact that our code is not vectorized.

V. Discussion

i. *Gaussian Quadrature*

Table 1 shows unsatisfactory results for GQ-Legendre which is clearly illustrated in Figure 2a, where we observe that the results are unstable and fluctuating with low numerical precision, resulting in a high relative eorror. It is also observable in Figure 2b that GQ-Legendre is dependent on if the Legendre polynomial is odd or even, where the even Legendre polynomials converge faster to the exact answer compared to the odd Legendre polynomials. We do not know what causes this effect, but we think it lies in the approximation of our initial function, and the fact that only odd polynomials can be approximated exact using Legendre.

Since we are solving equation 2 in Cartesian coordinates when we are doing GQ-Legendre, we are dealing with a function that is not easily approximated with a polynomial. Because of this, the weights we use in the GQ summation are not satisfactory and results in a bad approximation of the integral. Using a higher degree Legendre polynomial than 50 and expanding the integration limits to go beyond $[-3, 3]$ would lead to better weights and then precision, but the balance between numerical precision and computational time would make this not suitable.

GQ-Laguerre is a much more stable and accurate method compared to GQ-Legendre as shown in table 1, where we see a lower relative error for the same degree polynomial and also a very high numerical precision for high N. In figure 3, it is shown that the GQ-Laguerre can approximate the function with a lower degree polynomial than GQ-Legendre before it converges. After $N = 30$, GQ-Laguerre reproduces the analytical result with 3 leading digits after the decimal point. The reason for a more rapidly convergence for Laguerre is due to the fact that when we are converting our integral to spherical coordinates and extract the exponential out of the function, the resulting function is more easily approximated as a polynomial. The weights we then calculate using GQ-Laguerre on the radial parts and GQ-Legendre on the angular parts gives a far better approximation resulting in the better converged method as shown in figure 2.

ii. *Monte Carlo*

The results presented in table 3 obtained by Monte Carlo integration seem to converge to the exact answer and the standard deviation seems to decrese when the number of iterations N is increased. These results corresponds to the theory of Monte Carlo as presented above. Monte Carlo BF shown in figure 5a) gives an unpredictable approximation for low N, and clearly converge towards a better approximation as N is large. This is mostly due to our choose of PDF. Where we have a uniform distribution which is not suitable for a Gaussian-like function. Yielding a bad approximation of the integral for low N. Furthermore we can see by figure 6 that the relative error for BF is unpredictable for N up to 10^6, and converge towards zero as N is large enough.

Furthermore, the standard deviation of importance sampling is significantly lower compared to the BF. The cause for this huge difference in accuracy between the methods is due to the fact of variable change from cartesian into spherical coordinates and that the exponential factor in the integrand was incorporated into the PDF, which simplified the numerical calculation. The new and improved PDF therefore gives a more accurate approximation of the integral.

Figure 7 shows the CPU-time for both Monte Carlo methods and there do not seem to be a notable difference between them. Moreover, table 4 shows the CPU-time for both the parallel and

non-parallel implementation, and we notice a considerably speed up concerning the CPU-time. As one would expect, running the calculations on four CPU's gives considerably faster calculations compared to one and two CPU's. This speed up is due to dividing the calculation in-between multiple CPU's instead of one.

VI. Conclusion

Gaussian Quadrature method are a set of numerical methods that can be used when evaluating integrals occurring in science. Depending of the form of the function that is to be integrated and its integration limits, the integral can be tailored to the appropriate orthogonal polynomial. It will require high computational power if a high accuracy is needed and the integral should also not have too many dimensions. For our problem were GQ-Laguerre superior to GQ-Legendre, due to the weight function $W(x) = x^{\alpha}e^{-x}$ of Laguerre polynomials which naturally occurs in the correlation energy.

Although GQ as a method gave us a very good numerical accuracy for the problem we looked at, will it still be dependent on how well a function can be approximated by a orthogonal polynomial. The cost of computational power when for example doing multiple correlation energies for a many-particle system is too high, and for quantum mechanical calculations (or any integration calculation for that matter) that may need an exceptionally high precision, the Gaussian quadrature in general are maybe not that suitable.

Monte Carlo integration gives by far the best numerical values.

Appendices

A. Repulsion Integral

Calculating the electron-electron repulsion analytically can be done in multiple ways, but as it stands now it is unsolvable, with practically infinitely many singularities, one for every position in the universe where both electrons could be simultaneously. A neat way of solving this problem would be to expand $\frac{1}{|\boldsymbol{r_1}-\boldsymbol{r_2}|}$ in terms of the spherical harmonics, utilizing the orthogonality properties of the associated Legendre polynomials to simplify the problem. At our level of academic education however, we have chosen to tackle this problem by changing coordinate system from Cartesian coordinates to spherical coordinates, with the polar axis pointed at the first electron. By choosing this new coordinate system we can rewrite the denominator to an easily integratable form. Fixing the polar axis to point at the first electron sets:
$\theta_1 = 0 \implies x_1 = r_1 \sin\theta_1 \cos\theta_1 = 0$, $y_1 = r_1 \sin\theta_1 \sin\phi_1 = 0$ and $z_1 = r_1 \cos\theta_1 = r_1$.

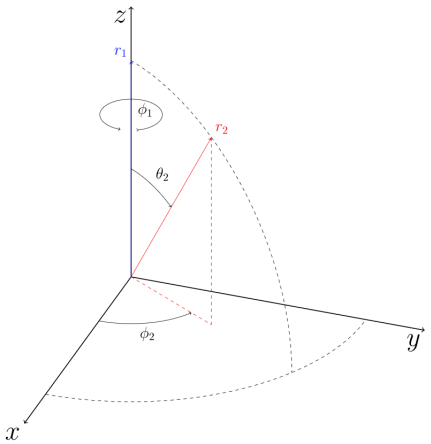


Figure 8:
Spherical coordinate system, with r_1 always pointing at the polar axis, used for easier calculation.

$$
\begin{aligned}
|\boldsymbol{r_1}-\boldsymbol{r_2}| &= \sqrt{(x_1-x_2)^2+(y_1-y_2)^2+(z_1-z_2)^2} \\
&= \sqrt{(-r_2\sin\theta_2\cos\phi_2)^2+(-r_2\sin\theta_2\sin\phi_2)^2+(r_1-r_2\cos\theta_2)^2} \\
&= \sqrt{{r_2}^2\sin^2\theta_2\cos^2\phi_2+{r_2}^2\sin^2\theta_2\sin^2\phi_2+{r_1}^2-2r_1r_2\cos\theta_2+{r_2}^2\cos^2\theta_2} \\
&= \sqrt{{r_1}^2+{r_2}^2\underbrace{\Big(\sin^2\theta_2\big(\cos^2\phi_2+\sin^2\phi_2\big)+\cos^2\theta_2\Big)}_{=1}-2r_1r_2\cos\theta_2} \\
&= \sqrt{{r_1}^2+{r_2}^2-2r_1r_2\cos\theta_2}
\end{aligned}
$$

The integral in this new coordinate system is now on spherical form, where the denominator only depends on the second electrons angle θ relative to the position of the first electron, making it possible to get an analytical expression for the expectation value of the repulsion integral.

$$
\left\langle \frac{1}{|\boldsymbol{r_1}-\boldsymbol{r_2}|} \right\rangle = \int_0^\infty dr_1 \int_0^\infty dr_2 \int_0^\pi d\theta_1 \int_0^\pi d\theta_2 \int_0^{2\pi} d\phi_1 \int_0^{2\pi} d\phi_2 \frac{e^{-4(r_1+r_2)}{r_1}^2{r_2}^2\sin\theta_1\sin\theta_2}{\sqrt{{r_1}^2+{r_2}^2-2r_1r_2\cos\theta_2}}
$$

Noticing that this integral does not depend on many of the integration variables, three of these integrals can be solved immediately with no problems:

$$I_{1-3} = \int_0^{\pi} \sin\theta_1 d\theta_1 \int_0^{2\pi} d\phi_1 \int_0^{2\pi} d\phi_2 = 2 \cdot 2\pi \cdot 2\pi = 8\pi^2$$

Now there are 3 more integrals to be solved, the integral over θ_2 and the remaining integrals over r_1 and r_2. There is one problem though, and that is distinguishing between when the first particle is closer to the core than the other, and when it is farther away from it. As will be explained in a bit, this relation falls out from the solution after solving for θ_2, which is why we will start with this one.

$$I_4 = \int_0^{\pi} \frac{\sin\theta_2}{\sqrt{{r_1}^2 + {r_2}^2 - 2r_1 r_2\cos\theta_2}} d\theta_2 = \frac{1}{2r_1 r_2} \int_{{r_1}^2+{r_2}^2-2r_1r_2}^{{r_1}^2+{r_2}^2+2r_1r_2} \frac{1}{\sqrt{u}} du = \Big[\frac{1}{2r_1r_2} \cdot 2\sqrt{u}\Big]_{{r_1}^2+{r_2}^2-2r_1r_2}^{{r_1}^2+{r_2}^2+2r_1r_2}$$

$$= \frac{\sqrt{{r_1}^2 + {r_2}^2 + 2r_1r_2} - \sqrt{{r_1}^2 + {r_2}^2 - 2r_1r_2}}{r_1r_2} = \frac{\sqrt{(r_1+r_2)^2} - \sqrt{(r_1-r_2)^2}}{r_1r_2}$$

$$= \frac{|r_1+r_2| - |r_1-r_2|}{r_1r_2} = \begin{cases} \frac{|r_1+r_2|-|r_2-r_1|}{r_1r_2} = 2/r_2, & r_1 < r_2 \\ \frac{|r_1+r_2|-|r_1-r_2|}{r_1r_2} = 2/r_1, & r_1 > r_2 \end{cases}$$

The reason for why the last term above is split for whenever r_1 is smaller or larger, comes from the term where you take the square root of the difference between the two vectors squared. To ensure real values and not complex, the choice of taking the largest radial length and subtracting the lower splits this equation. From this previous calculation, the relation between the radial length of the electrons, will then be taken care of by splitting the integral from 0 to ∞ into two parts.

$$I_5 = \int_0^{\infty} {r_1}^2{r_2}^2 e^{-4(r_1+r_2)} \cdot \left(\frac{|r_1+r_2| - |r_1-r_2|}{r_1r_2}\right) dr_2 = 2{r_1}^2 e^{-4r_1}\left(\int_0^{r_1} \frac{{r_2}^2 e^{-4r_2}}{r_1} dr_2 + \int_{r_1}^{\infty} r_2 e^{-4r_2} dr_2\right)$$

$$= 2{r_1}^2 e^{-4r_1}\left(\Big[-\frac{(8{r_2}^2 + 4r_2 + 1)e^{-4r_2}}{32r_1}\Big]_0^{r_1} + \Big[-\frac{(4r_2+1)e^{-4r_2}}{16}\Big]_{r_1}^{\infty}\right)$$

$$= 2{r_1}^2 e^{-4r_1}\left(-\frac{(8{r_1}^2 + 4r_1 + 1)e^{-4r_1}}{32r_1} + \frac{1}{32r_1} + 0 + \frac{(4r_1+1)e^{-4r_1}}{16}\right)$$

$$= 2{r_1}^2 e^{-4r_1}\left(-\frac{8r_1e^{-4r_1}}{32} - \frac{4e^{-4r_1}}{32} - \frac{e^{-4r_1}}{32r_1} + \frac{1}{32r_1} + \frac{4r_1e^{-4r_1}}{16} + \frac{e^{-4r_1}}{16}\right)$$

$$= r_1 e^{-4r_1}\left(\cancel{-\frac{16{r_1}^2e^{-4r_1}}{32}} - \frac{8r_1e^{-4r_1}}{32} - \frac{2e^{-4r_1}}{32} + \frac{2}{32} + \cancel{\frac{16{r_1}^2e^{-4r_1}}{32}} + \frac{4r_1e^{-4r_1}}{32}\right)$$

$$= \frac{e^{-4r_1}}{32}\left(2r_1 - 4{r_1}^2e^{-4r_1} - 2r_1e^{-4r_1}\right)$$

The two integrals that was solved in the previous section, are two very common integrals that comes up in physics all the time, especially quantum physics, so we do not explicitly show how to solve them. These two integrals are $\int_a^b x^2 e^{-\alpha x} dx = [-\frac{(\alpha^2 x^2 + 2\alpha x + 2)e^{-\alpha x}}{\alpha^3}]_a^b$ and $\int_a^b x e^{-\alpha x} dx = [-\frac{(\alpha x + 1)e^{-\alpha x}}{\alpha^2}]_a^b$. What remains is now the final integral over r_1 and multiplying the result with the factors from I_{1-3}. Since the problem with what electron is closest to the core is already taken care for, this integral is pretty straight forward and will be solved from 0 to ∞ as it stands. The same integrals as in I_5 will be used here also, but with different constants.

$$I_6 = \int_0^\infty \frac{e^{-4r_1}}{32}\left(2r_1 - 4{r_1}^2 e^{-4r_1} - 2r_1 e^{-4r_1}\right) dr_1 = \int_0^\infty \left(\frac{r_1 e^{-4r_1}}{16} - \frac{{r_1}^2 e^{-8r_1}}{8} - \frac{r_1 e^{-8r_1}}{16}\right) dr_1$$

$$= \frac{1}{16}\Big[-\frac{(4r_1+1)e^{-4r_1}}{16}\Big]_0^\infty - \frac{1}{8}\Big[-\frac{(64r_1^2+16r_1+2)e^{-8r_1}}{512}\Big]_0^\infty - \frac{1}{16}\Big[-\frac{(8r_1+1)e^{-8r_1}}{64}\Big]_0^\infty$$

$$= \frac{1}{16}\cdot\frac{1}{16} - \frac{1}{8}\cdot\frac{2}{512} - \frac{1}{16}\cdot\frac{1}{64} = \frac{5}{2048}$$

$$\left\langle \frac{1}{|\boldsymbol{r_1} - \boldsymbol{r_2}|} \right\rangle = I_{1-3}\cdot I_6 = 8\pi^2 \cdot \frac{5}{2048} = \underline{\underline{\frac{5\pi^2}{256}}}$$

B. Spherical Coordinates

In appendix A, a specialized spherical coordinate system is used. For our numerical integration we will use the general spherical coordinate system, which will change our denominator once again.

$$
\begin{aligned}
|\boldsymbol{r_1}-\boldsymbol{r_2}| &= \sqrt{(x_1-x_2)^2+(y_1-y_2)^2+(z_1-z_2)^2} \\
&= \sqrt{(r_1\sin\theta_1\cos\phi_1 - r_2\sin\theta_2\cos\phi_2)^2 + (r_1\sin\theta_1\sin\phi_1 - r_2\sin\theta_2\sin\phi_1)^2 + (r_1\cos\theta_1 - r_2\cos\theta_2)^2} \\
&= \sqrt{\begin{aligned}
&{r_1}^2\sin^2\theta_1\cos^2\phi_1 - 2r_1r_2\sin\theta_1\cos\phi_1\sin\theta_2\cos\phi_2 + {r_2}^2\sin^2\theta_2\cos^2\phi_2 \\
&+ {r_1}^2\sin^2\theta_1\sin^2\phi_1 - 2r_1r_2\sin\theta_1\sin\phi_1\sin\theta_2\sin\phi_2 + {r_2}^2\sin^2\theta_2\sin^2\phi_2 \\
&+ {r_1}^2\cos^2\theta_1 - 2r_1r_2\cos\theta_1\cos\theta_2 + {r_2}^2\cos^2\theta_2
\end{aligned}} \\
&= \sqrt{\begin{aligned}
&{r_1}^2\underbrace{\big(\sin^2\theta_1(\cos^2\phi_1+\sin^2\phi_1)+\cos^2\theta_1\big)}_{=1} + {r_2}^2\underbrace{\big(\sin^2\theta_2(\cos^2\phi_2+\sin^2\phi_2)+\cos^2\theta_2\big)}_{=1} \\
&- 2r_1r_2\big(\sin\theta_1\sin\theta_2(\cos\phi_1\cos\phi_2+\sin\phi_1\sin\phi_2)+\cos\theta_1\cos\theta_2\big)
\end{aligned}} \\
&= \sqrt{{r_1}^2+{r_2}^2-2r_1r_2\big(\cos\theta_1\cos\theta_2+\sin\theta_1\sin\theta_2\cos(\phi_1-\phi_2)\big)} \\
&= \sqrt{{r_1}^2+{r_2}^2-2r_1r_2\cos\beta}
\end{aligned}
$$

$$
\left\langle\frac{1}{|\boldsymbol{r_1}-\boldsymbol{r_2}|}\right\rangle = \int_0^\infty dr_1\int_0^\infty dr_2\int_0^\pi d\theta_1\int_0^\pi d\theta_2\int_0^{2\pi}d\phi_1\int_0^{2\pi}d\phi_2\frac{e^{-4(r_1+r_2)}{r_1}^2{r_2}^2\sin\theta_1\sin\theta_2}{\sqrt{{r_1}^2+{r_2}^2-2r_1r_2\cos\beta}}
$$

$$
\cos\beta \equiv \big(\cos\theta_1\cos\theta_2+\sin\theta_1\sin\theta_2\cos(\phi_1-\phi_2)\big)
$$

i. *Gaussian Quadrature Laguerre*

The GC-Laguerre method is applied to integrals on the form:

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

To make our integral similar as the expression above, we will need to do a variable change where

$$
\begin{aligned}
u_1 = 4r_1 \quad \& \quad u_2 = 4r_2 \\
du_1 = 4dr_1 \quad \& \quad du_2 = 4dr_2
\end{aligned}
$$

Then if we substitute variables the integral can be transformed into

$$I = \int_0^\infty \frac{du_1}{4} \int_0^\infty \frac{du_2}{4} \int_0^\pi d\theta_1 \int_0^\pi d\theta_2 \int_0^{2\pi} d\phi_1 \int_0^{2\pi} d\phi_2 \frac{e^{-(u_1+u_2)}(\frac{u_1}{4})^2(\frac{u_2}{4})^2 \sin\theta_1 \sin\theta_2}{\sqrt{(\frac{u_1}{4})^2 + (\frac{u_2}{4})^2 - 2(\frac{u_1}{4})(\frac{u_2}{4})\cos\beta}}$$
$$= \frac{1}{4^5} \int_0^\infty du_1 \int_0^\infty du_2 \int_0^\pi d\theta_1 \int_0^\pi d\theta_2 \int_0^{2\pi} d\phi_1 \int_0^{2\pi} d\phi_2 \frac{e^{-(u_1+u_2)} {u_1}^2 {u_2}^2 \sin\theta_1 \sin\theta_2}{\sqrt{{u_1}^2 + {u_2}^2 - 2u_1u_2\cos\beta}}$$

We will only implement GQ-Laguerre on the radial part of the integral, as for the angular parts we will use the GQ-Legendre with a mapping of the integration limits.

ii. *Monte Carlo Importance Sampling*

Importance sampling is a method where one uses a PDF that is centered around the function one is evaluating.

Firstly, we use the spherical expression of our integral.

$$I = \int_0^\infty dr_1 \int_0^\infty dr_2 \int_0^\pi d\theta_1 \int_0^\pi d\theta_2 \int_0^{2\pi} d\phi_1 \int_0^{2\pi} d\phi_2 \frac{e^{-4(r_1+r_2)} {r_1}^2 {r_2}^2 \sin\theta_1 \sin\theta_2}{\sqrt{{r_1}^2 + {r_2}^2 - 2r_1r_2\cos\beta}}$$

By examining the integrand, one easily notices that there is an exponential dependency, so a clever way to simplify the expression, is by utilizing the exponential distribution and absorb the exponential factor in the integral in to the exponential PDF.

We do this by defining a exponential distribution on the form

$$p(y) = 4e^{-4y}$$

but our integral does not have the factor 4, so we will have to divide by that factor later when we are doing our calculation.

And with the assumption that the probability is conserved we have

$$p(y)dy = p(x)dx$$

where $p(x)$ is the uniform distribution with $x \in [0, 1]$ and $p(x)dx = dx$.

Then if we integrate up we get

$$x(y) = \int_0^y 4e^{-4y'} dy' = 1 - e^{-4y}$$

On the other hand, this means that

$$y(x) = -\frac{1}{4}\ln(1-x)$$

This gives us the new random variable y in the domain $y \in [0, \infty)$ determined through the random variable $x \in [0, 1]$.

This means that the expression we are integrating reduces into

$$f(r_1, r_2, \theta_1, \theta_2, \phi_1, \phi_2) = \frac{{r_1}^2 {r_2}^2 \sin\theta_1 \sin\theta_2}{\sqrt{{r_1}^2 + {r_2}^2 - 2r_1 r_2 \cos\beta}}$$

And now we can approximate the integral into

$$\begin{aligned} I &= \int_0^\infty dr_1 \int_0^\infty dr_2 \int_0^\pi d\theta_1 \int_0^\pi d\theta_2 \int_0^{2\pi} d\phi_1 \int_0^{2\pi} d\phi_2 \frac{e^{-4(r_1+r_2)} {r_1}^2 {r_2}^2 \sin\theta_1 \sin\theta_2}{\sqrt{{r_1}^2 + {r_2}^2 - 2r_1 r_2 \cos\beta}} \\ &\approx \frac{4\pi^4}{16} \frac{1}{N} \sum_{i=1}^{N} f(r_{1(i)}, r_{2(i)}, \theta_{1(i)}, \theta_{2(i)}, \phi_{1(i)}, \phi_{2(i)}) \end{aligned}$$

Where the exponential factor has been absorbed into the exponential distribution with $r_{1(i)}$ and $r_{2(i)}$ being random numbers distributed by the exponential distribution in the interval $[0, \infty)$. The angles $\theta_{1(i)}$ and $\theta_{2(i)}$ are random numbers uniformly distributed in $[0, \pi]$, while $\phi_{1(i)}$ and $\phi_{2(i)}$ are random numbers uniformly distributed in $[0, 2\pi]$.

The factor $4\pi^4$ comes from the integration limits for the angles, $(\pi - 0)^2 \times (2\pi - 0)^2$, where as the factor $\frac{1}{16}$ comes from when we multiplied the factor 4 into $p(y)$ earlier for both r_1 and r_2.

References

[1] Max Born and Robert Oppenheimer. Zur quantentheorie der molekeln. *Annalen der physik*, 389(20):457–484, 1927.

[2] D. R. Hartree. The wave mechanics of an atom with a non-coulomb central field. part i. theory and methods. *Mathematical Proceedings of the Cambridge Philosophical Society*, 24(1):89–110, 1928.

[3] David J Griffiths. *Introduction to quantum mechanics*, page 152. Cambridge University Press, second edition, 2016.

[4] M. Hjorth-Jensen. Computational Physics. *University of Oslo*, `https://github.com/CompPhysics/ComputationalPhysics`, 2013.

[5] H. Bruns. Zur theorie der kugelfunctionen. *Journal für die reine und angewandte Mathematik*, 90:322–328, 1881.

[6] Gabriel Szegö. Inequalities for the zeros of legendre polynomials and related functions. *Transactions of the American Mathematical Society*, 39(1):1–17, 1936.

[7] William H Press, Saul A Teukolsky, William T Vetterling, and Brian P Flannery. *Numerical recipes: The art of scientific computing*, page 184. Cambridge university press, third edition, 2007.