

# Project 3

Anna Stray Rongve  
Knut Magnus Aasrud  
Amund Midtgard Raniseth

October 20, 2019

## Abstract

This report addresses different numerical methods for solving a six-dimensional integral. The integral of interest is the energy between two electrons in a helium atom repelling each other, due to the Coulomb interaction. We assume that the wave function for each electron can be modelled like the single-particle wave function of an electron in the hydrogen atom. Solving this integral is done using Gaussian-Quadrature with Legendre and Laguerre polynomials, as well as two approaches to the Monte Carlo method of integration. The standard deviation of these solutions are also calculated. In addition to this, every procedure is timed for comparison.

## 1 Introduction

Development in methods for solving integrals has been important in order to solve problems with an increasing degree of complexity. Gaussian quadrature is a good example which is a method first developed by Jacobi in 1676. The first version gave exact results for algebraic polynomials of degree  $n-1$  or less. The "new" Gaussian version has a significant increase in accuracy with exact results for polynomials of degree  $2n-1$  or less due to free choice of weights.

KILDE: woho

Gauss-Legendre and Gauss-Laguerre are two types of Gaussian quadrature which, together with the well known Monte Carlo method, will be compared in accuracy and speed for a multidimensional integral for a Helium atom.

Some theory is first presented with a following discussion of the three methods mentioned above.

## 2 Theory

### 2.1 Wavefunction of Helium

The single-particle wave function of an electron  $i$  in the  $1s$  state is given in terms of a dimensionless variable (the wave function is not normalized) as

$$\psi_{1s}(\vec{r}_i) = e^{-\alpha r_i}$$

Where the electron position  $\vec{r}_i$  is

$$\vec{r}_i = x_i \vec{e}_x + y_i \vec{e}_y + z_i \vec{e}_z$$

and its distance from the origin  $r_i$  is

$$r_i = \sqrt{x_i^2 + y_i^2 + z_i^2}$$

$\alpha$  is a parameter set to 2, which corresponds to the charge of the Helium atom,  $Z = 2$ .

For our system with two electrons, we have the product of the two 1s wave functions defined as

$$\Psi(\vec{r}_1, \vec{r}_2) = e^{-\alpha(r_1 + r_2)}$$

This leads to the integral which will be solved numerically with the different methods mentioned earlier. The value of the integral corresponds to the expectation value of the energy between the two electrons repelling each other due to Columb interactions.

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

This is the integration that will be performed numerically in multiple ways in this paper. The analytical result is  $5\pi/16^2$ .

## 2.2 Gaussian Quadrature

The main idea of Gaussian quadrature is to integrate over a set of points  $x_i$  not equally spaced with weights  $w_i$ , which are calculated in `/code/Gauss-Quadrature/src/gauleg.cpp`. The weights are found through orthogonal polynomials(Laguerre and Legendre polynomials) in a set interval. The points  $x_i$  are chosen in a optimal sense and lie in the interval.

The intgral is approximated as

$$\int_a^b W(x)f(x) \approx \sum_{i=1}^n \omega_i f(x_i)$$

For a more detailed derivation and explanation of Gaussian quadrature see [1].

### 2.2.1 Gauss-Legendre

Using Gauss-Legendre quadrature with Legendre polynomials will make it possible to solve the integral numerically. The first step is to change the integration limits from  $-\infty$  and  $\infty$  to  $-\lambda$  and  $\lambda$ . The  $\lambda$ 's are found by inserting it for  $r_i$  in the expression  $e^{-\alpha r_i}$  because  $r_i \approx \lambda$  when  $e^{-\alpha r_i} \approx 0$ . From figure 1,  $\lambda \in [-5, 5]$  is therefore a good approximation for the integration limits.

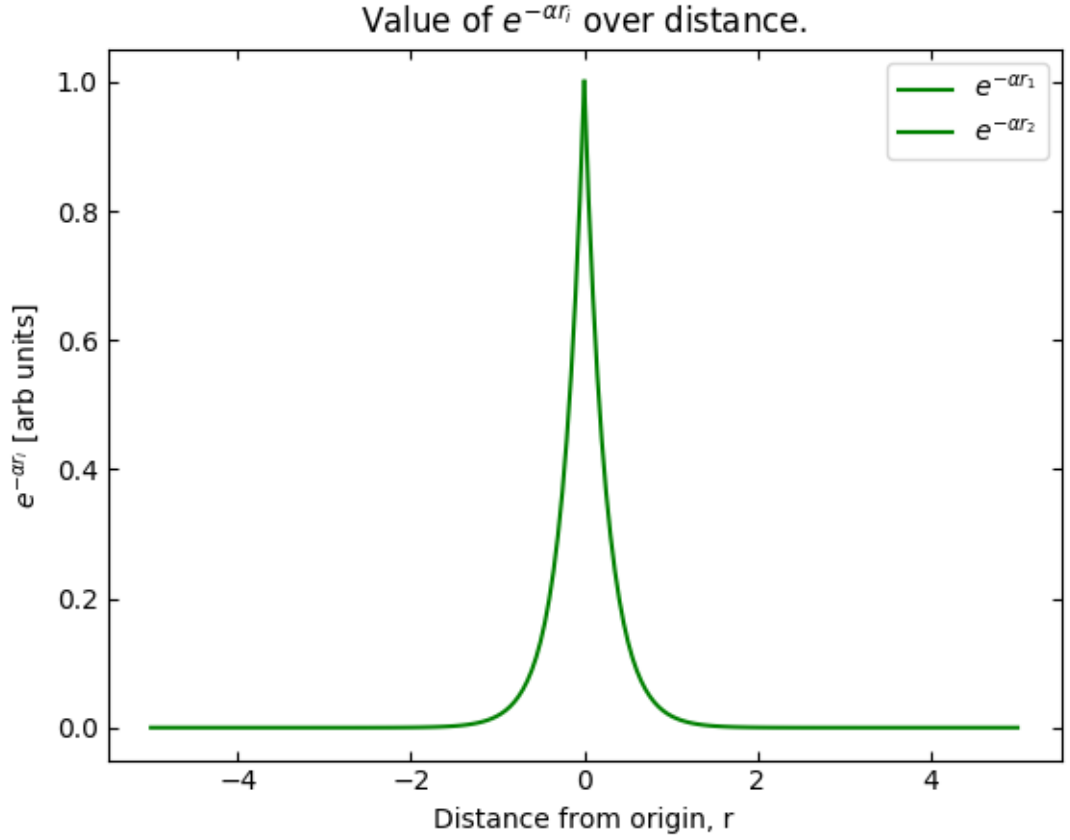


Figure 1: Plot of wavefunction in one dimension

The weights and mesh points are computed using `/code/Gauss-Quadrature/src/gauleg.cpp`. Eventually ending up with a sixdimensional integral, where all six integration limits are the same.

$$\int_a^b \int_a^b \int_a^b \int_a^b \int_a^b \int_a^b e^{-x} f(x) dx \approx \sum_{i=1}^n w_i f(x_i)$$

### 2.2.2 Improved Gauss-Quadrature- Laguerre

Gauss-Legendre quadrature gets the job done, but it is unstable and unsatisfactory. By changing to spherical coordinates and replacing Legendre- with Laguerre polynomials an improvement in accuracy is expected. The Laguerre polynomials are defined for  $x \in [0, \infty)$ , and in spherical coordinates:

$$d\vec{r}_1 d\vec{r}_2 = r_1^2 dr_1 r_2^2 dr_2 d\cos(\theta_1) d\cos(\theta_2) d\phi_1 d\phi_2$$

with

$$\frac{1}{r_{12}} = \frac{1}{\sqrt{r_1^2 + r_2^2 - 2r_1 r_2 \cos(\beta)}}$$

and

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

For numerical integration, the deployment of the following relation is necessary:

$$\int_0^\infty e^{-x} f(x) dx \approx \sum_{i=1}^n w_i f(x_i)$$

where  $x_i$  is the  $i$ -th root of the Laguerre polynomial  $L_n(x)$  and the weight  $w_i$  is given by

$$w_i = \frac{x_i}{(n+1)^2 [L_{n+1}(x_i)]^2}$$

The Laguerre polynomials are defined by Rodrigues formula:

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

or the recursive relation:

$$\begin{aligned} L_0(x) &= 1 \\ L_1(x) &= 1 - x \\ L_{n+1}(x) &= \frac{(2n+1-x)L_n(x) - nL_{n-1}(x)}{n+1} \end{aligned}$$

#### 2.2.1

## 2.3 Monte Carlo

### 2.3.1 Generalized

Monte Carlo integration is based on the idea of finding the mean of a function in a domain by sampling random function values. This mean multiplied by the volume of the domain will be an approximation of the integral.

Say we have an integral  $I$  of  $f(\mathbf{x})$  we want to find:

$$I = \int_D f(\mathbf{x}) d\mathbf{x}$$

where  $\mathbf{x}$  is in the domain  $D$ . This integral can be approximated by using random numbers distributed on  $D$  by the probability distribution function (PDF)  $p(\mathbf{x})$ . Discretizing, the approximated integral now becomes

$$I \approx \langle I \rangle = \frac{1}{N} \sum_{i=0}^N \frac{f(\mathbf{x}_i)}{p(\mathbf{x}_i)}, \quad (2)$$

where  $N$  is the number of sampled values.

### 2.3.2 Naïve approach (uniform PDF)

To solve our six-dimensional integral, we first take the naïve approach and distribute our randomly chosen variables on the uniform distribution

$$\theta(x) = \begin{cases} \frac{1}{b-a}, & \text{for } x \in [a, b] \\ 0 & \text{else} \end{cases},$$

and keep our variables  $\mathbf{r}_1$  and  $\mathbf{r}_2$  in cartesian coordinates. Putting the uniform distribution into (2), we get the naïve approximation of an integral:

$$\langle I \rangle = \frac{V}{N} \sum_{i=0}^N f(\mathbf{x}_i). \quad (3)$$

Here  $V$  is the integration volume (for  $d$  dimensions in cartesian coordinates  $V = (b - a)^d$ , with  $b$  and  $a$  being the integration limits for each dimension). Going back to our original integral (1), our approximation of it using this method is

### 2.3.3 Paralellization

To run the computations faster, openMP will be used to paralellize the code. This shares the workload across multiple processor threads and results in a substantial decrease in time spent for the same amount of operations. Some important remarks when doing Monte-Carlo integration in paralell is:

- Create a random number generator in each thread.

Legendre		
N	Value	Error
11	0.297447	0.104681
15	0.315863	0.123098
21	0.268075	0.075310
25	0.240135	0.047370
27	0.229623	0.036858

Table 1: Fill me in!

Laguerre		
N	Value	Error
11	0.183021	0.009743
15	0.193285	0.000520
21	0.194807	0.002050
25	0.194804	0.002030
27	0.194795	0.002029

Table 2: Fill me in!

- Keep the summations private for each thread.
- Sum the private summations from each thread together after the calculations are completed.

By doing this we avoid having the threads wait for the random number generator and writing to the same memory, thereby achieving optimal speedup. The code is commented in for example `/code/Monte-Carlo/src/naiveMC.cpp`.

## 3 Results

### 3.1 Laguerre/Legendre

$$N \in [-5, 5]$$

### 3.2 Paralellization

Our paralellization results was achieved using a quad core Intel Core i5-8250U processor with 6MB cache at 1.6GHz base clock, which boosted to 3.4GHz during testing. Thermal throttling was avoided. The memory was 4GB 1866MHz LPDDR3 soldered on board.

We also ran this test on an octa-core processor with memory of 8GB 1866MHz, and achieved no noticable speedup compared to the abovementioned computer.

For runtime inputs the number of samples was set to  $10^8$ , with an approximation of infinity of  $\lambda = 5$ .

Compile flags	-O3 -fopenMP	-O3	-fopenmp	no optimization
Naive MC	12s	31s	71s	173s
Improved MC	15s	38s	79s	200s

Table 3: Shows the time spent on the same calculations with different compile parameters on a quad core processor. ( $N = 10^8, \lambda = 5$ )

## 4 Discussion

### 4.1 Paralellization

From table 3 it is easy to understand the impact of correct optimization. Not only was the paralellization of the code a big time-saver but also the vectorization flag (-O3) made a really dramatic impact.

Both from no optimization, to paralellization, and from vectorization to vectorization and paralellization, the time spent is halved. However, this was paralellized over four cores, so shouldn't the time be one fourth of the original? The bottleneck is probably memory speed, as we ran the same calculations on a octacore processor with more capacity, but same frequency RAM, and achieved the same results.

This means that further improvements on the paralellization can be done by using faster memory, or changeing the code to access memory less frequent.

## 5 Conclusion

this is a reference to intro: 1

## References

- [1] Morten Hjorth-jensen. *Computational Physics Lectures : Introduction to Monte Carlo methods*. 2019.