Project 3

Anna Stray Rongve
Knut Magnus Aasrud
Amund Midtgard Raniseth

October 22, 2019

Abstract

This report adresses different numerical methods for solving a six-dimensional integral. The integral of interest is the energy between to electrons in a helium atom repelling eachother due to the Coloumb 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. The integral is solved with Gaussian-Quadrature with Legendre and Laguerre polynomials, as well as two approaches to the Monte Carlo method. The standard deviation of these solutions are also calculated.
We find that while Gauss-Laguerre and Gauss-Legandre have different areas of performance, Monte Carlo integration is an overall better numerical integral solver - at least for multi-dimensional integrals.

1 Introduction

Development in methods for solving integrals have long been important in order to solve problems with an increasing degree of complexity. Gaussian quadrature is an elegant method, that drastically increases the accuracy of numerical integration, compared to other methods like trapizoidal integration. It was first developed by Jacobi in 1676. The first version gave exact results for algebraic polynomials of negree $n-1$ 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. [3]
Gauss-Legendre and Gauss-Laguerre are two types of Gaussian quadrature which in this report will be compared in accuracy and speed for a multidimensional integral describing the energy of electrons in a Helium atom. In addition, two approaches to the Monte Carlo method of integration are implemented and compared as well. Parallelization will also be done to the program running the Monte Carlo integration.
First the theory of the different methods is presented, followed by our results and finally a discussion.

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

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

where the electron position $\mathbf{r}_i$ is

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

and its distance from the origin r_i is

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

α is a parameter set to 2, which corresponds to the charge of the helium atom, $Z = 2$. [1]

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

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

This leads to the integral which will be solved nummerically with the different methods mentioned above. The value of the integral corresponds to the expectation value of the energy between the two electrons repelling each other, due to Columb interactions, and is as follows:

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

This is the integration that will be performed numerically in multiple ways in this paper. The analytical result is $5\pi^2/16^2 \approx 0.1927657$.

2.2 Gaussian Quadrature

The main idea of Gaussian quadrature is to integrate over a set of points x_i, not equally spaced and each with weights w_i. The weights are found through orthogonal polynomials (Laguerre and Legendre) in a set interval. The points x_i are chosen in an optimal sense and lie in said interval.
The integral is approximated as

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

For a more detalied derivation and explanation of Gaussian quadrature see [2]. The program used to find mesh points and weights is found at `/code/Gauss-Quadrature/src/gauleg.cpp` .

2.2.1 Gauss-Legendre

Using Gauss 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 ∞ to $-\lambda$ and λ. λ is 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.

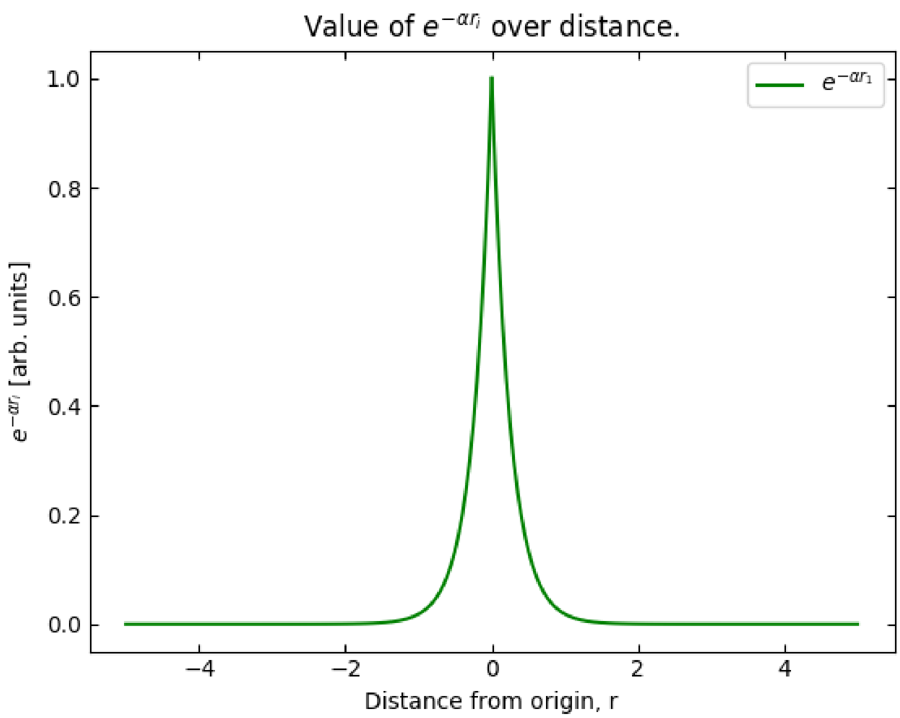

Figure 1: Plot of wavefunction in one dimension

Eventually we end 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

The Gauss-Legendre quadrature gets the job done, but it takes a large number of integration points to yield satisfactory results, and the choise of λ is hard to make. 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:

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

with

$$\frac{1}{r_{12}} = \frac{1}{\sqrt{r_1^2 + r_2^2 - 2r_1 r_2 cos(\beta)}} \tag{3}$$

and

$$cos(\beta) = cos(\theta_1)cos(\theta_2) + sin(\theta_1)sin(\theta_2)cos(\phi_1 - \phi_2) \tag{4}$$

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

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

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 the Rodrigues formula:

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

or the recursive relation:

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

2.3 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

$$I \approx \langle I \rangle = \frac{1}{N} \sum_{i=0}^{N} \frac{f(\mathbf{x}_i)}{p(\mathbf{x}_i)}, \tag{5}$$

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 (5), we get the naïve approximation of an integral:

$$\langle I \rangle = \frac{V}{N} \sum_{i=0}^{N} f(\mathbf{x}_i). \tag{6}$$

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

$$\langle I \rangle = \frac{(b-a)^2}{N} \sum_{i=0}^{N} e^{-2\alpha(r_{1,i}+r_{2,i})} \frac{1}{|\mathbf{r}_{1,i} - \mathbf{r}_{2,i}|}, \tag{7}$$

with $\mathbf{r}_{1/2,i}$ being randomly chosen vectors and $b = a = \infty$, or our approximation of infinity, namely $\lambda = 5$ (see section 2.2.1).

2.3.3 Importance sampling (exponential distribution)

As mentioned in section 2.2.1, our integrand quickly goes to zero. This means that inserting bigger approximations for infinity, λ, requires a greater number of sampling points, since we are not sure if the random numbers will give us the significant values of the integrand.

A sensible way around this is to distribute the randomly chosen variables on a probability distribution matching the function we're integrating. The quite obvious choice here is the exponential distribution $\lambda e^{-\lambda x}$. Inserting it into the general Monte Carlo integral approximation (equation (5)), together with the integrand we are finding the integral of, we get

$$\langle I \rangle = \frac{1}{N} \sum_{i=0}^{N} \frac{e^{-2\alpha(r_{1,i}+r_{2,i})}}{\lambda e^{-\lambda(r_{1,i}+r_{2,i})}} \frac{1}{|\mathbf{r}_{1,i} - \mathbf{r}_{2,i}|} = \frac{1}{4N} \sum_{i=0}^{N} \frac{1}{|\mathbf{r}_{1,i} - \mathbf{r}_{2,i}|}.$$

Here we put $\lambda = 4$, since $\alpha = 2$. This distribution does however not apply well with negative numbers, and thus we have to change into spherical coordinates. With the results from equations (2), (3) and (4), our approximated integral now reads:

$$\langle I \rangle = \frac{\pi^4}{4N} \sum_{i=0}^{N} \frac{r_1^2 r_2^2}{\sqrt{r_1^2 + r_2^2 - 2r_1 r_2 \cos\theta_1 \cos\theta_2 + \sqrt{1 - \cos\theta_1^2}\sqrt{1 - \cos\theta_2^2}\cos(\phi_1 - \phi 2)}}. \tag{8}$$

2.4 Standard deviation (STD)

The variance of our function mean value is given as

$$\sigma_f^2 = \frac{1}{N} \sum_{i=0}^{N} (f(\mathbf{x}_i) - \langle f \rangle)^2 = \langle f^2 \rangle - \langle f \rangle^2,$$

and thus the variance of the approximated integral is

$$\sigma_I^2 = \frac{V^2}{N^2} \sum_{i=0}^{N} \sigma_f^2 = \frac{V^2 \sigma_f^2}{N}.$$

The standard deviation of our Monte Carlo integration is the square root of the variance, so

$$\text{STD} = \sigma_I = \frac{V\sigma_f}{\sqrt{N}}. \tag{9}$$

2.5 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 substantional 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.
- 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 Gauss-Legendre

Solving our integral with Legendre polynomials gives unstable results for $\lambda = 5$ as seen in table 1. Though with a careful choice of $N = 27$ and integration limits $\lambda = 2.9$ our results are precise with 4 leading digits after the decimal point. The error is calculated as the absolute error for both Laguerre and Legendre. The results from our Legendre integration program are found in table 1.

,

Legendre		
N	Approximate integral	Error
11	0.297447	0.104681
13	0.318350	0.125584
15	0.315863	0.123098
17	0.302268	0.109502
19	0.285064	0.092298
21	0.268075	0.075310
25	0.240135	0.047370
27	0.229623	0.036858
27*	0.192725	0.000039

Table 1: Values of the integral for different N's, calculated with Gauss-Legendre. Integration limits are $x \in [-5, 5]$. *: Special case with integration limits $x \in [-2.9, 2.9]$

3.2 Gauss-Laguerre

Improving our algorithm using Legendre polynomials for angles and Laguerre polynomials for radial parts improved accuracy of our results. An increase in N from N = 11 to N = 15 gives an increase in precision, though for any higher increase the accuracy decreases slightly, which is shown in table 2.

Laguerre		
N	Approximate integral	absolutt error
11	0.183021	0.009743
13	0.190217	0.002548
15	0.193285	0.000520
17	0.194396	0.001630
19	0.194732	0.001965
21	0.194807	0.002050
25	0.194804	0.002030
27	0.194795	0.002029

Table 2: Values of the integral for different N's, calculated with Gauss-Laguerre. Integration limits are $x \in [-5, 5]$.

3.3 Monte Carlo

3.3.1 Naïve approach

The results from our Monte Carlo integration program (main.exe), are listed in table 3.

Naïve Monte Carlo			
N	Approximate integral	Standard deviation	Error
10^5	0.21953065	0.154683	0.026764935
10^6	0.14149215	0.0368397	0.051273556
10^7	0.16704012	0.023165	0.025725592
10^8	0.17903453	0.00936631	0.013731177
10^9	0.19105511	0.0041004	0.0017106036

Table 3: Results from running Monte Carlo with cartesian coordinates and integration limits $x \in [-5, 5]$ - our approximation of infinity.

For higher N's, the approximated integral get closer to the actual value and the standard deviation decreases. The error ($|\text{Exact} - \text{Approximated}|$) does however not match up with the standard deviation, and oscillates a bit up and down, despite having a decreasing trend.

3.3.2 Importance sampling

The results from our Monte Carlo integration program (main.exe), are listed in table 4.

Improved Monte Carlo			
N	Approximate integral	Standard deviation	Error
10^5	0.13773907	0.284624	0.055026645
10^6	0.19068327	0.405372	0.0020824368
10^7	0.2075781	0.381901	0.014812393
10^8	0.19459392	0.092418	0.001828214
10^9	0.20918288	0.0646068	0.016417166

Table 4: Results from running Monte Carlo with importance sampling along the exponential distribution and using spherical coordinates.

The improved Monte Carlo integration gets within a small error margin for smaller N's than the naïve, However, it over- and undershoots randomly. The trend is that the standard deviation decreases, but does not match up with the error ($|\text{Exact} - \text{Approximated}|$).

3.4 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 2133MHz LPDDR3 soldered on board. See table 5

We also ran this test on an octa-core processor with memory of 8GB 2400MHz (12.5% faster), and achieved an additional speedup compared to the abovementioned computer. See table 6

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

Runtime with different optimizations				
Compile flags	-O3 -fopenMP	-O3	-fopenmp	No optimization
Naive MC	12s	31s	71s	173s
Improved MC	15s	38s	79s	200s

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

Runtime with optimization on octa-core		
Compile flags	-O3 -fopenMP	% faster than the quad-core
Naive MC	8s	50%
Improved MC	11s	36%

Table 6: Shows the time spent on the Monte-Carlo calculations on an octa-core system.($N = 10^8, \lambda = 5$)

4 Discussion

4.1 Gaussian quadrature

From the tables presented in the results section (3.1 and 3.2) one can simply compare the two methods in accuracy and stability with increasing N. It seems that for the Quadrature methods, a higher number of integration points, beyond what discussed in the results($N > 11$), does not yield better results. The reason for this might be due to the function beeing really close to zero beyond $a = -3$ and $b = 3$. The Laguerre and Legandre polynomials might also increase for a higher N. In the special case for $N = 27$, $a = -2.9$ and $b = 2.9$ the results improove. A good improvement of the experiment/study might therefore be to compare results for different integration limits eg. $[-3, 3]$ or $[-4, 4]$. Comparing Legendre with Laguerre there is a significant improvement in precision - and to some point the stability as well - when increasing N, which is expected.

4.2 Monte Carlo

Looking at the results, the non-deterministic nature of Monte Carlo integration shines through. They are not consistent across runs and seem to fluctuate randomly (which they of course do). However, looking at the standard deviation and error, the trend is that the accuracy increases - which is a good sign. The approximations also come "quite" close (meaning order of 10^{-3}...). From equation (9) in section 2 Theory, its also worth noting that the standard deviation decreases by order of $\frac{1}{\sqrt{N}}$, regardless of how many dimensions you integrate. Compared to the Gaussian-Quadrature methods, this makes Monte Carlo integration way more viable for multi-dimensional integral solving.

As a note to the large difference between the standard deviation and the error, this is probably due to the high unceirtanty in the integral result. We might get lucky and get a spot on answer - giving us a tiny error - but other times we might not. This is reflected in the standard deviation being quite high. This is an issue, since we don't get quite as accurate answers as we might've hoped, but it is probably due to not efficient enough code and computers that aren't powerful enough.

4.3 Parallelization

From table 5 it is easy to understand the impact of correct optimization. Not only was the parallelization of the code a big time-saver but also the vectorization flag (-O3) made a dramatic impact.

Both from no optimization, to parallelization, and from vectorization to vectorization and parallelization, the time spent is halved. However, this was parallelized over four cores, so shouldn't the time be one fourth of the original? The bottleneck is probably a mix of memory speed, small cache and low processing power as the memory speed upgrade of 12% on the octa-core PC is not enough to justify the 40% speed increase compared to the quad-core PC.

This means that further improvements on the parallelization can be done by using faster memory, changing the code to access memory less frequently, and to add more processing power.

5 Conclusion

It seems for the quadrature methods that a higher number of integration points N - beyond whats discussed in the results($N > 11$) - does not yield better results. Comparing Legandre with Laguerre there is a significant improvement in precision, not least when it comes to stability when increasing N.

Monte Carlo, on the other hand, is way less stable and the results can be unpredictable - at least for our implementation. However, compared to the quadrature methods, Monte Carlo is preferred when it comes to multi-dimensional integrals like the one solved here. This is due to the computational power needed and the complexity of the calculations, which are simplified/reduced drastically by utilizing non-deterministic integration.

Parallelization and other methods of optimizing had a big impact on the speed of the calculation. However, for this relatively simple integral, the time spent is not exactly impressive.

References

[1] Morten Hjorth-Jensen. *Project 3, numerical integration, deadline October 21*. 2019.

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

[3] Gradimir V. Milovanović. "Generalized Gaussian quadratures for integrals with logarithmic singularity". In: *Filomat* 30.4 (2016), pp. 1111–1126. ISSN: 03545180. DOI: 10.2298/FIL1604111M.