Draft version October 23, 2019
Preprint typeset using LaTeX style emulateapj v. 12/16/11

NUMERICAL INTEGRATION

Torstein Solheim Ølberg and Ada Mathea S. Veddegjerde

Draft version October 23, 2019

ABSTRACT

We study the quantum mechanical expectation value of the correlation energy between two electrons which repel each other. We find this by calculating an integral using different integration methods. The methods we use are Gaussian Quadrature with Legendre and Laguerre polynomials, and Monte Carlo. We find that the Legendre polymonials do not give good enough results, and Laguerre, even though it gives better results, does not give very excact result and need to many summations. We had problems with using the Monte Carlo method, but an improved Monte Carlo gave very good results.

1. INTRODUCTION

In this paper we will be studying the wave functions of electrons, we will modell them like the single-particle wave function of an electron in the hydrogen atom.

We want to find the quantum mechanical expectation value of the correlation energy between two electrons which repel each other via the classical Coulomb interaction, we get this by computing the integral 1.

We will attempt to compute the integral, and try different methods to achieve favorable results. The methods we will use are Gaussian Quadrature with both Legendre and Laguerre polynomials, and Monte Carlo methods.

In the report we will start by going through all the theory we will use during our calculations. The we will go through what we did in the metods section. Our results will then be presented in graphs and tables in the results section. These results are what we later discuss and draw our conclutions from.

2. THEORY

2.1. *Particle Wavefunction*

The single-particle wave function for an election i in the $1s$ state is given in terms of a dimentionless variable (the wave funcion is not properly normalized)

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

as

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

where α is a parameter which we will fix equal to 2, corresponding to the charge of the helium atom $Z = 2$, and

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

The ansatz for the wave function for two electrons is then given by the product of two so-called $1s$ wave functions as

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

The expectation value is given by:

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

torsteol@student.matnat.uio.no
amveddeg@student.matnat.uio.no

[1] Institute of Physics at the University of Oslo

Ignore the missing normalization factor.

2.2. *Gaussian Quadrature*

The Gaussian Quadrature is a numerical integration method based on approximating an integral by a sum of a weight and a function at spesific points.

$$I = \int_a^b f(x)dx \approx \sum_{i=0}^{N} w_i f(x_i)$$

The weights are determined from a type of polynomial of order N and x_i is the zero-points of that polynomial.

2.3. *Weight Functions*

A weight function is defined to give the most exact results for a given polynomial. With it we get integration points defined spesifically for the shape of the polynomial it belongs to.

2.4. *Legendre Polynomials*

Legendre polynomials are polynomials that solve the differencial equations

$$C(1-x^2)P + (1-x^2)\frac{d}{dx}\left((1-x^2)\frac{dP}{dx}\right) = 0$$

For the Gauss-Legendre Quadrature the weights are then given by $2L_{0,i}^{-1}$ where $L_{N,i}$ is the Legendre polynom of N-th order. The i is a counter for the x values, which are the null-points of the polynomial.

2.5. *Laguerre polynomials*

Laguerre polynomials are solutions of Laguerre's equation:

$$xy'' + (\alpha + 1 - x)y' + ny = 0$$

Note that this gives the generalized Laguerre polynomials, since α can have other values than zero.

The weight function for Laguerre polynomials is given by

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

This weight function is used to rewrite integrals, like so:

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

2.6. *Spherical coordinates*

One can switch to spherical coordinates like so:

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

with

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

and

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

2.7. *Cosinus and Sinus Relation*

It can be usefull to know that

$$d(g(x)) = g'(x)dx.$$

And therefore

$$\int_a^b d(g(x)) = \int_a^b g'(x)dx,$$

2.8. *Monte Carlo Method*

The Monte Carlo method selects N random numbers (x_i) in a given interval and use the numbers to evaluate the function $f(x_i)$. Summing the contributions we get gives us the final mean value

$$\langle f \rangle = \frac{1}{N} \sum_{i=1}^{N} f(x_i) p(x_i), \tag{4}$$

and standard deviation

$$\sigma_f^2 = \left(\frac{1}{N} \sum_{i=1}^{N} f(x_i)^2 - \left(\frac{1}{N} \sum_{i=1}^{N} f(x_i) \right)^2 \right) p(x_i) \tag{5}$$

In the brute force method, with uniform distribution, $p(x_i) = 1$.

For exponential distribution

$$p(r) = \alpha exp(-\alpha r) \tag{6}$$

A usefull approximation

$$\int_0^\infty f(r)p(r) \approx \frac{1}{N} \sum_i^N f(r_i) \tag{7}$$

To map uniform distribution values to exponential distribution use

$$r_i = -log(1 - x_i) \tag{8}$$

3. METHOD

First we used the Gaussian Legendre Quadrature method to compute the integral 1 in C++ for N equal to 3, 4 and 5. We used the function gauleg from the lib.cpp library in Ølberg og Veddegjerde (2019) to generate the . For some reason it was not possible to use $N = 6$ for this method, as the program either crashed or run for ever. Then we plotted the result for $N = 5$.

TABLE 1
Obtained values from Gaussian Legendre

N	Values
3	1.05764
4	0.287955
5	0.129483

Note. — We see that that the value from the integration converges to something with higher values of N

Now we want to try improving our results. Therefore we will try using Laguerre polynomials 2.5. First we will switch 1 to spherical coordinates (2.6)

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

We also see that we can use 2.7, and insert this.
When we compute this we want to implement the function we integrate without the weight function 2. With the way it is written now it is hard to exctract this weight function so we perform a change of variables. We define the new variables to be $u_1 = 2\alpha r_1$ and $u_2 = 2\alpha r_2$.
Now we can easily exctract the weight function (2), which now looks like $u_1^2 u_2^2 e^{-(u_1+u_2)}$. We are left with

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

We compute this for N equal to 3, 5, 10 and 15. We used the function gaulag from the same library as the last implementation. We plotted the result for $N = 5$.

Then we implemented the Monte Carlo integration method, using the random number generator *std* :: $uniform\_int\_distribution$ and N points ranging from one to 10 000. We plotted the result.

We tried optimalizing our result by using the same method as we used when calculating with Laguerre polynomials, when we switched to polar coordinates. Here we also had to take in to account that the $r_1^2 r_2^2$ do not get taken care of by polynomals as with Laguerre, and we must include this in our function (6, 7). We also had to map random numbers $x \in [0, 1]$ we found with *std* :: $uniform\_real\_distribution$, with an exponential relation 8. The angles we computed with a simple mapping $\lambda \in \pi[0, 1]$ and $\phi \in 2\pi[0, 1]$. We plotted the result. Finally we checked the CPU time for the Monte Carlo methods, and in an attempt to shorten this time we parallelied our code. These time we put in a table.

4. RESULTS

Figure 1 is the integration value in our first attempt at calculating the integral. We tried calculating with different values of N, these result have been put in the table 1. Figure 2 is the integration value after we changed to the Laguerre method. Here we tried with more N values, the result can be found in the table 2.

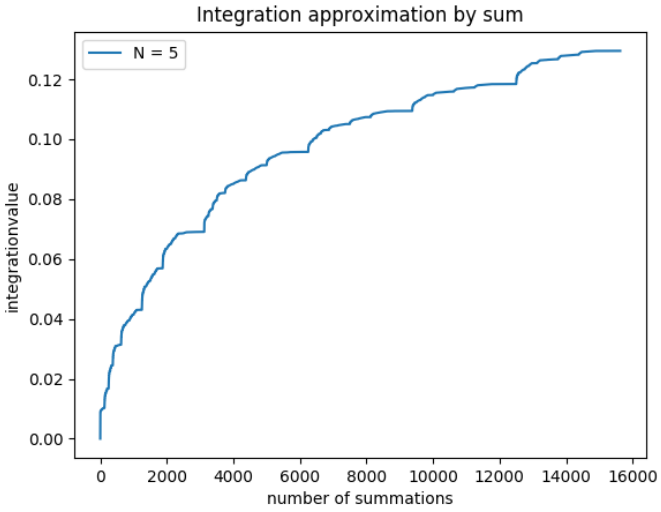

FIG. 1.— The integration sum by time. From using Legendre. A plot of the integration value as a function of the number of summations (N^6). The graph increases from zero in a pretty even curve, starting to flatten at about summation 6000, but still rising at the last summation.

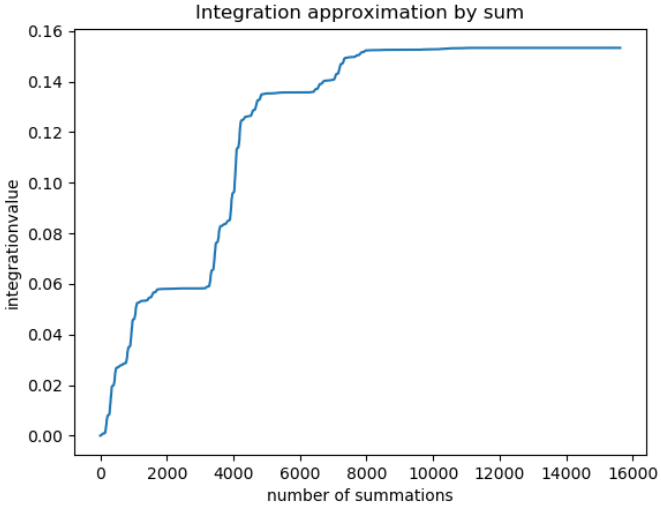

FIG. 2.— The integration sum by time. From using Laguerre. A plot of the integration value as a function of the number of summations (N^6). The graph increases from zero untill it flattens at 0.15.

TABLE 2
OBTAINED VALUES FROM GAUSSIAN LAGUERRE

N	Values
3	0.118045
5	0.153304
10	0.178245
15	0.185254

NOTE. — We see that that the value from the integration converges to something with higher values of N

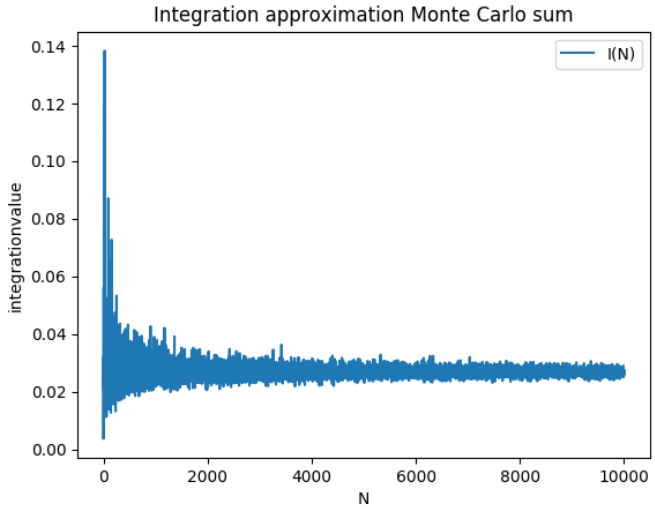

FIG. 3.— The integration value for increasing N for a brute force Monte Carlo integration. We see that it converges to some value around 0.03 for larger and larger N.

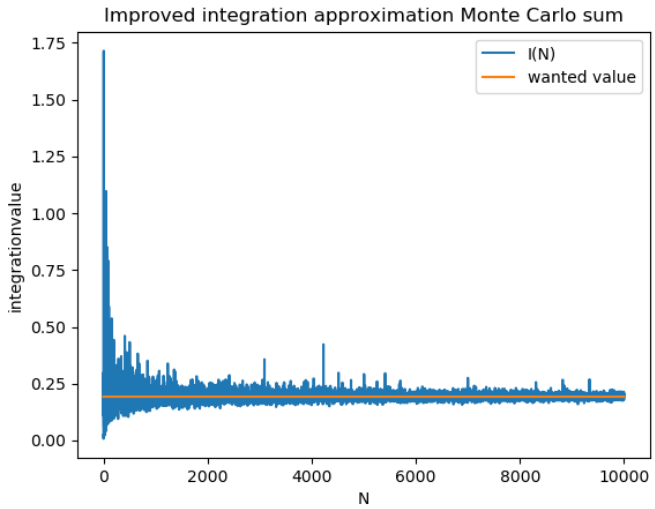

FIG. 4.— The integration value for increasing N for an improved brute force Monte Carlo integration. We see that it converges to some value around 0.1927657 for larger and larger N.

TABLE 3
CPU TIME FOR RUNNING MONTE CARLO INTEGRATION

Integration method	CPU time
MC not parallelized	0.002317 s
IMC not parallelized	54.3992 s
MC parallelized	0.004412 s
IMC parallelized	116.607 s

NOTE. — Table over the CPU times for running Monte Carlo integration (MC) and the improved Monte Carlo integration (IMC). For both parallelized code, and not parallelized code. The values for the parallelized code are almost twice as large.

Figure 3 shows the result of our brute force Monte Carlo integration for increasing values of integration points N. It is clear the it converges to som value around 0.03.

Figure 4 shows the result of our improved brute force Monte Carlo integration for increasing values of integration points N. It is clear the it converges to som value around 0.19.

The table 3 shows the CPU time for running the code for brute force Monte Carlo (3) and the improved brute force Monte Carlo (4). The Monte Carlo is generally faster that the improved Monte Carlo, and the non parallelized code is generally faster than the parallelized code.

5. DISCUSSION

The result we get from using the Legendre method is shown in 1 and 1. The integration is not very satisfactory, since the value we want to get is $\frac{5\pi^2}{16^2} \approx 0.1927657$ and the value we get is around 0.13 (1). If we look at the table (1) we start at 1.06 and get smaller values for bigger N, so the desired value is actually skipped between $N = 4$ and $N = 5$. Therefore this method is not particularly stable when N is a low number. We assume the value will stabilize with higher values of N, but we are not able to test this as it requires too many calculations.

Now we try the Laguerre method with the hope that we will get more accurate results. We show the results in the same format as we did the Legendre method, with 2 and 2. Now we achieve better results. We still do not get a very accurate integration value, but now we can see that the graph (2) flattens out after about 8000 summations, which tells us the integration value it has found is quite stable. the value varies quite alot for different values for N. Even with $N = 15$, which is absurdly high for this method since it does too many summations to be effective, we do not get a very exact integration value. However in this method the integration value starts quite low and increases with higher N, therefore we can se that is converges towards the correct integration value.

Since we still have not been able to get a satisfactory result we use the brute force Monte Carlo method (2.8), which is known to be an effective and accurate method for multidimentional integrals. Sadly it converges to a value around 0.03, which is way of the expected value. This might be because the implementation of the random number generator is faulty, giving an uneven distribution instead of a uniform distribution. But it could just as easily be an error in our calculations. Anyhow we can not use this result when trying to determine the best method for integration since it does not work for us.

In a final attempt at getting an accurate result we have the improved Monte Carlo, the result of which we can see in 4. Here we can see an extremely accurate calculation. The graph quickly converges towards the wanted value.

The results we got from the CPU time (3) seem fine, the improved Monte Carlo taking much longer as expected. However the parallelized code acctually use longer, which is strange when we do these types of calculations. Anyhow it gives non satisfactiory results. Parhaps it could be done in a different way to make the code faster, but we were not able to find a way to do that.

6. CONCLUSIONS

From the results we can see that the with larger N-th order polynomials for both Legendre and Laguerre Gauss-Quadrature gives better results, but they are not very satisfactory for the low level Ns we are able to use. However, we can see that the Laguerre method gives better results than the Legendre method in this case.

We were unable to conclude anything from the ordinary Monte Carlo method. But the improved Monte Carlo method gave very satisfactiory results, even if it takes a lot of time, the result is worth it.

REFERENCES

Project 3: Ølberg, Torstein Solheim; Veddegjerdet, Ada Mathea; 2019, https://github.com/MrTorstein/Fys3150_Project3