Project 3

Benedicte Allum Pedersen, Emil Heland Broll
Fredrik Oftedal Forr

Abstract

We have computed the integral for the expectation value of the correlation energy between to electrons in a helium atom by using Gauss-Legendre and Gauss-Laguerre quadrature as well as Monte Carlo integration. We experienced that the brute force methods gave a less satisfactory resulst compared to the methods that required some more thinking and adaptation of the integrand and the approximated limits. The exact, analytical value of the integral should be 0.192765. The best value we got from Gauss-Legendre was 0.192651, while the best value from Gauss-Laguerre was 0.194779. We also experienced that Gauss-Laguerre gave better results when we increased the number of integration points, while this was not the case for Gauss-Legendre. The Monte Carlo brute force method improved the precision when increasing the number of integration points, and the improved Monte Carlo method followed the same general trend, but with some outliers. The fastest, most precise method was the improved Monte Carlo method, and overall, the brute force methods were slower and more imprecise.

Introduction

In this report we will give a brief rundown on different numerical integration methods. The methods we will go through are Gauss-Legendre and Gauss-Laguerre quadrature as well as Monte Carlo integration. We will also try to improve the different methods and find the strengths and weaknesses of the methods compared to each other.
We will explain the various integration methods, results from our integrations and a discussion of the results, and finally a conclusion on how the different methods perform and compare to each other.

The integral we will study is the six-dimensional integral determining the ground state correlation energy between two electrons in a helium atom.

The single-particle wave function for an electron i in the $1s$ state in a hydrogen atom is as follows:

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

where $\mathbf{r}_i = x_i\mathbf{e}_x + y_i\mathbf{e}_y + z_i\mathbf{e}_z$, and the value r_i is given by:

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

We set $\alpha = 2$ which corresponds to the charge of the helium atom, Z = 2. We assume that the wave function for each electron in the helium atom can be modelled like the single-particle wavefunction above. The wavefunction for two electrons is then given by the product of two single-particle wavefunctions:

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

We need to solve the intergral for the expectation value of the correlation energy between the two electrons:

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

The exact value of this integral is $5\pi^2/16^2 = 0.192765$.

Method

In this project we are using C++ to program the numerical algorithms for solving the integrals.

Our integrand

In order to perform the integration, we need to know how to approximate the infinite limits of the integral. To do this, we plot the wave function for the electrons in the Helium atom in figure 1. From this plot we can tell that the integral converges to zero for $x < -2$ and $x > 2$. Therefore the integral would not need to be evaluated outside of those values of x, as the total contribution from the area outside of that interval would be negligible.

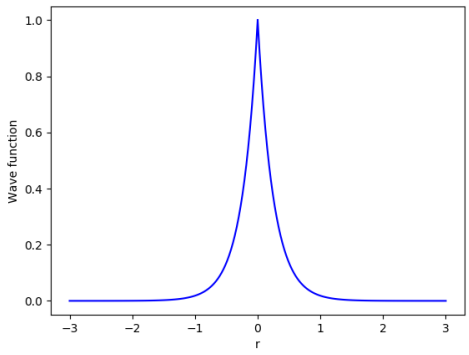

Figure 1: Plot of the wave function for to electron in a Helium atom.

Gaussian quadrature

The first method we will look at is Gaussian quadrature. To compute the integral numerically we approximate by discretising our variables:

$$I = \int_a^b f(x)dx = \int_a^b W(x)g(x)dx \approx \sum_{i=1}^{N} \omega_i f(x_i)$$

Unlike other more basic methods for numerical integrations where the mesh points x_i are equidistantly spaced, the mesh points in the Gaussian quadrature are not equidistantly spaced. In the sum above, the w_i corresponds to the weights. Gaussian quadrature uses orthonogonal Legendre- and Laguerre-polynomials to obtain the mesh points and weights.

Gauss-Legendre is a brute-force method, and in order to obtain the value of the integral using this method we have to set up the mesh points and weights that corresponds to some finite integrations limits [a, b]. To set up the mesh points and weights we haved used the function *gauleg* from professor Morten Hjorth-Jensen's example program. We then need to apporoximate our integration limmits $-\infty$ and ∞ with $-\lambda$ and $+\lambda$.

While the Legendre polynomials are defined for $x \in [-1, 1]$ the **Laguerre polynomials** are defined for $x \in [0, \infty)$. To improve our result, we rewrite the integral by changing to spherical coordinates:

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

Thus the integral becomes:

$$I = \int_0^\infty r_1^2 dr_1 \int_0^\infty r_2^2 dr_2 \int_0^\pi dcos(\theta_1) \int_0^\pi dcos(\theta_2) \int_0^{2\pi} d\phi_1 \int_0^{2\pi} d\phi_2 \frac{e^{-2\alpha(r_1+r_2)}}{r_{12}}$$

where

$$\frac{1}{r_{12}} = \frac{}{\sqrt{r_1^2 + r_2^2 - 2r_1 r_2 cos(\beta)}},$$
$$cos(\beta) = cos(\theta_1)cos(\theta_2) + sin(\theta_1)sin(\theta_2)cos(\phi_1 - \phi_2)$$

We perform the integration over $\theta \in [0, \pi]$, $\phi \in [0, 2\pi]$ and $r \in [0, \infty)$. To calculate the mesh points and weight for the angles we use the same function as for the Gauss-Legendre quadrature. To calculate the mesh points and weights corresponding to r_1 and r_2 we use professor Hjorth-Jensen's function gauss_laguerre.cpp.

Monte Carlo integration

Monte Carlo integration is a bizarre algorithm that involves generating a bunch of random numbers within the integral's limits, evaluating the function for these

numbers, and calculating an average sum multiplied with the width of the integral.

We will perform a brute force Monte Carlo integration of our system first. We use the same function as in a) and b), but in the Monte Carlo method, we make a random guess at the coordinates r_1 and r_2 using a simple uniform distribution function in C++, *uniform_real_distribution* using the *mt19937*-algorithm for random numbers.
According to figure 1, the function we will be integrating converges to 0 when the coordinates for $r \in [-3, 3]$, so we approximate the infinite limits of our integral with these values in order to make sure our random values are within this interval. This lets us approximate the integral like:

$$I \approx (3 - (-3))^6 \frac{1}{N} \sum_{i=1}^{N} f(x_{1,(i)}, y_{1,(i)}, z_{1,(i)}, x_{2,(i)}, y_{2,(i)}, z_{2,(i)})$$

For the next part of the Monte Carlo integration, we will try to improve our method by adapting the algorithm to the actual integrand and integration variables. Our function is definitely exponential, meaning we can use an exponential distribution of the form $p(x) = \frac{1}{\lambda} e^{-x/\lambda}$. In addition, we can avoid approximating ∞ by transforming our integral into polar coordinates, as we did in order to use Laguerre polynomials. This results in the following new function to be integrated:

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

The integral can then be approximated like this:

$$I \approx \frac{(2\pi)^2 \pi^2}{16N} \sum_{i=1}^{N} f(r_1, r_2, \phi_1, \phi_2, \theta_1, \theta_2)$$

In order to be able to get the best possible results with the Monte Carlo method, we want N to be as big as possible. To be able to push N as high as possible while still getting our code to run in a reasonable amount of time, we use various optimizing compiler flags, and we parallellise our code using OpenMP, letting us utilize the different cores in our computers simultaneously.

Results

Table 1 and table 2 shows the results for the two Gaussian quadratures with different values of N. The tables also shows the difference from the exact value

of the integral, 0.192765. The integral values for both of the Gaussian quadratures are plotted against different number of integration points in figure 2We have also plotted the difference from the exact value for Gauss-Legendre and Gauss-Laguerre in figure 3.

The Gauss-Legendre quadrature gave different results when we changed the limits of the integral, $-\lambda$ and λ. We found that we got the best result for Gauss-Legendre using $\lambda = 2.89$ and $N = 27$. That resulted in the value 0.192651, a difference of $1.14 \cdot 10^{-4}$ from the exact value.

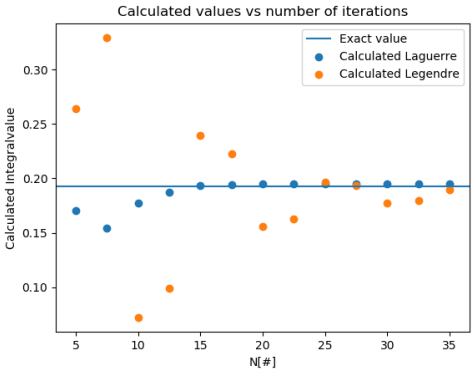

Figure 2: Calculated values of the integral against number of iterations using Gauss-Legendre and Gauss-Laguerre.

Table 1: Integration using Gauss-Legendre, varying N, with $\lambda = 3$

N	Result	Diff. from exact	Time used (s)
10	0.071980	0.120785	0.113204
15	0.239088	0.046323	1.299465
20	0.156139	0.036626	7.389662
25	0.196817	0.003052	28.340030
27	0.193524	0.000759	44.707851
30	0.177283	0.015482	87.724024

The brute force Monte Carlo-method results are presented in table 3. Our improved and adapted Monte Carlo-method results are presented in table 4.

Discussion

The Gauss-Legendre quadrature is a brute force method, and we can see from our results that is is not a very precise method. We had to try with different

Table 2: Integration using Gauss-Laguerre, variying N

N	Result	Diff. from exact	Time used (s)
10	0.177081	0.015684	0.072103
15	0.193285	0.000520	0.871607
20	0.194786	0.002021	4.997395
25	0.194804	0.002039	20.496292
27	0.193524	0.002030	31.343583
30	0.194779	0.002014	60.214597

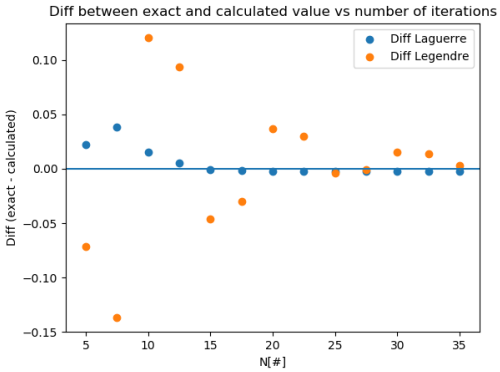

Figure 3: Difference from the exact value of the integral for Gauss-Legendre and Gauss-Laguerre against number of iterations.

Table 3: Integration using Monte Carle integration, variying N

N	Result	Diff. from exact	Time used (s)
10^5	0.244992	0.052227	0.00665
10^6	0.183088	0.009676	0.06485
10^7	0.193923	0.001159	0.66153
10^8	0.192657	0.000108	6.49192

Table 4: Integration using imporoved Monte Carle integration, variying N

N	Result	Diff. from exact	Time used (s)	Time used when parallelized (s)
10^5	0.192894	0.000129	0.0139	0.0093
10^6	0.193056	0.000291	0.1406	0.0939
10^7	0.193333	0.000568	1.3990	0.9992
10^8	0.193324	0.000049	13.993	9.9839

integration limits in order to test which limits that gave the best results. Increasing the number of mesh points above, $N = 27$ actually resulted in a less

precise approximation, which was unexpected.

By changing to spherical coordinates, we avoid approximating infinity in 4 of the 6 dimenisions of the integral, and lets us use the Gauss-Laguerre quadrature. In this method, we got better results overall, and increasing the number of mesh points improved the precision up to a point, where the precision remained almost constant. We got the best result for $N = 15$, resulting in a difference of $5.02 \cdot 10^{-4}$ from the exact value.
The results for the Monte Carlo integration is found in 3 and 4. When the results were analyzed, we observed that the integral was different every time the program was run. This was expected, because we are generating random numbers with a new seed every time we ran the program. Beause of this, we ran the program several times and found the average of those integrals. This gave us reliable results to analyze. The results show that the standard, brute force, Monte Carlo method is a relatively poor method, not presenting a satisfying result even with 10^8 random points. Our improved Monte Carlo integration, however, gave precise, satisfying results with only 10^5 integration points. Strangely, though, 10^6 and 10^7 gave increasingly imprecise results, while 10^8 yielded a more precise result than all the above, although taking 100 times longer than 10^5.
Additionally, we can see that parallelization slightly improves the running time of the improved Monte Carlo method, but not by a factor 4, which is the number of cores in the computer that ran the program, which is what we would expect.

Conclusion

Our results show that the brute force ways of numerically approximating our integral are unsatisfactory and not precise enough. When we account for and specialize our methods to our integrand and the integral limits, allowing us to use Laguerre polynomials or an exponential distribution, and spherical coordinates, we get much more precise results with fewer mesh points for the Gaussian quadrature and fewer random points in the Monte Carlo method.
Overall, comparing the results of the Gaussian quadrature using Laguerre-polynomials to the improved Monte Carlo method, we see that both give reasonably precise results, but the Gaussian method doesn't continue to improve by increasing the number of mesh points, while the improved Monte Carlo method seems to improve with an increasing number of integration points.
Time-wise, the Monte Carlo method is more efficient than the other methods, giving better precision faster, and the improved method even more so. It is also easy to parallelize, improving both precision and speed.

Bibliography

Link to our GitHub repository.