Draft version October 21, 2019
Typeset using LATEX **twocolumn** style in AASTeX62

Solving integrals by means of Guassian quadrature and Monte Carlo integration methods

Håkon Tansem, Nils-Ole Stutzer, and Bernhard Nornes Lotsberg

Abstract

We have implemented, tested and compared four different numerical methods for solving integrals. The methods implemented were a brute force Gauss-Legendre quadrature, an improved Gauss-Laguerre quadrature as well as a brute force and an improved Monte Carlo integration method. These methods were tested on an integral computing the expectation value for the potential of two interacting electrons in a simplified helium atom with a known analytical solution. Our results indicate that for the integral, the relative error for the improved Gaussian quadrature method steadily decreases as the resolution of the computational grid increases. The brute force method on the other hand produces a relative error fluctuating unpredictably as the resolution of the grid increases making it inferior to the improved method when solving this integral. The Monte Carlo methods were also calculated using parallelization indicating that the CPU time decreases proportionally to the amount of CPU threads used in the calculation. When taking the relative error into account our results indicate that for the integral, the improved Monte Carlo method is the preferred method of choice followed by the Gauss-Lagurre quadrature method.

The source codes for this paper can be found at https://github.com/bernharl/FYS4150-Project-3.

1. INTRODUCTION

When solving problems in science and mathematics an ever recurring problem is to solve integrals. Integrals are found in many problems in various fields such as physics and economics. In this paper we consider ways of solving an example of a six dimensional expectation value problem from quantum mechanics in several different ways. We consider a brute force Gauss-Legendre quadrature, an improved Gauss-Laguerre quadrature, a brute force monte carlo integration and a monte carlo integration with importance sampling as shown by Press et al. (2007) and Jensen (2015). The resulting integrals are compared to the analytical solution and the run times are compared, in order to find which method is most efficient.

In the theory section we present needed theory wich we discuss how to implement in the method section. The results are presented in the results section and discussed in the discussion section.

2. THEORY

2.1. *The integral*

Before stating the needed integration methods we present the integral to integrate. Assuming the wave function of two electrons can be modelled as a single-particle wave function in a hydrogen atom, the wave function of the ith electon in the 1 s state is given by

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

where the dimensionless position

$$\vec{r}_i = x_i\hat{e}_x + y_i\hat{e}_y + z_i\hat{e}_z \tag{2}$$

with orthogonal unit vectors $\hat{e}_i$.

The distance from the origin is expressed as $r_i = \sqrt{x_i^2 + y_i^2 + z_i^2}$ and we let the parameter $\alpha = 2$ corresponding to the charge of a helium $Z = 2$ nucleus. Then the ansatz for the wave function for two electrons is given by the product of the two $1s$ wave functions

$$\Psi(\vec{r}_1, \vec{r}_2) = e^{-\alpha(r_1+r_2)}. \tag{3}$$

We want to find the expectation value of the correlation energy between electrons which repel each other by means of the Coulomb interaction as

$$\langle \frac{1}{\vec{r}_1 - \vec{r}_2} \rangle = \int d\vec{r}_1 d\vec{r}_2 e^{-2\alpha(r_1+r_2)} \frac{1}{|\vec{r}_1 - \vec{r}_2|}. \tag{4}$$

This integral has an analytical solution $5\pi^2/16^2$, which we can compere to the numerical results.

2.2. *Brute Force Gauss-Legendre Quadrature*

The following theory and derivations of the integration methods presented follow closely (Jensen 2015, Ch. 5.3).

The essence of Gaussian Quadrature (GQ) is to approximate an integral

$$I = \int f(x)dx \approx \sum_{i=1}^{N} \omega_i f(x_i), \tag{5}$$

for the weights ω_i and grid points x_i. The grid points and weights are obtained through the zeros of othogonal polynomials. These polynomials are orthogonal on some interval, for instance $[-1, 1]$ for Legendre polynomials. Since we must find N grid points and weigths, we must fit $2N$ parameters. Therefore we must approximate the integrand $f(x)$ with a polynomial of degree $2N-1$, i.e. $f(x) \approx P_{2N-1}(x)$. Then the integral

$$I \approx \int P_{2N-1}(x)dx = \sum_{i=0}^{N-1} P_{2N-1}(x_i)\omega_i. \tag{6}$$

GQ can integrate all polynomials of degree up to $2N-1$ exactly, we thus get an equality when approximating the integral of P_{2N-1}.

If we choose to expand the polynomial P_{2N-1} in terms of the Legendre polynomials L_N, we can through polynomial division write

$$P_{2N-1} = L_N(x)P_{N-1}(x) + Q_{N-1}(x), \tag{7}$$

where P_{N-1} and Q_{N-1} are polynomials of degree $N-1$. We can thus write

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

$$= \int_{-1}^{1} Q_{N-1}(x)dx, \tag{9}$$

where the last equallity is due to the orthogonallity between $L_N(x)$ and $P_{N-1}(x)$. Furthermore, $P_{2N-1}(x_k) = Q_{N-1}(x_k)$ for the zeros x_k $(k = 0, 1, 2, \ldots, N-1)$ of L_N, we can fully define the polynomial $Q_{N-1}(x)$ and thus the integral. The polynomial $Q_{N-1}(x)$ can further be expanded in terms of the Legendre basis

$$Q_{N-1}(x) = \sum_{i=0}^{N-1} \alpha L_i(x). \tag{10}$$

When integrating this we get

$$\int_{-1}^{1} Q_{N-1}(x)dx = \sum_{i=0}^{N-1} \alpha_i \int_{-1}^{1} L_0(x)L_i(x)dx = 2\alpha_0, \tag{11}$$

where we insert that the first Legendre polynomial is normalized to $L_0 = 1$ and utilize the orthogonality relation of the Legendre basis.

Since we know the value of $Q_{N-1}(x)$ at the zeros of $L_N(x)$, we can rewrite (10)

$$Q_{N-1}(x_k) = \sum_{i=0}^{N-1} \alpha L_i(x_k). \tag{12}$$

The resulting matrix $L_i(x_k) = L_{ik}$ has linearly independent columns due to the Legendre polynomials being linearly independent as well. Therefore the matrix L_{ik} is orthogonal, i.e.

$$L^{-1}L = I. \tag{13}$$

We multiply both sides of (12) by $\sum_{i=0}^{N-1} L_{ij}^{-1}$, so that

$$\alpha_k = \sum_{i=0}^{N-1} (L^{-1})_{ki} Q_{N-1}(x_i). \tag{14}$$

This result in addition to the approximation of the integral gives

$$I \approx \int_{-1}^{1} P_{2N-1}(x)dx = \int_{-1}^{1} Q_{N-1}(x)dx = 2\alpha_0 \tag{15}$$

$$= 2\sum_{i=0}^{N-1} (L^{-1})_{0i} P_{2N-1}(x_i). \tag{16}$$

Here we can clearly see that the weights $\omega_i = 2(L^{-1})_{0i}$ and the mesh points x_i are the zeros of $L_N(x)$. Finding the weights is now simply a matrix inversion problem and the mesh points can be found by for instance Newton's method. Thus (16) is on the same form as (5), yielding an approximation of I. When performing an integral with more general limits $[a, b]$, one can now perform a simple change of variables $\tilde{x} = \frac{b-a}{2}x + \frac{b+a}{2}$, to accomodate for this. Note that when considering the Gauss-Legendre approach, and we integrate (4) using cartesian coordinates, we must let the limits of all six cartesian itegrals $a = -\lambda$ and $b = \lambda$ where $\lambda \to \infty$.

When integrating a six dimensional integral like the quantum mechanical problem stated earlier, we can simply use Gauss-Legendre quadrature on all six integrals in a brute force way. The integral becomes

$$I = \int d\vec{r}_1 d\vec{r}_2 e^{-2\alpha(r_1+r_2)} \frac{1}{|\vec{r}_1 - \vec{r}_2|} \tag{17}$$

$$\approx \sum_{i,j,k,l,m,n=0}^{N-1} \omega_i\omega_j\omega_k\omega_l\omega_m\omega_n f, \tag{18}$$

where $f = f(x_1^i, y_1^j, z_1^k, x_2^l, y_2^m, z_2^n)$ denotes the integrand and each cartesian variable x_i has the same weights ω due to identical integration limits on the six integrals.

2.3. *Improved Gaussian Quadrature*

The above mentioned Gauss Laguerre quadrature is in fact quite inaccurate in the case of our integral. In order to improve on the method, one can use a different orthogonal polynomial basis. In our case since we have an integral on the form

$$I = \int_0^\infty f(x)dx = \int_0^\infty x^2 e^{-x} g(x)dx \tag{19}$$

it is in fact more efficient to use Laguerre polynomials as opposed to Legendre polynomials. Then when finding an approximation of the integral the x^2e^{-x} factor of the integrand is absorbed into the weights ω_i, making the weights more suitable for this specific integrand. The derivation of the weights is however not shown here as it is completely analogous to the derivation shown for the Gauss-Legendre quadrature.

In order to transorm our integrand to a form resembling (19) we need to transform from cartesian to spherical coordinates, i.e. $(x, y, z) \to (r, \theta, \phi)$ where $r \in [, \infty)$, $\theta \in [0, \pi]$ and $\phi \in [0, 2\pi]$. In spherical coordinates we get that the differential and the relative distance between the electrons become

$$d\vec{r}_1 d\vec{r}_2 = r_1^2 r_2^2 dr_1 dr_2 \sin(\theta_1)\sin(\theta_2) d\theta_1 d\theta_2 d\phi_1 d\phi_2 \tag{20}$$

$$|\vec{r}_1 - \vec{r}_2| = \sqrt{r_1^2 + r_2^2 - 2r_1 r_2 cos(\beta)}, \tag{21}$$

where $cos(\beta) = cos(\theta_1)\cos(\theta_2) + \sin(\theta_1)\sin(\theta_2)\cos(\phi_1 - \phi_2)$. Next we introduce the change of variables $u = \alpha r \implies du = \alpha dr$ so that the integral is on the form

$$I = \frac{1}{32\alpha^5}\int_0^{\pi}\int_0^{\pi}\int_0^{2\pi}\int_0^{2\pi}\int_0^{\infty}\int_0^{\infty} f du_1 du_2 d\theta_1 d\theta_2 d\phi_1 d\phi_2, \tag{22}$$

where $f = f(u_1, u_2, \theta_1, \theta_2, \phi_1, \phi_2) = \frac{\sin(\theta_1)\sin(\theta_2)u_1^2 u_2^2 e^{-(u_1+u_2)}}{\sqrt{u_1^2+u_2^2-2u_1u_2cos(\beta)}}$ is the new integrand. Now, for the radial part (u_1 and u_2) a Gauss-Laguerre approach is used, while we use Gauss-Legendre quadrature for the angular part.

2.4. *Brute Force Monte Carlo Integration*

An integration method frequently used, especially when computing multi-dimensional integrals as its error remains independent of the dimensionality of the integral (Jensen 2015, p. 343-344), is the Monte Carlo integration. This integration method approximates the integral with an expectation value. Consider for instance an integral

$$I = \int_a^b f(x)dx = (b-a)\int_a^b \frac{f(x)}{b-a}dx \tag{23}$$

$$= (b-a)\int_a^b f(x)p(x)dx, \tag{24}$$

where we let $p(x) = \frac{1}{b-a}$ be the uniform probability density function PDF for stochastic variables $x \in [a, b]$. Since we know that an expectation value

$$\langle f(x)\rangle = \int_a^b f(x)p(x)dx \approx \frac{1}{N}\sum_{i=0}^{N-1} f(x_i), \tag{25}$$

where we used that the expectation value of $f(x)$ is approximately the average of the $f(x_i)$-s where the x_i-s are drawn from the distribution $p(x)$, for large enough sample size N.

Note that we implicitly performed a mapping in this case. Because uniform distribution by default only returns values $y \in [0, 1]$, we change the variable to $x = a + (b-a)y$ so as to draw values x_i from a uniform distribution between a and b.

The integral can thus be approximated by

$$I \approx (b-a)\langle f(x)\rangle \approx \frac{b-a}{N}\sum_{i=0}^{N-1} f(x_i). \tag{26}$$

In order to get an estimate for the accuracy of the integration we can calculate the variance defined as

$$\sigma_f^2 = (b-a)\left(\frac{1}{N}\sum_{i=0}^{N-1} f^2(x_i) - \left(\frac{1}{N}\sum_{i=0}^{N-1} f(x_i)\right)^2\right) = (b-a)\,(\langle \tag{27}$$

The standard deviation is defined as $\sigma_N = \sigma_f/\sqrt{N}$ according to the central limit theorem for N samples (Jensen 2015, p. 343). The variance for a multi-dimensionl case is computed in a completely analogous way. As a Monte Carlo integration is a statistical experiment, the variance σ^2 is a measure of the spread from the mean of the integral approximation. We thus want to minimize this quatity.

When solving (4) using cartesian coordinates, we use the same integration limits as in the Gauss-Legendre quadrature. The we simply get a six-dimensional expectation value so that

$$I = (b-a)^6\langle f\rangle \approx \frac{(b-a)^6}{N}\sum_{i=0}^{N-1} f(x_1^i, y_1^i, z_1^i, x_2^i, y_2^i, z_2^i), \tag{28}$$

where the cartesian coordiantes x^i are all drawn from a uniform distribution for $x^i \in [-\lambda, \lambda]$. The variance is calculated completely analogously to the one-dimensional case.

2.5. *Improved Monte Carlo Integration*

One way to improve the Monte Carlo Integration shown in the previous subsection, i.e. to reduce its variance, is to use a different PDF, that fits the shape of the integrand better, to draw the samples from.

If we consider the quantum mechanical integral in spherical coordinates, as shown previously, we recognize the e^{-u} factors as an exponential distribution. Thus we let $p(y) = e^{-y}$ denote the exponential distribution. Using conservation of probability under change of variable, we set $p(y)dy = \exp(-y)dy = p(x)dx = dx$ for the uniform

PDF $p(x)$ with $x \in [0,1]$. If we integrate this cumulative distribution we find

$$x(y) = \int_0^y \exp(-\xi)d\xi = 1 - \exp(-y), \tag{29}$$

which we can invert to get the mapping $y(x) = -\ln(1-x)$ from the uniform to the exponential PDF. Now $u = y \in [0\infty)$ is used for the radial distance. We can now absorb the exponential part of the integral into the expectation value so that

$$I = \int_0^\infty f(u)du = \int_0^\infty \frac{f(u)}{p(u)}p(u)du \tag{30}$$

$$= \int_0^1 g(u(x))dx \approx \frac{1}{N}\sum_{i=0}^{N-1} g(u(x_i)), \tag{31}$$

where we use that $p(u) = \exp(-u)$ and $p(u)du = dx$ to change variables. The samples $u(x_i)$ are now drawn from the exponential distribution $p(u)$. This is what is called importance sampling, which results in a lower variance than using the brute force Monte Carlo integration, because we sample from a distribution which fits the shape of the integrand better.

Note that the integral we want to estimate in spherical coordinates only has limits $[0,\infty)$ for the two radial coordiantes u_1 and u_2. We must thus sample from the uniform distribution for the angular integrals, as the angles exist within the finite limits $[0,\pi]$ and $[0,2\pi]$, while the radial integral should utilize importance sampling from the exponential distribution.

The final integral looks as follows

$$I \approx \frac{1}{N}\frac{\pi^4}{8\alpha^5}\sum_{i=0}^{N-1}\frac{\sin(\theta_1^i)\sin(\theta_2^i)(u_1^i)^2(u_2^i)^2}{\sqrt{(u_1^i)^2+(u_2^i)^2-2u_1^iu_2^icos(\beta_i)}}, \tag{32}$$

where the index i simply represents that the corresponding samples are drawn from their corresponding PDFs. The factor $\frac{\pi^4}{8\alpha^5}$ is a Jacobi determinant from the change of variables.

3. METHOD

When approximating the integrals, one may encounter singularities in the integrand when $|\vec{r}_1 - \vec{r}_2|$ becomes too small, which is due to the discretization. In order to handle these singularies we simply let the integrand $f = 0$ when this happens.

Next when implementing the two GQ integrators, the integration sum approximating the six-dimensional integral are calculated in a sixfold loop. When calculating the integration weights and grid points we use the algorithms presented in (Press et al. (2007)) (implemented in the `weights.cpp`-file). In the brute force cartesian GQ approach as well as the angular integrals in the improved GQ we compute the weights and meshpoints using the provided `gauleg`-function, since the integration limits are finite, while the weights and mesh points of the radial integral are produced using the provided `gauss_laguerre`-function. The integrals are calulated with the two GQ methods for different grid sizes N and then compared. In order to produce a satisfactory result using the brute force Gauss-Legendre quadrature we tried different grid sizes N and approximation of infinity λ.

When implementing the Monte Carlo integrations, we produce the needed psuedo-random numbers using a Mersenne-Twister algorithm, as it has a sufficiently large period. Furthermore the mapping from the uniform PDF with $x \in [0,1]$ to the uniform PDF with $x \in [a,b]$ and the exponential distributions are done implicitly inside the C++ `random`-packages used. The Monte Carlo integrals are computed for different sample sizes N, different degrees of parallelization using OpenMP and different compiler flags, for comparison.

4. RESULTS

The results were produced running on a MacBook Pro (macOS Catalina 10.15) with 8GB RAM using a dual core Intel Core i5-7360U 2.3GHz CPU with four threads. The program was compiled using OpenMP for the parallelization with the Apple Clang compiler version 11.0.0 (clang-1100.0.33.8).

For the results listed below we found that $\lambda = 1.5$ was a sufficient approximation for infinity when using grid sizes $N \sim 20$ to $N \sim 30$ when using brute force Gaussian quadrature as described in section 2.2. This choice for λ in addition to $\lambda = 3$ was also tested when calculating the brute force Monte Carlo method as described in section 2.4. When calculating the integral, the relative error was computed for the integral given by (4) using the brute force method, as described in section 2.2. The relative error was also calculated for the integral in spherical coordinates given by (22) using the improved method, as described in section 2.3. These results are shown in Figure 1. In the figure a line is drawn to illustrate when the method achieves three digit precision. The CPU time for the improved Gaussian quadrature method with grid resolution $N = 30$ was measured to be approximately 45000ms depending on background processes.

The relative error was also compared for both the brute force and the improved Monte Carlo methods described in section 2.4 and 2.5 respectively. This is shown in Figure 2. For the same calculations using both Monte Carlo methods, the variance was also plotted. This is result is illustrated in Figure 3.

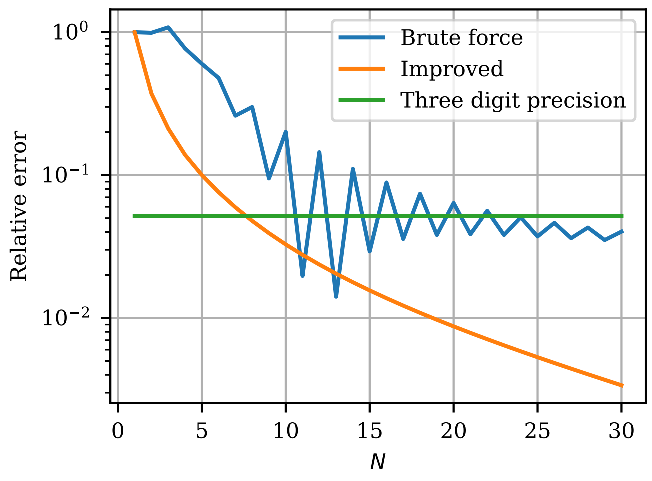

Figure 1. Figure showing the relative error for the brute force Gaussian quadrature and the improved Gaussian quadrature methods, described in section 2.2 and 2.3 respectively, as a function of grid resolution N. A line is drawn to illustrate when the integral reaches three digit precision.

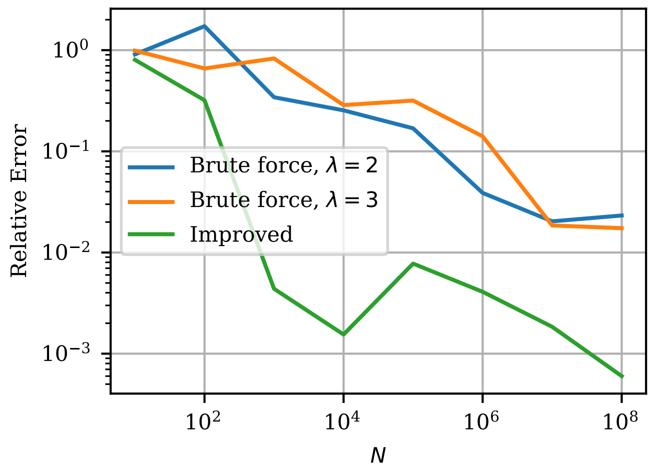

Figure 2. Figure showing the relative error for the brute force ($\lambda = 1.5$ and $\lambda = 3$) and the improved Monte Carlo methods, described in section 2.4 and 2.5 respectively, as a function of the sample size N.

The CPU time for both the brute force and the improved Monte Carlo methods were calculated using no parallelization and parallelization with two threads. This is shown in Figure 4. The three different compiler flags -O1, -O2 and -O3 were also compared to study their impact on the CPU time, which is shown in Figure 5. This result was produced using no parallelization.

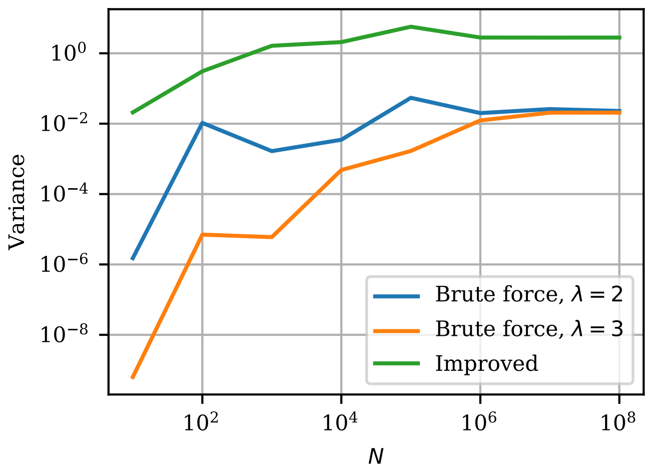

Figure 3. Figure showing the variance for the brute force ($\lambda = 1.5$ and $\lambda = 3$) and the improved Monte Carlo methods, as described in section 2.4 and 2.5 respectively, as a function of the sample size N.

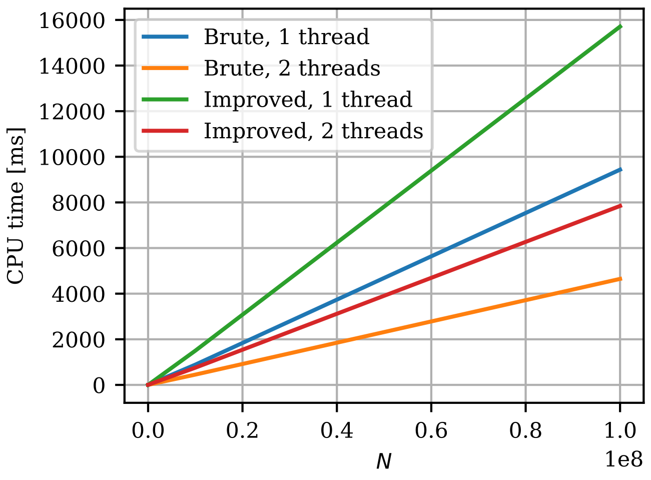

Figure 4. Figure showing the CPU time for the brute force and the improved Monte Carlo methods, described in section 2.4 and 2.5 respectively, as a function of the sample size N. For both methods the CPU time was compared both unparallelized and parallelized with two threads. Here $\lambda = 3$ was used for the brute force method.

5. DISCUSSION

When computing the integral (4) using brute force Gaussian quadrature, as mentioned in the section 4, when using a grid size of $N \sim 20$ to $N \sim 30$, we found that $\lambda = 1.5$ was a sufficient approximation for infinity. This achieved a three digit precision. However, one can in Figure 1 see that the relative error of the Gauss-Legendre quadrature varies rapidly over the grid size N. Thus, the brute force approach yields an unsatisfactory precision as it behaves

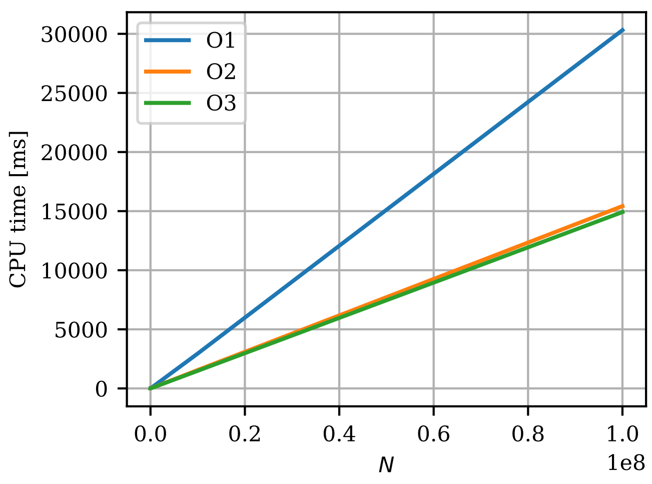

Figure 5. Figure showing showing the CPU time for the improved Monte Carlo method, described in section 2.5, as a function of the sample size N using the different compiler flags -O1, -O2 and -O3. These results were produced using no parallelization.

unpredictably. Therewhile, the relative error of the improved Gaussian quadrature decreases for increasing grid size N. Thus the improved Gauss-Laguerre quadrature method provides a far more predictable decrease in the relative error, making it the Gaussian quadrature of choice for the problem at hand. The differences between these two methods are consistant with the theory, as the Gauss-Laguerre quadrature constructs more suitable weights fitting the integrand better. Also we need to approximate infinity when computing (4) in cartesian coordiantes using Gauss-Legendre quadrature, while the Gauss-Laguerre quadrature using spherical coordiantes absorbs the infinity term in the weights, effectively circumventing this potential source of error of an insufficient infinity approximation.

When comparing the relative error for the brute force and improved Monte Carlo methods, our results, as shown in Figure 2, indicate that for both values of λ the improved method reaches a significantly higher precision for higher sample sizes N. This shows that when we draw from a probability distribution suited to the integrand our results improve. When comparing the variance, as shown in Figure 3, for the brute force method with the different values for λ and the improved method, one can see that the variance of the improved method begins to stabilize when the sample size reaches approximately $N \sim 10^4$. The brute force methods with different values for λ begins to stabilize for a much higher sample size $N \sim 10^6$. One can also see that the variance is sensitive to different choices of λ. Therefore it seems as if the improved method behaves more stably and predictably as there is one less parameter to tune for the integral at hand. Note that the variance for the improved method stabilizes at a higher value than for the brute force methods. One would expect the variance for both methods to stabilized at the same order of magnitude. In our results this is not the case as the improved method stabilizes at a higher order of magnitude. We were unable to explain this result as the relative error seems to behave reasonably. However the behaviour of the variance for both methods is what one would expect. This discrepancy suggests further research is needed.

When comparing the CPU time for the improved Gaussian quadrature method with grid resolution $N = 30$ to the CPU time for the improved Monte Carlo method with sample size $N = 10^8$, both unparallelized, one can see that the CPU time for the improved Monte Carlo method is significantly better. If we compare the relative error one can also see, from figures 1 and 2, that the relative error for the improved Monte Carlo method is a lot smaller. This indicates that the improved Monte Carlo method provides a better accuracy for a shorter CPU time. It is also worth noting that the results presented originate from a single run of the program. CPU time can be sensitive to background processes running on the processor. This can cause the results to vary slightly for different runs. A further improvement would be to take the average over multiple runs. Another source for inaccuracy is the approximation of infinity as $\lambda = 1.5$

The parallelization provided a significant improvement to the CPU time for the brute force and improved Monte Carlo methods. From Figure 4 one can see that the parallelization with two threads approximately halves the CPU time compared to doing the same calculation with one thread. From the same figure, one can also see that the CPU time for the brute force method is lower than the CPU time for the improved method. This shows that allthough increasing precision, drawing from an exponential distribution comes with a drawback in form of calculations taking longer time. This may be due to the logarithm present in the mapping from the uniform distribution to the exponential distribution, shown in section 2.5.

When studying the solution for the improved Monte Carlo method using no parallelization and different compiler flags, our results indicate that the compiler flags had an impact on the CPU time. This is shown in Figure 5. The difference between the O1 and O2 flags is significant and provides a speed increase by approximately a factor 2. The difference between O2 and O3 however is minimal. In our implementation of the methods, we included an if-statement to set $f = 0$ if $|\vec{r}_1 - \vec{r}_2| = 0$ to avoid divison by zero as described in section 3. For the compiler flags to be able to optimize the code, the code should be predictable. If-statements make the code unpredictable in the sense that there is no way to predict when the condition occurs. This makes it hard for the compiler to vectorize the loops. This may also suggest that the difference between O2 and O3 is not significant for this purpose. A further improvement could be to rewrite the integral given by 4 on a form that does not contain singularities both to improve CPU time when using compiler flags as well as excluding potential errors from setting $f = 0$.

6. CONCLUSION

We have implemented four different numerical integration methods to find the expectation value of the potential of two interacting electrons in a simplfied helium atom model. For the brute force Gauss-Legendre method in cartesian coordianes, the relative error was found to fluctuate rapidly over increasing grid size N. Therewhile the improved Gauss-Laguerre approach in spherical coordinates provided a relative error steadily decreasing for increasing N, rendering it the GQ of choice for the problem at hand. When Comparing the brute force and improved Monte Carlo methods, we found that the improved Monte Carlo method had a longer CPU time, however the variance stabilizes for lower values of sample size N. The variance for the improved and brute force methods stabilized at different orders of magnitude. This discrepancy should be looked into in more detail. When comparing the improved GQ and Monte Carlo integration with the highest grid and sample size tested, we found the Monte Carlo integration to provide the best accuracy as well as having a significantly shorter CPU time.

When comparing the run time of the brute force and improved Monte Carlo integrations for different sample sizes and different degrees of parallelization, we found that the improved Monte Carlo method in general had a longer CPU time than the brute force method. However, a notable speed up was noticed when the method was parallelized using one and two threads.

Last we tested the CPU time of the improved Monte Carlo method with different compiler flags without parallelization, for different grid sizes. We found a significant speed increase by a factor 2 between the O1 and O2 flags. However for our problem the O2 and O3 flags proved almost identical. This could be due to the if-statement in the integrand preventing efficient vectorization or the O3 flag could prove excessive for this problem.

In order to improve our results one could rewrite the integrand in a more convenient manner, so as to circumvent singularities. Also one could run the Monte Carlo integrals and measure the CPU time several times for each sample size N and take the mean to get a more accurate results.

REFERENCES

Department of Physics. 2019, Project 3, numerical integration, deadline October 21, Norway: University of Oslo, https://github.com/CompPhysics/ComputationalPhysics/tree/master/doc/Projects/2019/Project3/pdf, Visited: 19.10.2019

Jensen, M. H. 2015, Computational Physics, Lecture notes Fall 2015, , , https://github.com/CompPhysics/ComputationalPhysics/blob/master/doc/Lectures/lectures2015.pdf, Visited: 6.9.2019

Press, W., Teukolsky, S., Vetterling, W., & Flannery, B. 2007, Numerical Recipes 3rd Edition: The Art of Scientific Computing, Cambridge University Press