Draft version October 18, 2019
Typeset using LaTeX **twocolumn** style in AASTeX62

Solving Eigenvalue Problems by Means of the Jacobi Algorithm

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

Abstract

Solving integrals is an essential part of science and mathematics. 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. Our method was tested on an integral computing the expectation value of the potential of two interacting electrons in a simplified helium atom with a known analytical value. Our results indicate that for our 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 for this integral. For the Monte Carlo methods we show that adapting the probability distribution to the given integral dramatically improves the variance, thereby gaining a significant increase in precision. The Monte Carlo methods was 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 our integral, the improved Monte Carlo method is preferred 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 all sorts of manners, e.g. when computing expectation values in quantum mechanics. In this paper we will consider ways of solving an example of a six dimensional expectation value problem from quantum mechanics integral in several different ways. We will 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 will be 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 used 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 $r_i = \sqrt{x_i^2 + y_i^2 + z_i^2}$ and we let the parameter $\alpha = 2$ corresponding to the charge of a helium $Z = 2$. Then the ansats for the wave function for two electrons is given by the product of the two 1 s wave functions

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

We now 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 later compare numerical results to.

2.2. *Brute Force Gauss-Legendre Quadrature*

The following theory and derivations of the integration methods presented follows 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 some weights ω_i and grid points x_i. The grid points and wights are obtained through the zeros of othogonal polynomials. These polynomials are orthogonal on some intervale, 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 $f(x)$ by 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 $2N-1$ exactelly, we thus get an equallity when approximating the integral of P_{2N-1}.

If we choose to expande the polynomial P_{2N-1} in terms of the Legendre polynomials L_N, we can through polynomial devision 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 thus 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 then 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 meshpoints x_i are the zeros of $L_N(x)$. Finding the weights is now simply a matrix inversion problem and the meshpoints can be found by for instance Newtons method. Thus (16) is on the same form as (5), yielding an approximation to 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. Then the integral simply 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 meansioned Gauss Laguerre quadrature is in fact quite inacurate 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 way more efficient to use Laguerre polynomials as oppose to Legendre polynomials. Then when finding an approximation to the integral the x^2e^{-x} factor of the integrand is absorbed into the weights ω_i. Thus the approximation becomes more accurate, as we get more suitable weights. 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 resembeling (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 the differential 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 $f = f(u_1, u_2, \theta_1, \theta_2, \phi_1, \phi_2) = \frac{\sin(\theta_1)\sin(\theta_2) u_1^2 u_2^2 e^{-(u_1+u_2)}}{\sqrt{u_1^2 + u_2^2 - 2u_1 u_2 cos(\beta)}}$ 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, espessially when computing multi-dimensional integrals as its error remains constant for any higher dimension, is the Monte Carlo integration. In this integration method one approximates the integral by 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 implisitly 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

$$\sigma^2 = \frac{1}{N}\sum_{i=0}^{N-1} f^2(x_i) - \left(\frac{1}{N}\sum_{i=0}^{N-1} f(x_i)\right)^2 = \langle f^2 \rangle - \langle f \rangle^2. \tag{27}$$

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 whant 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 analogous 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 PDE, 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 absorbe 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 used that $p(u) = \exp(-u)$ and that $p(u)du = dx$ 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 simply sample from a distribution which fits the integrand better.

However, 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 finit limits $[0, \pi]$ and $[0, 2\pi]$, while the radial integral should utilize importance sampling from the exponential distribution.

The final integral then looks as follows

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

where the index i simply represents that the corresponding samples are drawn from their respective PDF.

3. METHOD

When approximating the integrals, one may encounter singularities in the integrand when $|\vec{r}_1 - \vec{r}_2|$ becomes too small, which 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 simply 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. The weights and meshpoints of the radial integral therewhile are produced using the provided `gauss_laguerre`-function. The integrals where 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 tryed different grid sizes N and infinity approximations λ.

When implementing the Monte Carlo integrations, we produce the needed psuedo-random numbers using a Mersenne-Twister algorithm, as it has a suffficiently 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 implisitly inside the C++ `random`-packages used. The Monte Carlo integrals where 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 Mojave 10.14.6) 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 gcc compiler version (Homebrew GCC $9.2.0\_1$) $9.2.0$.

For the results listed below we found that $\lambda = 1.5$ was a sufficient approximation 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 remained the same when calculating the brute force Monte Carlo method as described in section 2.4. When calculating the integral, the relative error was calculated for the integral given by (**??**) 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 when the relative error surpasses 10^{-2} 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. It turned out to be approximately 51000ms

The relative error was also compared for both the brute force Monte Carlo method and the improved Monte Carlo method described in section 2.4 and 2.5 respectively. This is shown in figure 2. For the same calculations using both

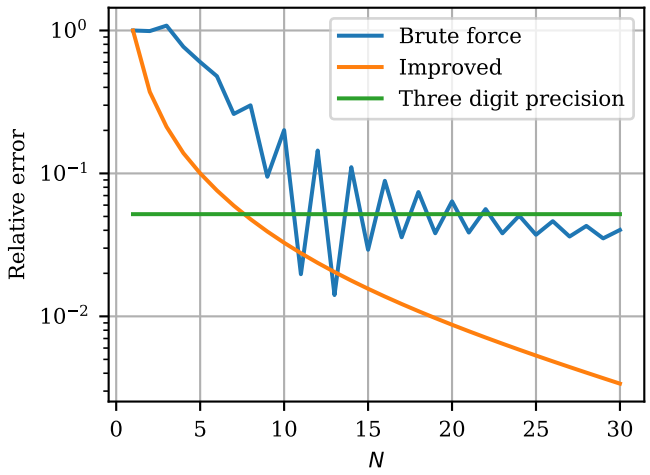

Figure 1. Figure showing the relative error for the brute force gaussian quadrature and the improved gaussian quadrature methods, described in section 2.2 and 2.3 respectively, as a function of grid resolution N. A line is drawn when the relative error is 10^-2 to illustrate when the integral reaches three digit precision.

Monte Carlo methods, the variance was also plotted. This is result is illustrated in figure 3.

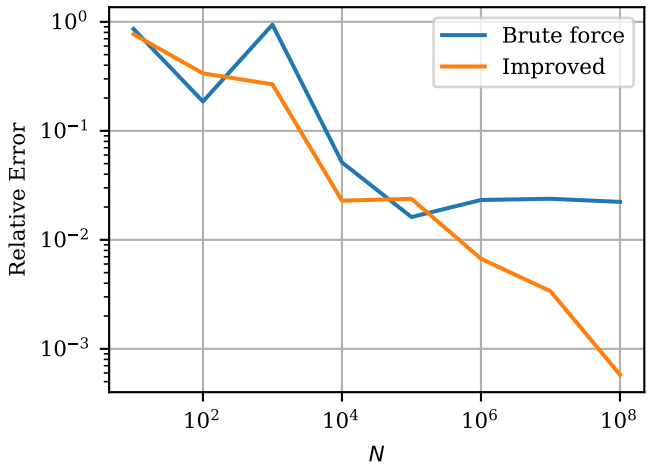

Figure 2. Figure showing the relative error for the brute force and the improved Monte Carlo methods, described in section 2.4 and 2.5 respectively, as a function of the sample size N.

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

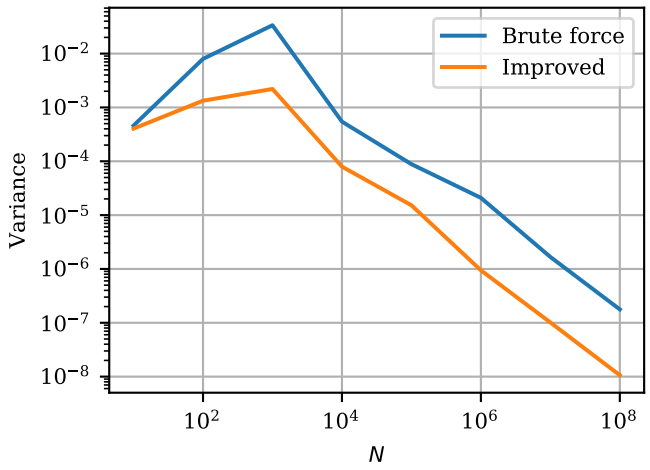

Figure 3. Figure showing the variance for the brute force and the improved Monte Carlo methods as described in section 2.4 and 2.5 respectively as a function of the sample size N.

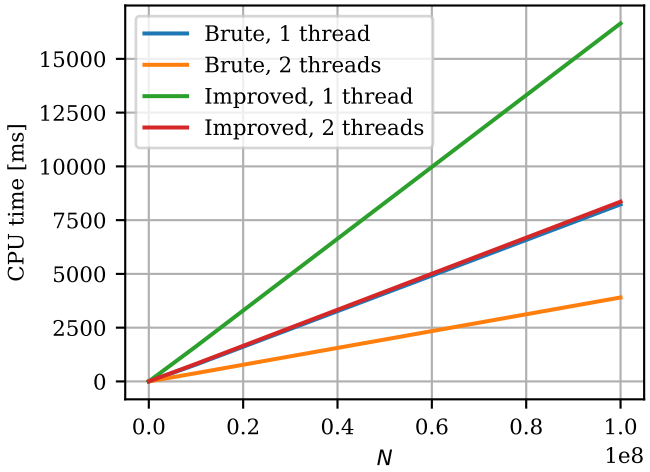

Figure 4. Figure showing the CPU time the brute force and the improved Monte Carlo methods, described in section 2.4 and 2.5 respectively, as a function of the sample size N. For both methods the CPU time was compared both unparallelized and parallelized with two threads.

5. DISCUSSION

When comparing the relative error for the brute force and improved Monte Carlo methods, our results, as shown in figure 2, indicate that 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 integral our results improve. This can also be seen by studying the variance for the brute force and improved methods, as shown in figure 3. Here we see that the Variance drops alot faster for the improved method and stays approximately an order of magnitude

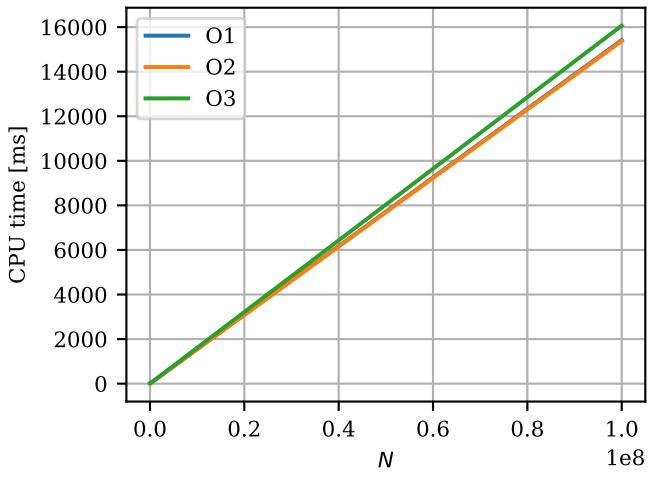

Figure 5. Figure showing showing the CPU time for the improved Monte Carlo method, described in section 2.5, as a function of the sample size N using the different compiler flags -O1, -O2 and -O3. These results were produced using parallelization with four threads.

lower than the variance for the brute force method. When comparing the CPU time for 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 error for the improved Monte Carlo method has a lot shorter CPU time. 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 iteration 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.

The paralellization 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 has a lower CPU time than 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. When studying the solution for the improved Monte Carlo method using four threads and different compiler flags, our results indicated that the flags had a low impact on the CPU time. This is shown in figure 5. 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, it should be predictable. If-statements make the code unpredictable in the sense that there is no way to predict when the condition occurs. A further improvement would be to rewrite the integral given by 4 on a form that does not contain singularities.

6. CONCLUSION

7.1014in

REFERENCES

Jensen, M. H. 2015, Computational Physics, Lecture notes Fall 2015

of Physics, D. 2019, Project 3, numerical integration, deadline October 21, Norway: University of Oslo

Press, W., Teukolsky, S., Vetterling, W., & Flannery, B. 2007, Numerical Recipes 3rd Edition: The Art of Scientific Computing, Cambridge University Press. https://books.google.no/books?id=1aAOdzK3FegC