FYS3150/4150 Project 3

Kristian Gjestad Vangsnes, Tore Klungland and Laura Beghini

Abstract

We examine the Gaussian quadrature and Monte Carlo methods to compute a six-dimensional integral which represents the ground state correlation energy between two electrons in a helium atom. We used both Gauss-Legendre and Gauss-Laguerre quadrature. The second method gives more precise results because Laguerre polynomials approximate better the integrating function than Legendre ones and in this case we do not need to approximate the integration interval. With those algorithms the result is reproduced with three leading digits with 45 mesh points. Although it leads to correct results, we find that for a high-dimensional integral, if high precision is desired, Gaussian quadrature quickly becomes rather inefficient. This because its computational time is proportional to N^n where n in the dimension of the integral and N the number of mesh points. Indeed it is about a thousand times slower than the faster Monte Carlo algorithm. Monte Carlo computational time is proportional to N. However this kind of integration need far more mesh point because it is based on the law of large numbers. The brute force Monte Carlo integration is as effective as the Gaussian quadrature. This is because we have to use more than 10^9 sample to make the result converge at the level of the third leading digit. The improved Monte Carlo integration uses the importance sampling, parallelization and vectorization so we get a better approximation of the function and a fast and precise result.

All programs used to produce and plot these results, as well as output files containing the results referred to in this text, can be seen at `https://github.com/Krissvang/Computational-Physics-group`.

UiO : University of Oslo

University of Oslo
Norway
October 21, 2019

Contents

1 Introduction

Integrals are useful to solve many of the most powerful equations used in physics today. This is true both for the classical world and the quantum one. Thus, it is important to know how to compute them. The aim of this project is to integrate with different methods a six-dimensional integral which is used to determine the ground state correlation energy between two electrons in a helium atom. We used both the Gauss quadrature and the Monte Carlo method. Both these methods approximate the integral with a discrete sum of terms. Each term is the product between the weight and the function calculated in the chosen mesh point. For the Gaussian Quadrature method the integration points are chosen to be the roots of an orthogonal polynomial of degree N. The integration weights are given by the inversion of a matrix defined by the orthogonal polynomials we have chosen. In this project we will use both the Legendre and Laguerre polynomials. In Monte Carlo integration, the integral of a function over a given area is approximated as the mean value of the function at N randomly selected points within this area. Thus the integration weights in this case will be all equal to one on the number of mesh points. The computed integral will converge to the expected value due to the law of large numbers [1]. For the Monte Carlo approach, we start with a brute force method and we tried to improve it by changing the distribution of the random mesh points. This process is called importance sampling.

In this text we will first outline these methods and explain how they can be implemented for the relevant problem. We will then compare and discuss the efficiency of the different methods, and how well they are able to reproduce the analytical result of the integral.

2 Methods

The aim of this project is to solve a six dimensional integral which represents the ground state correlation energy between two electrons in a helium atom. The one electron wavefunction is

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

where $\vec{r}_i = x_i\vec{e}_x + y_i\vec{e}_y + z_i\vec{e}_z$ and $r_i = \sqrt{x_i^2 + y_i^2 + z_i^2}$. We are considering here an helium atom which has the atomic number $Z = 2$ so in our case $\alpha = 2$. The ansatz for the two electrons wavefunction is given by the product of the two one electron functions

$$\Psi(\vec{r}_1, \vec{r}_2) = e^{-\alpha(r_1 + r_2)}. \tag{2}$$

Note that here we are neglecting the interaction between the electrons. The integral that we need to solve is

$$I = < \frac{1}{|\vec{r}_1 - \vec{r}_2|} > = \int_{-\infty}^{+\infty} dx_1 \int_{-\infty}^{\infty} dy_1 \int_{-\infty}^{+\infty} dz_1 \int_{-\infty}^{+\infty} dx_2 \int_{-\infty}^{\infty} dy_2 \int_{-\infty}^{+\infty} dz_2 \, \frac{e^{-2\alpha(r_1 + r_2)}}{|\vec{r}_1 - \vec{r}_2|}. \tag{3}$$

This integral has an analytical solution given by $5\pi^2/16^2 \approx 0.19277$ [2]. By using the Fubini's theorem [3] we can change the order of the integrals and solve them one by one. In order to solve each integral we have used the Gaussian integration method and the Monte Carlo method. This methods are described in this section. We note that from a computational point of view, some problems may occur when $|r_1 - r_2| = 0$ in the integral. For this reason we need to account this. In the Gauss method this is done by neglecting the contribution of the points for which $|r_1 - r_2| < \varepsilon$. We assume $\varepsilon = 10^{-10}$ for the Gauss-Legendre method and $\varepsilon = 10^{-9/2}$ for the Gauss-Laguerre method.

2.1 Gauss-Legendre method

The integration methods such as the Trapezoidal rule, Simpsons rule etc. all have a fixed distance between the mesh points. We have to let go of this notion in order to achieve a higher precision. This is why Gaussian quadrature is the go to algorithm for lower dimension integrals. Gaussian quadrature uses orthogonal polynomials to determine the weights ω and the mesh points. The function is approximated using N mesh points as a polynomial of degree $2N-1$ ($f(x) \approx P_{2N-1}$). We can represent $f(x)$ as a polynomial of degree $2N-1$ due to the fact that we have N mesh points and N weights. We will use Gaussian quadrature with Legendre and Laguerre polynomials. The Legendre polynomials are the solution to the important differential equation

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

When $m_i \neq 0$ we obtain the associated Legendre polynomials, while when $m_i = 0$ we obtain the Legendre polynomials as solutions. As stated, the Legendre polynomials are orthogonal, i.e. they have the following orthogonality relation [4]

$$\int_{-1}^{1} L_i(x)L_j(x)dx = \frac{2}{2i+1}\delta_{ij}. \tag{4}$$

Where the Legendre polynomials are given by the formula

$$L_k(x) = \frac{1}{2^k k!}\frac{d^k}{dx^k}(x^2-1)^k. \tag{5}$$

Given a function $f(x)$ which we want to integrate, we approximate it with a polynomial P_{2N-1} which can be represented through polynomial division by

$$P_{2N-1} = L_N(x)P_{N-1} + Q_{N-1} \tag{6}$$

where P_{N-1} and Q_{N-1} are polynomials of degree $N-1$ or less due to the polynomial division. Since $L_N(x)$ is a Legendre polynomial we have that

$$\int_{-1}^{1} f(x)dx \approx \int_{-1}^{1} P_{2N-1}dx = \int_{-1}^{1} (L_N(x)P_{N-1}(x) + Q_{N-1}(x))dx = \int_{-1}^{1} Q_{N-1}(x)dx, \tag{7}$$

due to the orthogonality relation 4. If $f(x) = P(x)$ then this becomes an exact formula. We see that for the points where $L_N(x_i) = 0$, we have

$$P_{2N-1}(x_i) = Q_{N-1}(x_i) \qquad i = 0,1,...,N-1. \tag{8}$$

We choose these points as our mesh points. In order to find the integration weights ω_i we write $Q_{N-1}(x)$ as a sum of Legendre polynomials

$$Q_{N-1}(x_k) = \sum_{i=0}^{N-1} \alpha_i L_i(x_k) = \sum_{i=0}^{N-1} \alpha_i L_{ik} \tag{9}$$

where α_i is just a prefactor. Due to orthogonality we have

$$\int_{-1}^{1} Q_{N-1}(x)dx = 2\alpha_0. \tag{10}$$

We see we can write Eq. (9) as a matrix equation

$$\hat{Q}_{N-1} = \hat{L}\hat{\alpha} \tag{11}$$

where $\hat{Q} = [Q_{N-1}(x_0), ..., Q_{N-1}(x_{N-1})]^T$, $\hat{\alpha} = [\alpha_0, ..., \alpha_{N-1}]^T$ and the matrix

$$\hat{L} = \begin{bmatrix} L_0(x_0) & L_1(x_0) & ... & L_{N-1}(x_0) \\ L_0(x_1) & L_1(x_1) & ... & L_{N-1}(x_1) \\ ... & ... & ... & ... \\ L_0(x_{N-1}) & L_1(x_{N-1}) & ... & L_{N-1}(x_{N-1}) \end{bmatrix}. \tag{12}$$

Since the Legendre polynomials are orthogonal, we must have that the columns are linearly independent, which means that the matrix $\hat{L}$ has an inverse $\hat{L}^{-1}$. Hence we find α_0 with the equation

$$\alpha_0 = \sum_{i=0}^{N-1} (\hat{L}^{-1})_{0i} Q_{N-1}(x_i) = \sum_{i=0}^{N-1} (\hat{L}^{-1})_{0i} P_{2N-1}(x_i) \tag{13}$$

where the last equality follows from the fact that x_i are the roots of the Legendre polynomials. From Eq. (10) and Eq. (13) we have a complete way of calculating the integral in Eq. (7) [4]

$$\int_{-1}^{1} f(x)dx \approx \int_{-1}^{1} Q_{N-1}(x)dx = 2\alpha_0 = \sum_{i=0}^{N-1} 2(\hat{L}^{-1})_{0i} P_{2N-1}(x_i) = \sum_{i=0}^{N-1} \omega_i P_{2N-1}(x_i) \tag{14}$$

(in practice this is implemented as $\int_{-1}^{1} f(x)dx \approx \sum_{i=0}^{N-1} \omega_i f(x_i)$) where we see that the weights are given by $\omega_i = 2(\hat{L}^{-1})_{0i}$, and the points x_i by $L_N(x_i) = 0$. We notice that if $f(x) = P(x)$ this becomes an exact formula. If we want to integrate $f(x)$ over a interval $y \in [a,b]$, we let $y = \frac{b-a}{2}x + \frac{a+b}{2}$, we then have

$$\int_a^b f(y)dy = \frac{b-a}{2} \int_{-1}^{1} f(\frac{b-a}{2}x + \frac{a+b}{2})dx. \tag{15}$$

If we want to use equation 15 in order to solve the integrals in equation 3 we shall approximate the integration interval that is infinity with a finite domain.

2.2 Gauss-Laguerre method

If we want to integrate over the interval $[0, \infty)$ and we can write our weights on the form $\omega_i = x^{\alpha} e^{-x_i}$, then there is a better set of orthogonal polynomials to use than the Legendre polynomials, these are the Laguerre polynomials. These polynomials are the solution to the differential equation

$$\left(\frac{d^2}{dx^2} - \frac{d}{dx} + \frac{\lambda}{x} - \frac{l(l+1)}{x^2} \right) \mathscr{L}(x) = 0 \tag{16}$$

where $l \geq 0$ is an integer and λ is a constant. We see this equation multiple places in physics, one of them is in the radial Schrödinger equation for the hydrogen atom. The Laguerre polynomials fulfil the orthogonality relation [4]

$$\int_0^{\infty} e^{-x} \mathscr{L}_n(x)^2 dx = 1 \tag{17}$$

and the recursion relation $(n+1)\mathscr{L}_{n+1}(x) = (2n+1-x)\mathscr{L}_(x) - n\mathscr{L}_{n-1}(x)$. To use the Gauss-Laguerre method in order to solve the integrals in equation 3 it is useful to rewrite the integrals in spherical coordinates. By using

$$d\vec{r}_1 d\vec{r}_2 = r_1^2 dr_1 r_2^2 dr_2 \sin(\theta_1) d\theta_1 \sin(\theta_2) d\theta_2 d\phi_1 d\phi_2 \tag{18}$$

we get

$$I = \int_0^\infty dr_1 \int_0^\infty dr_2 \int_0^\pi d\theta_1 \int_0^\pi d\theta_2 \int_0^{2\pi} d\phi_1 \int_0^{2\pi} d\phi_2 \, r_1^2 r_2^2 \sin(\theta_1)\sin(\theta_2) e^{-2\alpha(r_1+r_2)} \frac{1}{r_{12}} \tag{19}$$

where

$$\frac{1}{r_{12}} = \sqrt{r_1^2 + r_2^2 - 2r_1 r_2 \cos(\beta)} \tag{20}$$

where

$$\cos(\beta) = \cos(\theta_1)\cos(\theta_2) + \sin(\theta_1)\sin(\theta_2)\cos(\phi_1 - \phi_2)\,. \tag{21}$$

Equation 20 and 21 could be found by using the law of cosines [5]. Now we can do the following change of variable in the integral $\rho_i = 4r_i$ so we get

$$I = \frac{1}{4^5} \int_0^\infty d\rho_1 \int_0^\infty d\rho_2 \int_0^\pi d\theta_1 \int_0^\pi d\theta_2 \int_0^{2\pi} d\phi_1 \int_0^{2\pi} d\phi_2 \, \rho_1^2 \rho_2^2 \sin(\theta_1)\sin(\theta_2) e^{-(\rho_1+\rho_2)} \frac{1}{\rho_{12}} \tag{22}$$

At this point is convenient to apply the Gauss-Laguerre method for the radial part of the integral because by doing this the factors $\rho_1^2\rho_2^2$ and $e^{-(\rho_1+\rho_2)}$ will be absorbed in the weights ω_i. Moreover by using this method we don't need to approximate the integration interval. To solve the angular part of the integral is still convenient to use the Legendre polynomials because in this case the integration domain is not infinite.

2.3 Brute-force Monte Carlo integration

In Monte Carlo integration, the integral of a function over a given area is approximated as the mean value of the function at N randomly selected points within this area. Here N is the number of Monte Carlo samples. For the present integral, this gives (calling the integrand $f(\vec{r}_1, \vec{r}_2)$)

$$\int_{-\infty}^{\infty} d\vec{r}_1 d\vec{r}_2 f(\vec{r}_1, \vec{r}_2) \approx \langle f \rangle = \frac{1}{N} \sum_{i=1}^{N} f(\vec{r}_{1i}, \vec{r}_{2i}) \tag{23}$$

where $\vec{r}_{1i}$ and $\vec{r}_{2i}$ are randomly selected points in the integration area. The standard deviation of this integral is given by [4]

$$\sigma_N \approx \frac{\sigma_f}{\sqrt{N}} \tag{24}$$

where $\sigma_f = \sqrt{\langle f^2 \rangle - \langle f \rangle^2}$.

As a first attempt at solving this integral using Monte Carlo integration we used a brute-force approach, using cartesian coordinates x_i ($i = 1, 2, \ldots, 6$; here $\vec{r}_1 = (x_1, x_2, x_3)$ and $\vec{r}_2 = (x_4, x_5, x_6)$) and selecting the points x_i from a uniform distribution. As infinite numbers cannot be represented numerically we must set some maximum value R such that $|x_i| \leq R$. We are thus calculating the integral

$$I_R = \left(\prod_{i=1}^{6} \int_{-R}^{R} dx_i \right) \frac{\exp\left[-2\alpha \left(\sqrt{x_1^2 + x_2^2 + x_3^2} + \sqrt{x_4^2 + x_5^2 + x_6^2} \right) \right]}{\sqrt{(x_1 - x_4)^2 + (x_2 - x_5)^2 + (x_3 - x_6)^2}} \tag{25}$$

(the integrand will hereafter be referred to as $f(x_i)$. In order to solve this using Monte Carlo integration we need to pick each coordinate from a uniform distribution between $-R$ and R. However, random number

generators choose numbers from the distribution $P_u(t) = \theta(x)\theta(1-x)$ where θ is the Heaviside step function. Therefore we must perform a change of coordinates $x_i(t_i) = -R + 2Rt$, $t_i \in [0,1]$, so that the integral becomes

$$I_R = (2R)^6 \left(\prod_{i=1}^{6} \int_0^1 dt_i \right) f(x_i(t_i)) \tag{26}$$

We approximated this integral, using a uniform distribution to find points t_i, for varying N, to examine the accuracy of the method as the number of Monte Carlo samples.

2.4 Monte Carlo integration with importance sampling

The method described above is intuitive and easy to implement, but it is also not particularly refined. Since it picks the points x_i uniformly, a lot of the samples will effectively be useless, since the function f is evaluated at a point where it is close to zero due to its exponential dependence. To obtain better results, one should therefore use a different probability distribution function (PDF) to generate the points x_i where the interesting behavior of f actually is. Here we will use a PDF with a form close to that of the integrand; this is called importance sampling [4].

As discussed previously, the integral can be rewritten in spherical coordinates as

$$I = \int_0^\infty dr_1 \int_0^\infty dr_2 \int_0^\pi d\theta_1 \int_0^\pi d\theta_2 \int_0^{2\pi} d\phi_1 \int_0^{2\pi} d\phi_2 e^{-2\alpha r_1} e^{-2\alpha r_2} g(r_1, r_2, \theta_1, \theta_2, \phi_1, \phi_2) \tag{27}$$

where $g(r_1, r_2, \theta_1, \theta_2, \phi_1, \phi_2) = r_1^2 r_2^2 \sin\theta_1 \sin\theta_2 / r_{12}$. We would like to select r_1 and r_2 from exponential distributions given by

$$P(r_i) = 2\alpha e^{-2\alpha r_i} \tag{28}$$

for $r_i \in [0, \infty)$. This can be done indirectly, by first choosing numbers t_i according to the uniform PDF $P_u(t_i)$. The probability must be conserved when transforming from one distribution to the other, meaning that (for $0 \leq t_i \leq 1$)

$$P_u(t_i)dt_i = dt_i = P(r_i)dr_i \tag{29}$$

Integrating, this gives a relation between t_i and r_i:

$$t_i(r_i) = \int_0^{r_i} P(r_i')dr_i' = 1 - e^{-2\alpha r_i} \tag{30}$$

Solving this for r_i gives

$$r_i(t_i) = -\frac{\ln(1-t_i)}{2\alpha} \tag{31}$$

The values for r_i are then generated by choosing t_i from the uniform distribution, and finding r_i from this expression. Note that there is no restriction on r_i here, so we avoid the extra source of error from approximating infinity.

For the integrals over r_i this gives

$$\int_0^\infty dr_i = \int_0^1 \left(\frac{dt_i}{dr_i} \right)^{-1} dt_i = \int_0^1 \frac{1}{P(r_i(t_i))} dt_i \tag{32}$$

For the angles we use uniform distributions as before; we select numbers t_i, $i = 3,4,5,6$, according to $P_u(t_i)$, and express the angles in terms of these as

$$\theta_i(t_{i+2}) = \pi t_{i+2} \tag{33}$$
$$\phi_i(t_{i+4}) = 2\pi t_{i+4} \tag{34}$$

With all the spherical coordinates expressed in terms of $t_i \in [0,1]$, the integral I can be rewritten as

$$I = \frac{4\pi^4}{(2\alpha)^2}\left(\prod_{i=1}^{6}\int_0^1 dt_i\right) g(r_1(t_1), r_2(t_2), \theta_1(t_3), \theta_2(t_4), \phi_1(t_5), \phi_2(t_6)) \tag{35}$$

The prefactor comes from the transformations of variables; there is a factor of π from each transformation $\theta \to t$, a factor 2π from each transformation $\phi \to t$, and a factor $1/2\alpha$ from each $r \to t$ since $P(r_i)$ contains an extra factor of 2α compared to $\exp(-2\alpha r_i)$. As in the brute-force case, we approximated this integral using a uniform distribution to pick points t_i, for different values of N.

3 Implementation

This sections explains how we implemented the algorithms and what kind of choices we have done.

3.1 General algorithms

We implemented the algorithms in the c++ programs called ”Gauss.cpp”, which contains the Gauss-Legendre and Gauss-Laguerre algorithms, and ”montecarlo.cpp” which contain the brute-force and improved Monte Carlo algorithms. We calculated the weights for the Gauss-Legendre and the Gauss-Laguerre algorithms using the algorithms outlined in the book ”Numerical Recipes 3rd Edition: The Art of Scientific Computing”[6]. When running the program ”Gauss.cpp” you will have to choose if you want to evaluate the one electron function, if you want to solve the integral with the Gauss-Legendre algorithm or with the Improved Gauss-Quadrature, i.e. the Gauss-Laguerre algorithm. This is done in the console after the program in executed. If you choose to evaluate the one electron function you have to choose the number of mesh points and the functions limit, it will then create a file in the ”Gauss_results” folder called ”One_electron_function” which contain as columns the points and the function value. By running the Gauss-Legendre algorithm you have to choose the number of integration points and an approximation of infinity for our function. By running the Gauss-Laguerre algorithm you have to choose the number of integration points and the filename of the output file. The program used to plot the results from these algorithms is a python program called ”Gauss_plot.py”. It plots of the wave functions, the errors for the Gauss algorithms and the logarithm of the CPU time for running the Gauss-Laguerre quadrature, vs. the logarithm of the number of integration points N.

As stated, the Monte Carlo algorithms are found in the ”montecarlo.cpp” file, in order to run the algorithms we have three programs: ”mc_brute_main.cpp”, which runs the brute force Monte Carlo algorithm. Here you have to choose the number of Monte Carlo samples (the number of Monte Carlo samples is given by 10ˆ(input)) and approximate infinity for the wave function, this is done in the console after the program is executed. The file ”mc_improved_main.cpp” runs the improved Monte Carlo algorithm and ”mc_improved_parallized.cpp” also

runs the improved Monte Carlo algorithm, but this program is parallized. When running the "mc_improved_main.cpp" you only have to choose the number of Monte Carlo samples. We parallized the Improved Monte Carlo algorithm using MPI, this mean you have to compile the file using the compiler "mpicxx", and execute the file using "mpiexec -np #" where # is the number of parallel jobs. Here you have to choose the number of Monte Carlo samples as an argument when running the function.

We also made three programs which runs the Monte Carlo algorithms a given number of times and writes the data to a file in the folder "mc-results".

We have used the random number generator Mersenne Twister (*mt19937*) in our Monte Carlo programs, as this is the preferred random number generator since it has a period of 2^{19937}. We set the seed using the system clock such that we get different results when running the algorithms. In the parallized algorithm we have to set the seed as the system clock plus the rank of the parallized job in order to get different results.

3.2 Unit tests

To ensure that the algorithms described above work as intended, and have been implemented correctly, we made several unit tests that the programs had to pass. First, we tested the Gauss-Laguerre and improved Monte Carlo methods (since these are expected to be the most accurate) on a simpler integral, namely

$$\int_{-\infty}^{\infty} d\vec{r}_1 d\vec{r}_2 \exp\left[-4(r_1+r_2)\right] \tag{36}$$

(here we have explicitly set $\alpha = 2$), which has an analytical solution given by $\pi^2/2^6$. This integral can be solved exactly as described above (the factor 4^{-5} in the Gauss-Laguerre method is changed to 4^{-6} as there are no r's in the denominator here). We ensured that the analytical result was reproduced with $N = 20$ points for the Gauss-Laguerre method, and $N = 10^8$ Monte Carlo samples in the Monte Carlo method. Since the Monte Carlo method depends on random numbers there will be statistical variations in the results, and it can thus not be expected to reproduce the analytical result every time. Instead, we required it to be within $5\sigma_N$ of the analytical solution.

We also checked that the function which gave the weights and points generated for the Gauss-Legendre method were correct. For $N = 2$ and integration domain $[-1,1]$ these should be $\omega_0 = \omega_1 = 1$ and $x_1 = -x_0 = 1/\sqrt{3}$ [4]. We therefore ensured that the function produced these values.

4 Results and discussion

The obtained results are discussed in this section. First of all we will focus on Gauss integration with both Legende and Laguerre polynomial. Then we will discuss the outcomes for the Monte Carlo integration, with and without importance sampling.

4.1 Gauss-Legendre method

The first step to solve the integral in this case is to find the more reliable approximation for the integral domain. In order to do this is useful to plot the one electron wavefunction. This is shown in Figure 1.

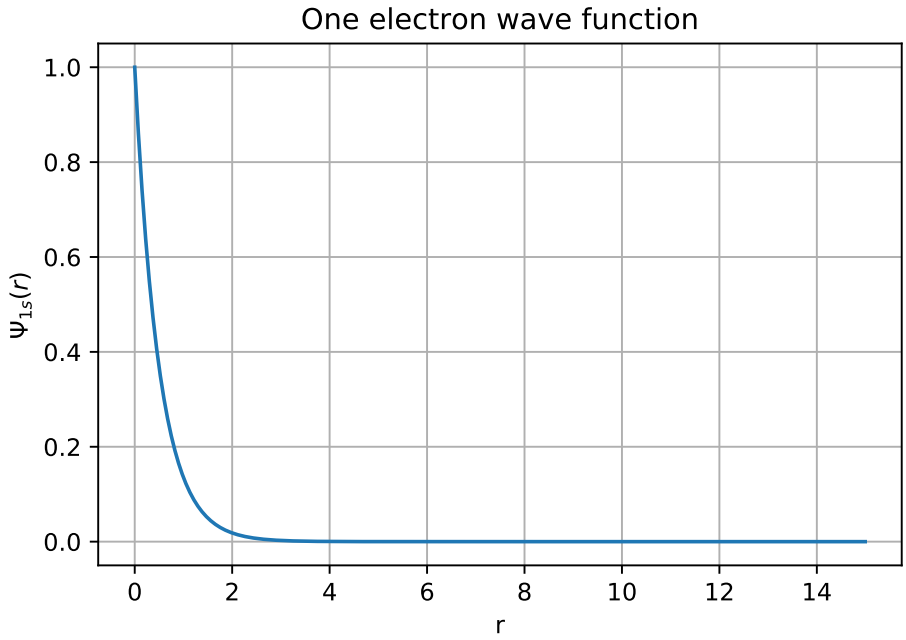

Fig. 1. Plot of the wave functions for one electron in a helium atom.

If $r_i \lesssim 4.6$ we have that $\psi_{1s}(r) < 10^{-4}$ so we assume that $\psi_{1s}(r) = 0$. We can do this because we are interested only in the first three leading digits of the integral as shown in Table 1. Moreover we are integrating the product of two single electron function and this fact justify even more our approximation. This because the one electron wavefunction never exceed 1.

Solving the integral with $N = 45$ mesh points in the domain $[-4.8, 4.8]$ we get that the computed results converges at the level of the third leading digit as shown in Table 1.

	Computed result	**Rounded computed value**	**Expected value**	**Error**
Gauss-Legendre	0.192652	0.193	$5\pi^2/16^2 \approx 0.193$	0.00011
Gauss-Laguerre	0.192596	0.193	$5\pi^2/16^2 \approx 0.193$	0.00017

Table 1. In this table is shown the comparison between the computed solution with The Legendre and Laguerre polynomials with $N = 45$ mesh points. The Gauss-Legendre algorithm was computed in the domain $[-4.8, 4.8]$. It is possible to see that both solution give the correct result up to three leading digits. However in this case the Gauss-Legendre integration gives a more precise result.

However this method is not completely satisfying because the result depends from the integration interval we choose as is shown in figure 2.

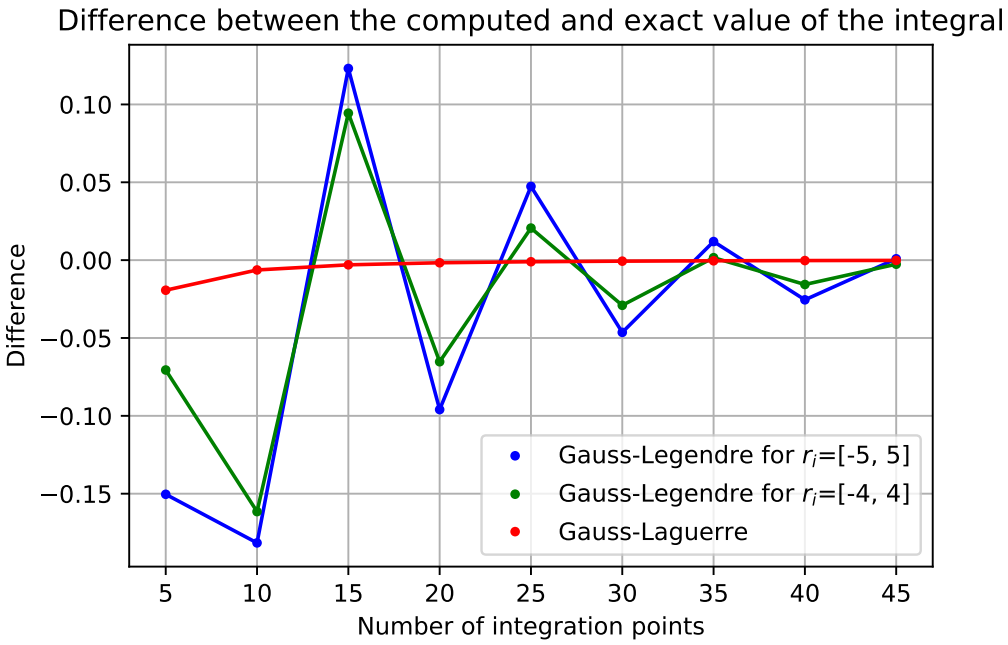

Fig. 2. Plot of the errors for the Gauss algorithms. When we use the Legendre polynomials the result depends from the integration interval we choose. For a given integration interval we can find a number of mesh point in order to obtain a smaller error with this method than with the Laguerre polynomials. Anyway the method with Laguerre polynomials does not require to approximate the integration interval and is on average far more precise.

4.2 Gauss-Laguerre method

With this method we don't need to approximate the integration domain. In this way the result of the integral does not depends from our approximation. Moreover, In this case the sum on the polynomials represents a better approximation of the function. This is because our function and the Laguerre polynomials have the same exponential part. For those reasons the Gauss-Laguerre method gives a more precise result than the Gauss-Legendre algorithm as is shown in figure 2. It is possible to get a more precise result with the Legendre polynomials than with the Laguerre-polynomials with the same number of mesh points, as shown in table 1.

We also examined the time used by the programs. There is no big difference between the computational time of the two Gauss algorithms and the trend is the same for both of them. The logarithm of the CPU time is plotted against the logarithm of the number of integration points N in figure 3, for various values between $N = 10$ and $N = 45$ (for fewer points the algorithms runs very quickly, to the point where the generating of integration weights and points dominates; thus the behavior of the time as a function of N is different here). From the slope of the data one can see that the time is approximately proportional to N^6; for $N = 45$ the Gauss-Laguerre algorithm took about 2 minutes and 23 seconds to run.

This illustrates one of the main disadvantages of Gaussian quadrature for high-dimensional integrals; when solving an n-coordinate integral with N integration points, each coordinate is evaluated at N points, giving roughly N^n operations in total (this explains the N^6 behavior noted above). Thus, for a high-dimensional integral, if high precision is desired, Gaussian quadrature quickly becomes rather inefficient.

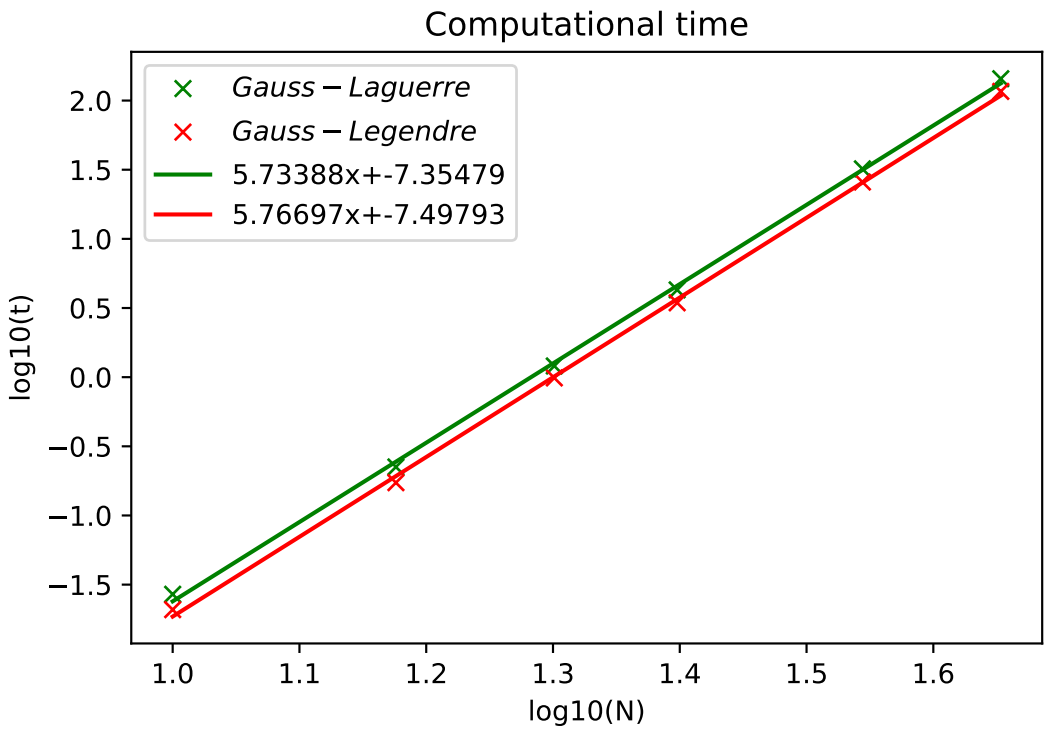

Fig. 3. Logarithm of the CPU time for running the Gauss-Laguerre and Gauss-Legendre quadrature, vs. the logarithm of the number of integration points N. A straight line is fitted to the data. The Gauss-Legendre algorithm was computed in the interval $[-5,5]$.

4.3 Brute force Monte Carlo integration

The brute force Monte Carlo algorithm was run with 10^6 integration points to 10^9 integration points, and the results is shown in 2. To get a more accurate value for the CPU time, we ran the algorithm 10 times (for each number of integration points) and calculated the averages (for the standard deviation, we calculated instead the RMS value, i.e. the square root of the mean of the variance σ_N^2). The results are shown in table 2. We

Table 2. Integral values, standard deviations and the times it takes to run the brute force Monte Carlo algorithm for a given integration points, where the max radius is set to 3. The standard deviations and times are averaged from 10 runs with specified number of integration points; the integral value is chosen from one arbitrarily chosen run. As a reminder, the analytical result of the integral is $5\pi^2/16^2 \approx 0.19277$.

Integration points	Integral	σ_N	Time (s)
1E6	0.20	0.07	0.10 ± 0.01
1E7	0.185	0.012	1.02 ± 0.06
1E8	0.197	0.004	9.9 ± 0.2
1E9	0.191	0.001	99.0 ± 1.7

found that the brute force algorithm does not work very well, we chose max r (r_{max}) to be 3 when running the algorithm, this was done because if r_{max} was smaller we would miss too much of the area, while if we chose r_{max} to be bigger we would have too many points from the Monte Carlo algorithm which would contribute nothing, see figure 1. We see that in order to get somewhat close to the correct result, and get the standard deviation down to 10^{-3}, we have to use more than 10^9 samples, which means that the program takes a long

time to run.

The time used is roughly proportional to the number of integration points. This is as expected; as there are no loops in the algorithm apart from the main one over the samples (for each sample, we simply pick random points and evaluate the function at these points, and then add this value to the integral), the number of floating point operations is proportional to N.

This shows why, when the dimensions of integrals grow large, Monte Carlo methods are better suited than Gaussian quadrature; for an n-dimensional integral with N samples, the Monte Carlo method simply evaluates the function at N randomly selected points in the n-dimensional space one is integrating over. In other words, the CPU time is roughly independent of the dimension of the integral; even for very high-dimensional integrals, the CPU time will be roughly proportional to N. Thus it is much less expensive, in terms of CPU time, to increase the precision with Monte Carlo integration than Gaussian quadrature for high-dimensional integrals.

4.4 Improved Monte Carlo integration

The results for the Monte Carlo method with importance sampling are shown, for 10^4 to 10^9 samples, in table 3. As anticipated from the arguments in the previous section, the results here are far better than those obtained from the brute-force approach; by comparing tables 2 and 3 one can see that the results (the integral and standard deviation) for N samples with the improved method roughly correspond to those for $10^3 N$ with the brute-force method, and for the highest values of N the analytical result is reproduced with good precision. Thus the improved version is much more effective; if one only needs a particular precision, this can be achieved with a much lower number of Monte Carlo samples when using importance sampling, which again means that the program is quicker. Alternatively, if one can only use some maximum number of samples (for example due to time constraints), the improved version can give a much better precision than the brute force one in the same amount of time.

The logarithm of the standard deviation σ_N is plotted against the logarithm of the number of Monte Carlo samples in figure 4. The slope is almost exactly -0.5, which means that the standard deviation is proportional to $1/\sqrt{N}$, in agreement with (24). This indicates that σ_f, the standard deviation of $f\left(\vec{r}_i, \vec{r}_j\right)$ (with $\vec{r}_i$ and $\vec{r}_j$ selected randomly as described in section 2.4) is approximately constant. This is not surprising, as the standard deviation of a function is only determined by the PDF; theoretically, in one dimension, it is given by [4]

$$\sigma_f^2 \equiv \int_{-\infty}^{\infty} (f(x) - \langle f(x) \rangle)^2 P(x) dx \tag{37}$$

where

$$\langle f(x) \rangle = \int_{-\infty}^{\infty} f(x) P(x) dx \tag{38}$$

and $P(x)$ is the PDF in question.

4.5 Parallelization and vectorization

Finally, we tested the effect of parallelization and vectorization on the CPU time; specifically, we compared the CPU times for the ”improved” Monte Carlo method (i.e. with importance sampling) when running two

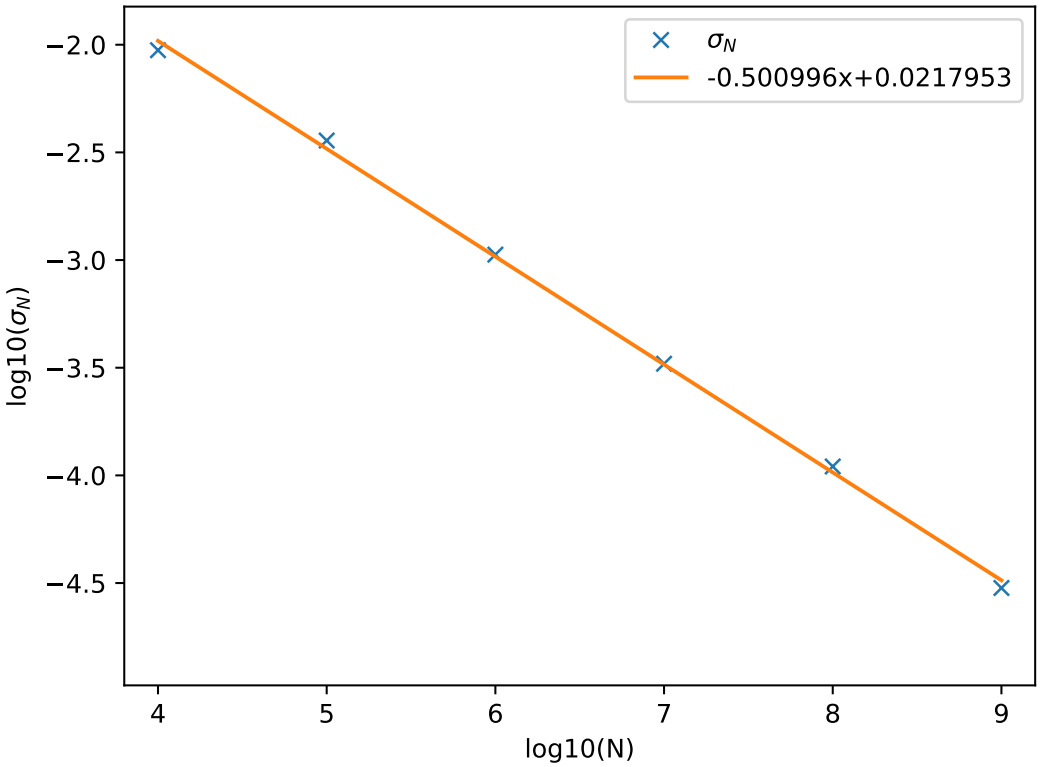

Fig. 4. Logarithm of the standard deviation of the integral from Monte Carlo integration with importance sampling, versus the logarithm of the number of samples. A straight line is fitted to the set.

processes in parallel (using MPI) to the times presented in table 3. We also examined how the time changed when the ”-O3” compiler flag, for vectorization (which was used for all the previous results) was no longer used. These results are shown in table 4 (the times are averaged over 10 runs, as before).

One can see that running the program with two parallel processes decreases the CPU time by about a factor of 2, as one would expect. It does not quite halve the CPU time, however; this might be due to the extra time required to call the additional functions necessary for parallelization (e.g. adding the contributions from each process). If that is the case, one might expect the CPU time to approximately become inversely proportional to the number of processes when this number gets larger. The effect of vectorization on the CPU time is also apparent; there is a speed-up of a factor of almost 4. This is due to the form of the Monte Carlo algorithm; it contains a loop over the iterations/samples, and as each iteration is independent of the previous one, it is possible to run several of these simultaneously.

Table 3. Integral values, standard deviations and the times it takes to run the improved Monte Carlo algorithm (with importance sampling) for a given number of integration points. The standard deviations and times are averaged from 10 runs with specified number of integration points; the integral value is chosen from one arbitrarily chosen run. As a reminder, the analytical result of the integral is $5\pi^2/16^2 \approx 0.19277$.

Integration points	Integral	σ_N	Time (s)
1E4	0.189	0.009	0.0018 ± 0.0005
1E5	0.188	0.004	0.018 ± 0.002
1E6	0.1934	0.0011	0.14 ± 0.02
1E7	0.1932	0.0003	1.13 ± 0.05
1E8	0.1928	0.0001	11.1 ± 0.4
1E9	0.19278	0.00003	110 ± 1

N	t [s] (2 processes, vectorized)	t [s] (1 process, vectorized)	t [s] (2 processes, not vectorized)	t [s] (1 process, not vectorized)
1E6	0.08 ± 0.01	0.14 ± 0.02	0.24 ± 0.04	0.44 ± 0.04
1E7	0.64 ± 0.09	1.13 ± 0.05	2.5 ± 0.9	4.27 ± 0.07
1E8	6.1 ± 0.3	11.1 ± 0.4	22.4 ± 0.5	42.4 ± 0.6
1E9	58 ± 1	110 ± 1	-	-

Table 4. CPU times (averaged over 10 runs) for varying N, with and without using parallelization and/or vectorization (the times with vectorization, without parallelization are simply copied from table 3). The results for the value of the integral and standard deviation are not shown, as they were essentially identical to those shown in table 3.

5 Conclusion

From the results for the Gaussian quadrature algorithms, we see that Gauss-Legendre and Gauss-Laguerre methods are able to find the expected integral results with three leading digits, when $N = 45$. The computational time (or CPU time) is approximately the same for both algorithms. For both methods, the number of floating points operation, as well as the computational time are proportional to N^n, where n in the dimension of the integral (in this case equal to six). Thus both the Gaussian quadrature methods are not useful for high dimensional integrals. The most relevant difference between this algorithms is that in order to get the desired result with the Gauss-Legendre integration we approximated the integration interval with a particular interval. Gauss-Laguerre integration does not require an approximation on the integration interval. Moreover in this case is more effective because Laguerre polynomials represents a better approximation of the function we want to integrate. Thus this method turns out to be way more precise for almost each value of N. Note, however, that the Gauss-Legendre method may still be very useful for other problems; for low-dimensional integrals with finite integration intervals it can give good accuracy (in fact, when we switched to the Gauss-Laguerre method for the radial integrals, we still used the Gauss-Legendre method for the angular integrals with good results), but with integration boundaries like those in this problem, more specialized methods are better suited.

The number of floating points operations, as well as the computational time, for the Monte Carlo integration are proportional to N. However, this method bases its functioning on the law of large numbers [1]. For this reason we need an huge number of point to get the desire result. For the brute force method we need more than

10^9 points. This means that this algorithm is slower than the Gaussian algorithms. However in this case the computational time is roughly independent of the dimension of the integral. Thus this method is suitable for high dimensional integrals (even more so for higher-dimensional integrals than here; for, e.g., a 20-dimensional integral, Gaussian quadrature will essentially be useless due to the time it takes to run the algorithm, so Monte Carlo methods must be used). The improved Monte Carlo method is far more effective because the PDF is more similar to the function. Thus we need less points for the same precision. This means that the program is quicker with a fixed maximum error, or more precise with a fixed number of mesh points. Vectorization and parallelization both decreases the computational time. In the end the faster Monte Carlo method turns out to be about a thousand time faster than the Gaussian quadrature methods.

References

[1] Richard Durrett. *Probability: theory and examples*. Thomson Brooks/Cole, 2007.

[2] *Assignment text for project 3 in FYS3150/FYS4150*. Department of Physics, University of Oslo, Norway.

[3] Walter Rudin. *Real and complex analysis*. McGraw Hill Education India, 2015.

[4] Morten Hjorth-Jensen. *Computational Physics Lecture Notes Fall 2015*.

[5] Clifford A. Pickover. *The Math book: from Pythagoras to the 57th dimension, 250 milestones in the history of mathematics*. Sterling, 2009.

[6] William H. Press et al. *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. 3rd ed. New York, NY, USA: Cambridge University Press, 2007.