Numerical integration of ground state correlation energy in a two-electron atom

Project 3, FYS4150
Autumn 2019

Jowita Borowska

ABSTRACT

In the following project, the six-dimensional integral, representing the ground state correlation energy between two electrons in a helium atom, is being solved numerically. Knowing the closed-form solution to the integral, $I = 5\pi^2/16^2 \approx 0.19277$, we can assess the accuracy of the obtained results. First, we estimate the integral in Cartesian coordinate system, where each of the six variables $(x_1, y_1, z_1, x_2, y_2, z_2)$ lies in the interval $[-3, 3]$, as the single-particle wave function, ψ_{1s}, is approximately zero outside this domain. We apply Gaussian Quadrature (GQ) method with Legendre polynomials in order to compute this integral, as well as the brute force Monte Carlo (MC) method (with each variable drawn randomly from the uniform distribution). Best result of the Gaussian-Legendre Quadrature is $I = 0.195817$ (for $N = 25$ integration points) and best estimate of the integral from the brute force MC is $I = 0.192754$ (for $N = 10^8$ samples). Thereafter, we transform the integral into the spherical coordinate system, which increases the overall accuracy of both methods. The GQ method is improved by employing Laguerre polynomials for the radial part (r_1, r_2) and Legendre polynomials for the angles $(\theta_1, \theta_2, \phi_1, \phi_2)$, and results in $I = 0.193285$ (for $N = 15$). We introduce also the MC with importance sampling, by using the exponential PDF for the radial variables, which is better suited for the given integrand, and gives precise results already for smaller number of samples, N. Finally, this part of the code is parallelized (executed with 5 threads), using OpenMP, which speeds-up the performance significantly (from 27.5 s to 12.6 s).

1 Introduction

Schroedinger equation for helium atom has no closed-form or analytical solution. However, some approximations can be made in order to estimate the wavefunction of two interacting electrons in the helium atom. We will assume that each electron can be modelled by a single-particle hydrogenic wave function, which for an electron i in state 1s reads

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

where $\mathbf{r}_i = x_i\mathbf{e}_x + y_i\mathbf{e}_y + z_i\mathbf{e}_z$ and $r_i = \sqrt{x_i^2 + y_i^2 + z_i^2}$. Parameter α will hereafter be fixed to $\alpha = 2$, corresponding to the charge of helium atom.
The wave function for two electrons is then approximated by a product of two 1s wave functions,

$$\psi(\mathbf{r_1}, \mathbf{r_2}) = \psi_{1s}(\mathbf{r}_1)\psi_{1s}(\mathbf{r}_2) = e^{-\alpha(r_1+r_2)}$$

(note that it is not properly normalized).
Then, the quantum mechanical expectation value of the correlation energy between these two electrons, interacting via a classical repulsive Coulomb potential, is given by

$$\left\langle \frac{1}{|\mathbf{r_1} - \mathbf{r_2}|} \right\rangle = \int_{-\infty}^{\infty} e^{-2\alpha(r_1+r_2)} \frac{1}{|\mathbf{r_1} - \mathbf{r_2}|} d\mathbf{r_1} d\mathbf{r_2}. \tag{2}$$

In the following project, the integral above will be solved numerically, employing Gauss Quadrature and Monte Carlo integration methods. The comparison with the closed-form solution, $5\pi^2/16^2$, will allow us to test the accuracy of computations. Each approach will be successively developed, in order to achieve higher precision and better computational efficiency. This includes a change of coordinate system, use of better suited polynomials for Gauss Quadrature method or importance sampling in case of the Monte Carlo integration. Finally, the code will be parallelized using Open Multi-Processing (OpenMP). Mathematical motivation, as well as numerical implementation details are thoroughly explained in the Methods section. The results produced by the program are gathered and discussed more closely in the subsequent section (Results and discussion).

The main program (project3_main.cpp) is written in C++ and together with the small Python code (project3_plot.py), used to plot, available in the following Github repository: github.com/jowborowska/CompPhysics

2 Methods

The mathematical description of integration methods applied in this project follows closely that of Hjorth-Jensen (2015), with some additional aspects added based on Press, Teukolsy, Vetterling and Flannery (2007).

2.1 Correlation energy integral in different coordinate systems

Cartesian coordinates

Expectation value of the correlation energy between two electrons is a six-dimensional integral. In Cartesian coordinate system we have then a set of three variables for each electron, x_1, y_1, z_1 and x_2, y_2, z_2, so that Eq. (2) can be rewritten as

$$I = \int_{-\infty}^{\infty} dx_1 \int_{-\infty}^{\infty} dy_1 \int_{-\infty}^{\infty} dz_1 \int_{-\infty}^{\infty} dx_2 \int_{-\infty}^{\infty} dy_2 \int_{-\infty}^{\infty} dz_2 \frac{e^{-2\alpha\left(\sqrt{x_1^2+y_1^2+z_1^2}+\sqrt{x_2^2+y_2^2+z_2^2}\right)}}{\sqrt{(x_1-x_2)^2+(y_1-y_2)^2+(z_1-z_2)^2}}.$$

However, setting integration limits from $-\infty$ to ∞ is not only impossible for our numerical approach, but also unnecessary. Analyzing a single-particle wave function (Eq. 1), we can find a limit for which its value drops to approximately zero,

$$\psi_{1s}(\lambda) = e^{-\alpha\lambda} \approx 0 \Rightarrow \ln(0.001) \approx -\alpha\lambda \;\Rightarrow \lambda \approx \frac{-6.9}{-2} = 3.45,$$

where we approximated zero with a small number, 0.001, to estimate logarithm, and set $\alpha = 2$. We make a plot of $\psi_{1s}(r_i)$ for the region containing λ in order to see if our approximation is accurate. As shown on Figure 1., the function drops almost entirely to zero between $r_i = 2$ and $r_i = 3$, so that $r_i > 3$ can be excluded from the domain. We will then settle on $\lambda = 3$, replacing integration limits $-\infty$ and ∞ with $-\lambda$ and λ, respectively.

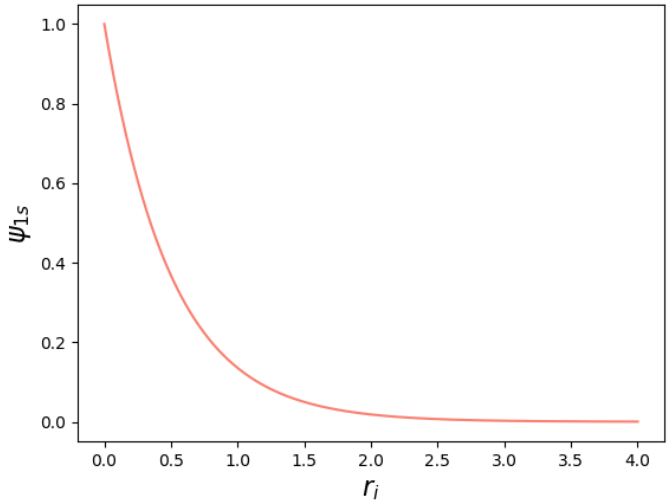

Figure 1: Plot of single-particle wave function, $\psi_{1s}(r_i) = e^{\alpha r_i}$, where $\alpha = 2$, supporting the method. The behaviour of function is meant to motivate the change of integration limits. We can clearly see that for $\sqrt{x_i^2 + y_i^2 + z_i^2} = r_i > 3$, the function is approximately zero, so that this region can be excluded from the integration domain.

Spherical coordinates

We can also change the integral to spherical coordinates. Then, $d\mathbf{r}_1 d\mathbf{r}_2$ from Eq. (2) becomes

$$d\mathbf{r}_1 d\mathbf{r}_2 = r1^2 dr_1 r_2^2 dr_2 dcos(\theta_1) dcos(\theta_2) d\phi_1 d\phi_2 = r1^2 dr_1 r_2^2 dr_2 sin(\theta_1) d\theta_1 sin(\theta_2) d\theta_2 d\phi_1 d\phi_2,$$

where in the last equaility we used $dcos(\theta_i) = sin(\theta_i) d\theta_i$. The integral to solve in this coordinate system is

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

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

2.2 Gaussian Quadrature

The main idea behind some numerical integration methods is to approximate an integral with a sum,

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

where ω_i are weights and x_i are mesh points x_i. Gaussian Quadrature gives a freedom of choosing not only the weighting coefficients, but also the location of mesh points at which the function is evaluated (so that the points are not equally spaced). We have then 2N degrees of freedom and the integrand can be approximated by an orthogonal polynomial of degree 2N-1, where N is a number of mesh points. Such a high order translates to the high accuracy of the method only when the function to integrate is smooth, i.e. well approximated by a polynomial. However, even if the integrand is not smooth, we can still extract from it the weight function, $W(x)$, in order to remove any integrable singularities from the integral (Press et al., 2007). This approach is reflected by the Gaussian quadrature formula,

$$\int_a^b f(x)dx = \int_a^b W(x)p(x)dx \approx \int_a^b P_{2N-1}(x)dx = \sum_{i=0}^{N-1} \omega_i P_{2N-1}(x_i). \tag{3}$$

The mesh points, x_i, are zeros of the chosen orthogonal polynomial (of order N) and the weights can be determined by solving a set of linear equations. Moreover, it is worth noticing that for a given weight function, $W(x)$, we end up evaluating the integrand for the smooth function $p(x)$.

There are several important polynomials, which arise from solving differential equations, and are frequently used in scientific applications. In this project we will employ Legendre and Laguerre polynomials in the Gaussian Quadrature method. First, we will evaluate the integral in Cartesian coordinates, using only Legendre polynomials. Thereafter, we will improve our Gaussian Quadrature method by evaluating integral in the spherical coordinate system, using Laguerre polynomials for the radial part (r_1 and r_2) and Legendre polynomials for the angular part (θ_1, θ_2, ϕ_1, ϕ_2).

Legendre polynomials

Legendre polynomials are defined in the interval $x \in [-1, 1]$ and characterized by the weight function $W(x) = 1$. They fulfill both the orthogonality relation,

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

(where L_i(x) is the Legendre polynomial of order i and δ_{ij} denotes the Kronecker delta), as well as the recursion relation,

$$(j+1)L_{j+1}(x) + jL_{j-1}(x) - (2j+1)xL_j(x) = 0.$$

As we have previously discussed, an integrand is approximated by a polynomial of order $2\mathrm{N} - 1$, $f(x) \approx P_{2N-1}(x)$. In order to explain how weights and mesh points are generated, we perform the following decomposition

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

where $P_{N-1}(x)$ and $Q_{N-1}(x)$ are some polynomials of order $\mathrm{N} - 1$ or less, and $L_N(x)$ is a Legendre polynomial of order N. We could represent P_{N-1} by the Legendre polynomials through

$$P_{N-1}(x) = \sum_{k=0}^{N-1} a_k L_k(x),$$

which implemented into the integrated former equation gives

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

$$= \sum_{k=0}^{N-1} \int_{-1}^{1} L_N(x)a_k L_k(x)dx + \int_{-1}^{1} Q_{N-1}(x)dx = 0 + \int_{-1}^{1} Q_{N-1}(x)dx,$$

where the last equality follows from the orthogonality relation, so that

$$\Rightarrow \int_{-1}^{1} P_{2N-1}(x)dx = \int_{-1}^{1} Q_{N-1}(x)dx. \tag{4}$$

We see, that it is sufficient to evaluate the integral over $Q_{N-1}(x)$. Moreover, at the mesh points x_k, where L_N is zero, we have

$$P_{2N-1}(x_k) = Q_{N-1}(x_k) \quad k = 0, 1, ..., N-1.$$

Representing Q_{N-1} in terms of Legendre polynomials (as $P_{N-1}(x)$ before),

$$Q_{N-1}(x) = \sum_{i=0}^{N-1} a_i L_i(x),$$

and employing the orthogonality relation, yields

$$\int_{-1}^{1} Q_{N-1}(x)dx = \sum_{i=0}^{N-1} a_i \int_{-1}^{1} L_0(x)L_i(x)dx = 2a_0, \tag{5}$$

where we set $L_0(x) = 1$! We know the values of Q_{N-1} at the zeros of L_N, so that

$$Q_{N-1}(x_k) = \sum_{i=0}^{N-1} a_i L_i(x_k) = \sum_{i=0}^{N-1} a_i L_{ik} \quad k = 0, 1, ..., N-1. \tag{6}$$

Since the Legendre polynomials are linearly independent of each other, none of the columns in matrix $\mathbf{L}$, containing L_{ik}, are linear combinations of the other columns, so that $\mathbf{L}^{-1}\mathbf{L} = \mathbf{I}$. Therefore, muliplying both sides of Eq. (5) with $\sum_{j=0}^{N-1} L_{ji}^{-1}$ results in

$$\sum_{i=0}^{N-1} \left(L^{-1}\right)_{ki} Q_{N-1}(x_i) = a_k. \tag{7}$$

Combining equations (3), (4), (5) and (7) gives

$$\sum_{i=0}^{N-1} \omega_i P_{2N-1}(x_i) = \int_{-1}^{1} P_{2N-1}(x)dx = \int_{-1}^{1} Q_{N-1}(x)dx = 2a_0 = 2\sum_{i=0}^{N-1} \left(L^{-1}\right)_{0i} P_{2N-1}(x_i),$$

so that we can clearly identify the weights, ω_i, with $2\left(L^{-1}\right)_{0i}$. The general formula for weights, derived by applying recursion relation, reads (Weisstein, 2019)

$$\omega_i = \frac{2(1-x_i^2)}{(N+1)^2[L_{N+1}(x_i)]^2},$$

where the mesh points, x_i, are zeros of Legendre polynomial of degree N, L_N.

Despite the fact that Legendre polynomials are defined in the interval $[-1, 1]$, their use is not limited to this region. We can always transform to the general interval, $[a, b]$, through a change of variable from t to x, where

$$t = \frac{b-a}{2}x + \frac{b+a}{2} \Rightarrow \int_a^b f(t)dt = \frac{b-a}{2}\int_{-1}^{1} f\left(\frac{b-a}{2}x + \frac{b+a}{2}\right)dx.$$

This approach will be followed in order to use Legendre polynomials to compute integral in Cartesian coordinates, where $x_1, y_1, z_1, x_2, y_2, z_2 \in [-\lambda, \lambda] = [-3, 3]$, as well as for the angular part of the spherical integral, in the improved Gaussian Quadrature, where $\theta_1, \theta_2 \in [0, \pi]$ and $\phi_1, \phi_2 \in [0, 2\pi]$.

Laguerre polynomials

Laguerre polynomials are defined in the interval $x \in [0, \infty)$ and characterized by the weight function $W(x) = x^\alpha e^{-x}$. Similar to the Legendre polynomials, they fulfill the orthogonality relation,

$$\int_0^\infty x^\alpha e^{-x}\mathcal{L}_n^\alpha(x)\mathcal{L}_m^\alpha(x)dx = \frac{(n+\alpha)!}{n!}\delta_{nm},$$

and recursion relation,

$$(n+1)\mathcal{L}_{n+1}^\alpha(x) = (2n+1-x)\mathcal{L}_n^\alpha(x) - n\mathcal{L}_{n-1}^\alpha(x).$$

In order to find associated weights and mesh points, we can take the exact same approach as in the case of Legendre polynomials, applying the appropriate orthogonality relation, as defined above. Then, we find that the mesh points, x_i, are zeros of Laguerre polynomial of degree N, $\mathcal{L}_N^\alpha$, and the weights follow the relation (Weisstein, 2019)

$$\omega_i = \frac{\Gamma(n+\alpha+1)x_i}{n!(n+1)^2\left[\mathcal{L}_{n+1}^\alpha(x_i)\right]^2}.$$

As it already has been mentioned, we will employ Laguerre polynomials in the improved Gaussian Quadrature method for the radial part (r_1, r_2) of the integral in spherical coordinates. We will set the weight function to be $W(r_i) = r_i^\alpha e^{-r_i} = e^{-r_i}$ (not to confuse $\alpha = 0$ here with previously defined $\alpha = 2$, arising from the charge of helium atom). Then, the integrand has to be reduced by one exponential factor, so that we use $\exp(-(2\alpha-1)(r_1+r_2)) = \exp(-3(r_1+r_2))$ in the function smoothed by the weight function.

2.3 Monte Carlo Integration

Brute force Monte Carlo Integration, uniform PDF

The simplest approach to the Monte Carlo integration method is based on approximating the integral with the average of the integrand function, f, for uniform probability density function, $p(x)$ (so that $p(x) = 1$ for $x \in [0, 1]$ and $p(x) = 0$ otherwise),

$$I = \int_0^1 f(x)dx \approx \langle f \rangle = \frac{1}{N}\sum_{i=1}^{N} f(x_i)p(x_i),$$

where N is the number of Monte Carlo samples. The random x_i variables, where the function, f, is evaluated, are generated from the interval $[0,1]$ (as $p(x_i)$ for x_i values outside this interval would be zero, erasing the contribution to the sum anyway). We can also define the variance of such integral, being a measure of the extent to which f deviates from its average over the region of integration,

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

The variance of the series of measurments is $\sigma_N^2 \approx \sigma_f^2/N$ and the standard deviation is defined as the square root of the variance. The goal of Monte Carlo calculations is to have σ_N as small as possible after N samples.

Naturally, this approach may also be used for a general interval, $[a,b]$, through a change of variable

$$x_j = a + (b-a)x_i,$$

where $x_i \in [0,1]$ is randomly drawn from the uniform distribution, and x_j is the variable that we actually use to evaluate the function contributing to the sum. This technique will be applied while computing the integral in Cartesian coordinates, where all of the variables lie in the interval $x_1, y_1, z_1, x_2, y_2, z_2 \in [-\lambda, \lambda] = [-3, 3]$, as well as for the angular part of the spherical integral, in the improved Monte Carlo integration method, where $\theta_1, \theta_2 \in [0, \pi]$ and $\phi_1, \phi_2 \in [0, 2\pi]$.

Improved Monte Carlo Integration, with importance sampling

Importance sampling - choosing an appropriate distribution from which we draw random variables, brings an improvement to the brute force Monte Carlo method. Using the probability density function (PDF), that resembles the behaviour of the integrand, lets us focus on the contributions to the sum giving a better Monte Carlo estimate. We want to change a variable $x \in [0,1]$, generated from the uniform distribution, $p(x)$, to a new variable, y, drawn from a different distribution. This transformation has to preserve the probability,

$$p(y)dy = p(x)dx \Rightarrow p(y)dy = dx \Rightarrow x(y) = \int_0^y p(y')dy'.$$

Computing the integral in spherical coordinates, we will still apply uniform distribution for the angles. However, since the radial variables of the integrand appear in the exponential, $e^{-4(r_1+r_2)}$, it is a good choice to use the exponential distribution intead, $p(y) = 4e^{-4y}$, so that

$$p(y)dy = 4e^{-4y}dy = dx \Rightarrow x(y) = \int_0^y 4e^{-4y'}dy' = 1 - e^{-4y}$$

$$\Rightarrow y(x) = -\frac{1}{4}\ln(1-x),$$

where x is again drawn randomly from the interval $[0,1]$, and y is a new random variable in the domain $y \in [0,\infty)$, in our case r_1 or r_2. Mark that using Monte Carlo method improved in such way, we need to exclude the exponential factor from the integrand and divide it by 4, since the integral becomes

$$I = \int_0^\infty F(y)dy = \int_0^\infty p(y)\frac{F(y)}{p(y)}dy = \int_0^1 \frac{F(y(x))}{p(y(x))}dy = \int_0^1 \frac{F(y(x))}{4e^{-4y(x)}}dy.$$

For our multidimensional integral we do that twice (for r_1 and r_2), so that we have to divide the integrand by $4 \cdot 4 = 16$, as well as extract the entire exponential factor, $e^{-4(r_1+r_2)}$.

Throughout this subsection, we have used mathematical formulation for one-dimensional integrals. The implementation of Monte Carlo method to compute our six-dimensional integral is pretty straightforward. The only adjustment, we have to apply, is multiplying the Monte Carlo estimate and the standard deviation by the factor representing volume of the integral. In the case of integral in Cartesian coordinates this factor is

$$\prod_{i=1}^{6}(b_i - a_i) = \prod_{i=1}^{6}(\lambda - (-\lambda)) = \prod_{i=1}^{6}(3 - (-3)) = 6^6,$$

since all the variables lie in the interval $[-3, 3]$. For the integral in spherical coordinate system we have

$$\prod_{i=1}^{6}(b_i - a_i) = (1-0)\cdot(1-0)\cdot(2\pi-0)\cdot(2\pi-0)\cdot(\pi-0)\cdot(\pi-0) = 4\pi^4,$$

where radial variables have been transformed, so that the corresponding interval is $[0, 1]$, and angles $\phi_1, \phi_2 \in [0, 2\pi]$, $\theta_1, \theta_2 \in [0, \pi]$.

2.4 Numerical implementation

The main program, written in C++, follows closely the approach described in this section. We have two functions returning integrands - in Cartesian coordinate system and in spherical coordinates, properly reduced dependent on the method. Here, a simple if-statement accounts for the possible singularities that could lead to the numerical problems (denominator of the integrand, $|\mathbf{r}_1 - \mathbf{r}_2|$, being close to zero). Gaussian Quadrature methods (one using Legendre polynomials and the improved one, applying both Legendre and Laguerre polynomials) make use of already written functions - gauss_laguerre() together with gammln(), available in the Code Examples for the project, as well as gauleg(), from the library lib.cpp. These functions compute and return the abcissas and weights of respectively Gauss-Laguerre and Gauss-Legendre formulae. Moreover, we have two functions applying the Monte Carlo integration method, where the random variables are generated from the uniform distribution, using ran0 function, again from lib.cpp library. All these ready functions have been fully implemented into the program, so that no additional external libraries are needed in order to successfuly compile and run the code.

We measure the CPU time needed to perform computations by each algorithm. In order to acheieve better efficiency and an optimal speed-up, we parallelize the improved Monte Carlo part of the code, using OpenMP. We set the number of threads to execute the parallel region (for loop) to 5. It is important to point out that the clock() function, used until now, measures the CPU time used by the process and would return the cumulative CPU time of all the threads involved. Therefore, we implement a real-clock timer called omp_get_wtime() for this parallelized part instead.

3 Results and discussion

3.1 Gaussian Quadrature

The results of integral evaluation by the Gaussian Quadrature method for various numbers of integration points, N, are shown in Table 1. It is clear that the values produced by the algorithm computing integral in Cartesian coordinates, only with Legendre polynomials, converge much slower towards the exact analytical solution, $5\pi^2/16^2 \approx 0.192766$. We get a sensible estimate, with relative error of order $\approx 1.6\%$ for $N = 25$ mesh points. However, the improved Gaussian Quadrature method (that solves the integral in spherical coordinates, using Legendre polynomials for angular part and Laguerre polynomials for radial part) always produces a more accurate result for a given N. It gives the estimate closest to the exact solution for $N = 15$ integration points, with corresponding relative error $\approx 0.27\%$.
It is important to point out, that results of both methods diverge from the analytical solution if we continue to increase the number of integration points, N, after reaching the estimate with maximal presicion and smallest error (i.e. $N = 25$, $I = 0.195817$ for Gaussian Legendre and $N = 15$, $I = 0.193285$ for Gaussian Legendre & Laguerre).

Analyzing the CPU time needed in order to perform computations, we can see an apparent tendency - using the improved algorithm takes more time for a given N. The benefit of improving the method is then a significant gain of accuracy (for a given number of integration points), at the cost of computational efficiency.

	Gaussian Legendre			Gaussian Legendre & Laguerre		
N	Integral	Relative error	CPU time (s)	Integral	Relative error	CPU time (s)
10	0.0719797	0.626595	0.050964	0.177081	0.0813679	0.137603
15	0.239088	0.240305	0.565272	0.193285	0.00269565	1.49674
20	0.156139	0.190004	3.17237	0.194786	0.0104784	8.55551
25	0.195817	0.0158267	12.0868	0.194804	0.0105751	33.0558
30	0.177283	0.080319	35.6826	0.194779	0.010443	98.5995

Table 1: The results of Gaussian Quadrature integration method produced by the program for different number of integration points, N. Table gathers computed values of integral, relative error and CPU time needed to perform the algorithm, for GQ with Legendre polynomials (and integrand in Cartesian coordinates) and improved GQ using both Legendre and Laguerre polynomials (integrand in spherical coordinates). The exact analytical solution of the integral is $5\pi^2/16^2 \approx 0.192766$.

3.2 Monte Carlo Integration

The results of integral evaluation by the Monte Carlo method for various numbers of samples, N, are shown in Table 2. The table gathers also the values of standard deviation, σ_N, associated with the computations. As we have already mentioned in the Methods section (2.3), σ_N is a measure of the extent to which the function deviates from its average (Monte Carlo estimate) over the range of integration. The goal is to have σ_N as small as possible after N samples. For both, brute force MC and MC with importance sampling, the value of standrad deviation decreases for larger N. This is a quite obvious result - the statistical approach gains precision for a larger set of samples. In general, the improved Monte Carlo method (with exponential distribution for the radial part of the integral in spherical coordinates) produces

much more accurate results throughout the entire range of analyzed N. This method needs less samples to estimate integral accurately, since the variables are drawn from the distribution that resembles the original integrand better than the uniform distribution. We need to use approximately $N = 10^7$ samples in the brute force Monte Carlo algorithm in order to produce a result which accuracy is comparable to the MC with importance sampling already with $N = 10^4$ samples.

Again, the CPU time elapsed using the improved method has a tendency to be larger - it takes around twice as long to perform the Monte Carlo computations with importance sampling, comparing to the brute force Monte Carlo. The improved Monte Carlo part of the code has been parallelized, as described in the Methods section (2.4). The time corresponding to $N = 10^8$ is then 12.6076 s, which is a significant speed-up.

	Brute force Monte Carlo			Monte Carlo with importance sampling		
N	Integral	σ_N	CPU time (s)	Integral	σ_N	CPU time (s)
10^4	0.150396	0.0896616	0.001105	0.189704	0.00851123	0.00274108
10^5	0.222673	0.0855526	0.011082	0.192263	0.00316787	0.0274117
10^6	0.190191	0.0176823	0.110341	0.192688	0.00102265	0.281262
10^7	0.179076	0.00787534	1.1345	0.192783	0.000326801	2.82842
10^8	0.192754	0.00363913	11.3998	0.192744	0.000104162	27.4771

Table 2: The results of Monte Carlo integration method produced by the program for different numbers of Monte Carlo samples, N. Table gathers computed values of integral, standard deviation, σ_N, and CPU time needed to perform the algorithm, for brute force MC (with unifrom PDF for all variables in Cartezian coordinates) and for MC with importance sampling (exponential PDF for radial part, uniform PDF for angular part of the integrand in spherical coordinates). The exact analytical solution of the integral is $5\pi^2/16^2 \approx 0.192766$.

3.3 Gaussian Quadrature vs Monte Carlo integration

The most apparent difference is the number of integration points used in Gaussian Quadrature method, compared to the number of Monte Carlo samples. It is quite obvious that such a statistical approach as Monte Carlo integration requires a much larger N than the Gaussian Quadrature method, where mesh points at which function is evaluated are not random.

In case of both methods, computing integrals in spherical coordinates takes longer CPU time than performing crude algorithms, which use Cartesian coordinates. This is also reasonable - integrand in spherical coordinates is more complex, as it contains sin() and cos() functions to calculate.

The precision of result that we can achieve with Monte Carlo method is much higher than the precision of Gaussian Quadrature. The best Gaussian Legendre-Laguerre estimate of the integral gives ≈ 0.19329, whereas for Monte Carlo method it is ≈ 0.19275, which for the analytical solution ≈ 0.19277 gives a solution exact to the fourth decimal digit.

Monte Carlo method is also much more stable than Gaussian Quadrature. The standard deviation keeps decreasing with larger N, while in case of Gaussian Quadrature we can easily miss the N-value giving the highest accuracy and make the error larger in the process of increasing the number of integration points.

4 Conclusions

There are several numerical integration methods that can be employed in order to compute a multidimensional integral. In this project, we have shown two - Gaussian Quadrature and Monte Carlo integration methods. The integral we have considered (correlation energy between two electrons in a helium atom) has a closed-form solution, making it possible to assess the accuracy of a given integration method. In not only this aspect, but also in the aspect of stability (dependence on the number of integration points), the Monte Carlo method turned out to be much more suitable for computing such a multidimensional integral.

Both Gaussian Quadrature and Monte Carlo methods could be improved by changing the coordinate system and taking a mathematical approach that was better for the integrand under consideration. This meant using Laguerre polynomials for radial part in GQ method and exponential PDF in MC with importance sampling. It is then a good practice to use some brain power in order to implement a problem in an optimal mathematical form, upgrading the quality of numerical results.

Nevertheless, making the integrand more complicated mathematically (here by introducing trigonometric functions) leads to longer CPU time needed in order to perform calculations. The efficiency of a program can be significanty improved by employing parallel computing, as it has been shown.

BIBLIOGRAPHY

Hjorth-Jensen, M. (2015) *Computational Physics - Lecture Notes.* Retrieved from:
www.github.com/CompPhysics/ComputationalPhysics/blob/master/doc/Lectures/lectures2015.pdf

Press, W. H., Teukolsy, S. A., Vetterling, W. T., Flannery, B. P. (2007) *Numerical Recipes. The art of scientific computing.* New York, NY: Cambridge University Press

Weisstein, E. W. (last updated 2019) *Laguerre-Gauss Quadrature.* and *Legendre-Gauss Quadrature.* From MathWorld–A Wolfram Web Resource.
Retrieved from: http://mathworld.wolfram.com/Laguerre-GaussQuadrature.html
and http://mathworld.wolfram.com/Legendre-GaussQuadrature.html