Project 3, FYS3150

Markus Borud Pettersen

September 2019

Abstract

1 Introduction

2 Theory & Algorithms

Since integration can be formulated as simply determining the area under some arbitrary curve, it can also be numerically approximated as

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

where $f(x)$ is some function, and x_i are discrete integration points, while w_i are the integration weights. In fact, (1) tells us that an integral can be approximated by summing smaller area segments of width w_i, and heigth $f(x_i)$, and in the limit $N \to \infty$, we see that these segments perfectly approximate any well-behaved function we can think of, and the integral becomes exact.

In real-world applications, however, we will never be able to use an infinite number of integration points, N, at least not numerically. Instead, we can use some clever tricks which in some cases allows us exact results, for even just a few integration points. To see how this is possible, we can first think of a simple example where we evaluate the function $f(x)$ at evenly spaced points. If $f(x)$ is a step function, i.e. takes on some constant value c in the interval $[a, b]$, and is zero otherwise, (1) becomes *exactly*

$$I = \int_a^b f(x)dx = \int_a^b c\ dx = c \cdot (b - a),$$

with only two integration steps. Observe that the integration weights from (1) are simply $-a$ and b.

While this is a somewhat dull example, it does tell us that by choosing an appropriate approximation to the function $f(x)$, properly spaced coordinates x_i and weights w_i, we can speed up many integration tasks. One common way of achieving these benefits, is by using so-called Gaussian quadrature methods.

2.1 Gaussian Quadrature

If we forego the requirement that our integration, or mesh points, must be evenly spaced, we can approximate an integral by selecting both an approximation to the function we are integrating, *and* the integration weights. In other words, we can approximate the integral as

$$\int_a^b f(x)dx = \int_a^b W(x)g(x)\ dx \approx \sum_{i=1}^{N} w_i g(x_i). \tag{2}$$

Here, $g(x)$ is a smooth function of our choosing, and $W(x)$ a so-called weight function. However, for Gaussian quadrature methods, $g(x)$ is taken to be an orthogonal polynomial, as they can be used to approximate any function, and, as we will see, the integration weights w_i and mesh points x_i can be determined quite easily. We should point out that by allowing ourselves to select both weights and mesh points, we actually have $2N$ parameters to be determined, N for the weights and N for the values of the mesh points, which means that we can achieve better approximations to a given function, still using only N mesh points during integration [1]. Motivated by this, we will approximate functions investigated in this text, using polynomials of degree $2N-1$.

2.2 Legendre & Laguerre Polynomials

To investigate how we can find integration weights and mesh points, we will consider a special class of orthogonal polynomials, namely the Legendre polynomials. If we denote a legendre polynomial of degree n as L_n, then it can be shown that they fulfill the orthogonality condition

$$\int_{-1}^{1} L_n(x)L_n'(x)dx = \frac{2}{2n}\delta_{nn'}, \tag{3}$$

where $\delta_{nn'}$ is the Kronecker delta function [1]. In addition, the Legendre polynomials are complete, in the sense that any function can be written in terms of linear combinations of them, provided we use infinitely many of them. Returning to our function f(x), we know that we want to approximate it using a polynomial of degree $2N-1$, which in turn can be written in terms of Legendre polynomials as

$$P_{2N-1}(x) = L_N(x)P_{N-1}(x) + R_{N-1}(x), \tag{4}$$

which we arrived at through simple polynomial division, and R_{N-1} is simply a remainder of the division. As we already noted, we can write P_{N-1} in terms of Legendre polynomials, due to completeness, but this means that

$$\int_{-1}^{1} L_N(x)P_{N-1}(x)\ dx = 0,$$

which means that

$$\int_{-1}^{1} f(x)\ dx \approx \int_{-1}^{1} P_{2N-1}\ dx = \int_{-1}^{1} R_{N-1}\ dx, \tag{5}$$

and that if x_0 is a root of L_N, then

$$P_{2N-1}(x_0) = R_{N-1}(x_0)$$

which might not look impressive, but it can be shown [1] that by rewriting the remainder in terms of Legendre polynomials, we can determine R_{N-1} exactly, which, by virtue of (5), tells us that we can integrate P_{2N-1} exactly, as we wished.

To finish off, we will also list the properties of another class of orthogonal polynomials, the Laguerre polynomials. These carry the weight function $W(x) = x^{\alpha}e^{-x}$, and fulfill the orthogonality relation, up to a factor,

$$\int_{0}^{\infty} P_N P_M\ dx = \delta_{MN},$$

where P_N denotes a Laguerre polynomial of degree N, P_M one of degree M, and δ_{MN} is the Kronecker-delta function. Importantly for us, this means that we have an integral which can be factorized into the weight function of the Laguerre quadrature, and some smooth function, $g(x)$, the integral simply becomes

$$\int_{0}^{\infty} x^{\alpha}e^{-x}g(x)\ dx \approx \sum_{i=1}^{N} w_i g(x_i),$$

as we can absorb the exponential terms according to (2).

2.3 Monte Carlo Integration

As we have already seen, multi-dimensional integral can be dealt with numerically, but require careful handling, and can be quite resource-intensive to compute. One way of dealing with this, is by applying so-called Monte Carlo integration methods. To recap, we already stated that we can approximate an integral numerically, as

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

given an appropriate choice of weights and mesh points. However, in the case of a d-dimensional integral, we require d summations, which means that the the number of computations increases exponentially with d. One rather satisfying way of circumventing this *curse of dimensionality*, is to imagine that x is actually some random variable, that we draw from a probability distribution function (PDF). We will denote the probability of drawing x from the PDF as $p(x)$, and by rewriting the integral, we then get

$$\int_a^b f(x)\ dx = \int_a^b p(x)\frac{f(x)}{p(x)}\ dx \approx \sum_{i=1}^N p(x_i)\frac{f(x_i)}{p(x_i)}. \tag{6}$$

While (6) may look somewhat unimpressive, one should note that the rightmost expression is nothing but the expectation value of $f(x)/p(x)$. In other words

$$\int_a^b f(x)\ dx \approx \left\langle \frac{f(x)}{p(x)} \right\rangle$$

, but in the limit, when the number of integration points tends toward infinity, this expression becomes exact, and if we express the expectation value using the mean, instead, we obtain the even simpler relation, that LAW OF LARGE NUMBERS

$$\int_a^b f(x)\ dx = \lim_{N\to\infty} \frac{1}{N}\sum_{i=1}^N \frac{f(x_i)}{p(x_i)}. \tag{7}$$

While (7) surely is a clever trick, we also see an obvious drawback, namely that it requires a large number of integration points in order to be valid, in most cases. But, the real beauty of (7) lies in the fact that we can perform the *exact* same calculation for a multidimensional integral as well. In other words, if the function f is a function of several independent variables, for example x, y, and z, then the integral can be approximated as

$$\int f(x,y,z)\ dxdydz \approx \frac{1}{N}\sum_{i=1}^N \frac{f(x,y,z)}{p(x,y,z)},$$

and if we treat x, y and z as random, independent numbers, the probability simply becomes $p(x,y,z) = p_x(x)p_y(y)p_z(z)$. Here, we have denoted the different probability functions according to their respective variable, to signify that we do not have to restrain ourselves to the case where x, y and z all follow the same PDF. In fact, as the limits of the integral tend to be different for different variables, we indeed have to draw them from different probability distributions, with different domains.

The great takeaway from (7), is that we can avoid the exponential growth in computation time that usually accompanies multidimensional integrals. To see how much of a speedup we might achieve, we can look at the error we might expect in a Monte Carlo calculation. Since we are treating the numerical integration as an expectation value, it is almost natural to consider the standard deviation of said calculation, which for uncorrelated variables can be written as

$$\sigma_N = \frac{1}{\sqrt{N}} \cdot \sqrt{\left\langle \left(\frac{f(x)}{p(x)} \right)^2 \right\rangle - \left\langle \frac{f(x)}{p(x)} \right\rangle^2}, \tag{8}$$

[1] with N being the number of integration points. Note that we have only stated the one-dimensional case above, but the same relation holds, also for multidimensional expressions. Importantly, (8) therefore tells us that the error is proportional to $1/\sqrt{N}$, regardless of the dimensionality of the integral.

One final point that should be adressed, is how, and why we select the random variables. As mentioned, we require that these variables need to follow some probability distribution, and that the domain of this distribution must match the interval of integration. As such, the simplest example we can come up with, might just be the uniform distribution, with probability density function

$$p(x) = \begin{cases} \frac{1}{b-a} & x \in [a,b], \\ 0 & \text{else}. \end{cases} \tag{9}$$

As we can see for the uniform distribution, the integral of a function can be approximated simply by finding the average of the function over the integration interval, and multiplying the result with a scaling factor. $(b-a)$, as the probability density is constant. If the integral is taken over the interval $[0,1]$, the integral simply becomes the mean function value over that interval, as the scaling factor is simply 1.

There is, however, no good reason to limit ourselves to the uniform distribution, as different PDFs might actually yield improved estimates of the integral. The reason for this is quite simple; what we are actually doing when approximating the integral using Monte Carlo methods, is actually approximating the area under the curve, using a rectangle of height $f(x)$, and width equal to the integration interval. As we add up these smaller areas, some will undershoot, and some will overshoot the actual area under the curve, and these errors will cancel out, giving us a decent approximation of the actual area. However, if we use the uniform distribution, we will be equally likely to sample any region of the curve in question, which is useful for slowly changing, and, ideally uniform functions. If our function is spiked around a certain value, on the other hand, it might be beneficial to sample this region more often than others, in order to obtain a better estimate of the area under this region.

In other words, it might be useful to use a probability distribution which *matches* the behaviour of the function we are integrating. This technique is called importance sampling [2], as we tend to sample the function more at its important regions, so to speak. As we will explore in the next section, an example where importance sampling is useful, is for exponential functions, where we simply draw random numbers from the exponential distribution.

2.4 Application: Expected Particle Separation in Helium Atom

As an example, we will investigate the expected value of the separation between two electrons in a helium atom, given by

$$\left\langle \frac{1}{|\mathbf{r}_1 - \mathbf{r}_2|} \right\rangle = \int_{-\infty}^{\infty} \frac{e^{-2\alpha(r_1+r_2)}}{|\mathbf{r}_1 - \mathbf{r}_2|} \, d\mathbf{r}_1 \, d\mathbf{r}_2, \tag{10}$$

[3] where $\mathbf{r}_1$ and $\mathbf{r}_2$ denote the position vectors of the two particles, while $r_1 = |\mathbf{r}_1| = \sqrt{x_1^2 + y_1^2 + z_1^2}$, and $r_2 = |\mathbf{r}_2|$, while α corresponds to the number of electrons, i.e. two in the case of the helium atom. $d\mathbf{r_1}$ and $d\mathbf{r_2}$ are the volume elements, given in cartesian coordinates as $dx_1dy_1dz_1$ and $dx_2dy_2dz_2$, respectively. As such, all limits necessarily go from $-\infty$ to ∞, which is less than ideal in a numerical sense, as we have to approximate infinity with some finite number. We can polish this result somewhat, by noting that

$$r_{12} = |\mathbf{r}_1 - \mathbf{r}_2| = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2} = \sqrt{r_1^2 + r_2^2 - 2\mathbf{r}_1 \cdot \mathbf{r}_2},$$

but this does little to alleviate the problem of infinite limits. However, as we briefly touched upon with the Laguerre polynomials, they are defined in the range $x \in [0, \infty)$, which also happens to be the range of the lengths of the position vectors, r_1 and r_2. Motivated by this, we can transform our integral by introducing spherical coordinates, and instead obtain

$$\int_0^\infty \int_0^{2\pi} \int_0^\pi \int_0^\infty \int_0^{2\pi} \int_0^\pi \frac{e^{-2\alpha(r_1+r_2)}}{r_{12}} r_1^2 r_2^2 \sin\theta_1 \sin\theta_2 d\theta_1 d\phi_1 dr_1 d\theta_2 d\phi_2 dr_2, \tag{11}$$

where r_1 and r_2 are the lengths of the position vectors, as before, θ_1 and θ_2 are the polar angles defined on the interval $[0, \pi]$, while ϕ_1 and ϕ_2 are the azimuthal angles defined on $[0, 2\pi]$. As the keen reader may have noticed, these limits are all compatible with the ranges of the Laguerre and Legendre polynomials. In fact, we can rewrite the radial part of the integral as

$$\int_0^\infty \int_0^\infty r_1^2 e^{-r_1} \cdot r_2^2 e^{-r_2} \cdot e^{-3(r_1+r_2)}/r_{12} dr_1 dr_2, \tag{12}$$

where we have inserted that $\alpha = 2$. In particular, we can see that the first terms correspond exactly to the weight function of the Laguerrre polynomials, and that the limits are also appropriate. While the limits for the angular parts of (11) are not in the range $[-1, 1]$, we can easily obtain such a range by a simple transformation of variables. With the transformed version of (11), we can therefore find a more reasonable estimate of the integral, by applying both Gauss-Legendre, and Gauss-Laguerre quadratures to the same calculation. In addition, we can use the fact that the weight function of the Gauss-Laguerre quadrature is of the form $x^\alpha e^{-x}$ to rewrite (11) by introducing $u_1 = r_1/4$, and $u_2 = r_2/4$. This allows us to absorb the exponential terms in the integrand into the Gauss-Laguerre weights. In other words, according to (2), the radial part of (11) simply becomes

$$\frac{1}{4^5} \int_0^\infty \int_0^\infty u_1^2 u_2^2 e^{-(u_1+u_2)}/r_{12}\ dr_1 dr_2 \approx \frac{1}{4^5} \sum_{i=1}^N \sum_{j=1}^N w_i w_j \tag{13}$$

where w denotes a Gauss-Laguerre quadrature weight.

We can also solve (11) using Monte Carlo integration. The only differerence from (7), is that we now concern ourselves with a six-dimensional integral, but if we denote the integral I, we can now approximate it using a single summation, i.e.

$$I \approx \frac{1}{N} \sum_{i=1}^N \frac{1}{p(r_{1,i})p(r_{2,i})p(\theta_{1,i})p(\theta_{2,i})p(\phi_{1,i})p(\phi_{2,i})} \cdot \frac{e^{-4(r_{1,i}+r_{2,i})}}{r_{12}} r_{1,i}^2 r_{2,i}^2 \sin(\theta_{1,i}) \sin(\theta_{2,i}),$$

where we have added additional subscripts to each variable, to signify that they are random numbers drawn from some probabilty distribution. If we use the uniform distribution, we see that the scaling factors from (9) become simply $1/(2\pi)$ and $1/\pi$ for the ϕ's and θ's, respectively. We are now presented with a challenge as far as the radial probabilities are concerned. If we set the upper bound of the uniform distribution to ∞, as the limits dictate, our sum automatically diverges. Once again, we must approximate infinity with some finite number, λ, and the radial scaling term is simply $1/\lambda$. Since both particles carry the same integration limits, we find that for the uniform distribution, we can approximate the integral as

$$I \approx \lambda^2 \cdot 4\pi^4 \frac{1}{N} \sum_{i=1}^N \frac{e^{-4(r_{1,i}+r_{2,i})}}{r_{12}} r_{1,i}^2 r_{2,i}^2 \sin(\theta_{1,i}) \sin(\theta_{2,i}). \tag{14}$$

To avoid having to approximate infinity, we can use importance sampling, and sample the radial components from the exponential distribution, e^{-x}. Doing so for both r_1 and r_2, we see that the probability is no longer constant, and that the scaling factor from (7), $p(r_1) = e^{-r_1}$, and $p(r_2) = e^{-r_2}$ has to be included in each step. However, this merely causes the cancellation of one exponential factor. If we keep sampling the other quantities from the uniform distribution, we finally arrive at

$$I \approx 4\pi^4 \frac{1}{N} \sum_{i=1}^{N} \frac{e^{-3(r_{1,i}+r_{2,i})}}{r_{12}} r_{1,i}^2 r_{2,i}^2 \sin(\theta_{1,i}) \sin(\theta_{2,i}).$$

3 Methods

3.1 Gaussian Quadrature

The various integration techniques discussed in the previous sections, were applied to determining the expected value of particle separation in a Helium atom, i.e. computing the integral in (10). All calculations were implemented using c++.

First, the integral was computed using cartesian coordinates, as in (10), using a six-fold summation, by approximating each coordinate dependency using (2). The weight function was taken to be one, in accordance with the Gauss-Legendre quadrature approach. In order to approximate infinity in each coordinate direction, the single particle wave-function was plotted, and inspected visually, in order to determine an adequate point at which the wavefunction tended towards zero. Based on this result, which is shown in Fig. 1, infinity was approximated as $\pm\infty \approx \pm\lambda = \pm 3$.

The integration weights and mesh points were then calculated using an external library function, available at https://github.com/CompPhysics/ComputationalPhysics/tree/master/doc/. Note that this library function also accomodated the integration limits from $-\lambda$ to λ, by performing a variable shift to the usual Gauss-Legendre integration limits from -1 to 1. Using the calculated weights and mesh points, the integral was then estimated according to (2).

To improve upon this calculation, a mix of Gauss-Legendre and Gauss-Laguerre quadratures were applied to the spherical-coordinate version of the expected value, i.e. (11). Using Gauss-Legendre quadratures for the angular coordinates θ and ϕ, and Gauss-Laguerre quadratures for the radial part of the equation, the integral was once again calculated in accordance with (2).

Note that the Gauss-Laguerre mesh points and integration weights were determined using the same external library, but the original implementation is due to [4]. Also, for both the cartesian, and spherical coordinate integral calculations, the contribution to the integral was ignored if the denominator was below a threshold value of 10^{-12}, in order to keep the sum from diverging. For more information on the actual code, SEE APPENDIX.

To determine the accuracy of the Gauss-Legendre and Gauss-Laguerre methods, both programs were executed and timed for various number of integration points, from $n = 2$ to 30. The results of the integration were then compared with the analytical expected value of the particle separation in the helium atom, $5\pi^2/16^2$ [3]. The error of the two methods were also compared directly, to investigate if the Gauss-Laguerre quadrature method outperformed the Gauss-Legendre quadrature.

3.2 Monte Carlo Integration

To investigate the possible benefits of Monte Carlo integration, a simple, brute-force algorithm was first developed, in which the spherical equation in (11) was computed, using a uniform distribution for each variable. The integration itself was implemented as detailed in the *Theory & Algorithms* section, by computing the average function value for a given number of integration points n, and sampling the integration variables randomly. It should be noted that these are of course only pseudorandom numbers, obtained using the c++ standard library *rand* function. In order for the random numbers to match the integration interval, a suitable shift of variables was also performed for each variable. To avoid a diverging sum, the infinite limit of the radial dependency was approximated by only sampling random values of r in the range $[0, 3]$, i.e. approximating $\infty \approx \lambda = 3$, as before.

In addition, Monte Carlo integration of the integral in (11), using importance sampling was also implemented. While the basic integration strategy was similar to that of the brute force method, the radial variables were now drawn from an exponential distribution. To do so, random uniform variables were drawn using the c++ standard library function *rand*, but were subsequently transformed according to $r_i = -\ln(1 - x_i)$, where x_i is a random, uniform number. All other variables were drawn in the same way as for the brute-force method. Note that the radial part of the integrand was rewritten to accompany importance sampling, as outlined in the *Theory & Algorithms* section.

Finally, a parallelized version of the importance sampling Monte Carlo integration program was developed, using the Message Passing Interface (MPI) library. This was done to see if increases in speed could be achieved.

Each Monte Carlo integration algorithm was then applied to determining the expected particle separation in the Helium atom. In the process, the execution time as well as absolute error, was logged for each method. In order to compare the brute force and importance sampling methods, the standard deviation in (8) was computed for each algorithm. For each algorithm, the g++ compiler was used, but for the parallelized program, the -O2 vectorization flag was also used, and a total of 8 parallel processes were attempted employed.

4 Results

4.1 Gaussian Quadrature

Fig. 1 shows the single-particle wavefunction $\psi(\mathbf{r}) = e^{-2|\mathbf{r}|}$, as a function of position. We can see that the wavefunction decays exponentially, and becomes close to zero beyond $|\mathbf{r}| = 3$.

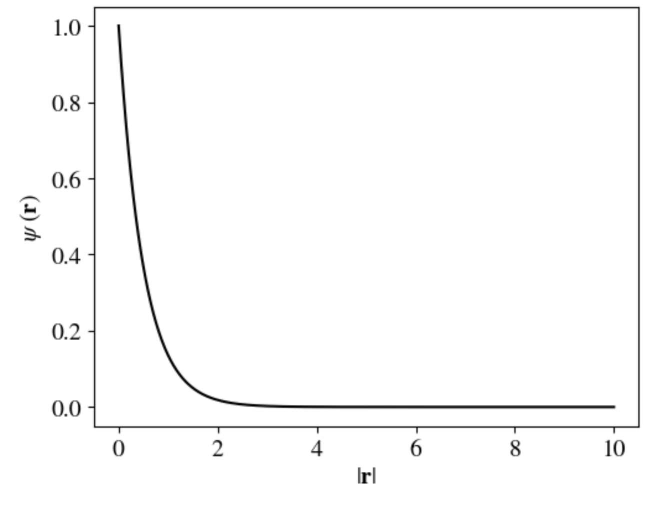

Figure 1: Plot of the single-particle wavefunction $\psi(\mathbf{r})$, given by $e^{-2|\mathbf{r}|}$, where $\mathbf{r}$ is the radial position vector of the given particle, here taken to be dimensionless.

Fig. 2 shows the results of the numerical integration of (10) and (11), using both Gauss-Legendre, and mixed Gauss-Laguerre/Gauss-Legendre quadrature methods. For the pure Gauss-Legendre quadrature, inifinity was approximated as $\lambda = 3$. The leftmost figure shows the value of the computed integral for both methods, alongside the exact value, $5\pi^2/16^2$, while the rightmost figure shows the corresponding error (log 10) of the same calculation. We can see that the mixed Gauss-Laguerre/Gauss-Legendre quadrature method converges quickly towards the exact value, and that the error decreases smoothly as a function of the number of integration points, n. For the Legendre quadrature, on the other hand, convergence is somewhat more slow, and it also appears to oscillate about the exact value, alternating for odd and even values of n. Note that no such effect can be seen for the mixed approach. Furthermore, we can see that the solution achieves approximately three significant digits after $n \approx 26$. For the pure Gauss-Legendre case, however, only odd values and $n \geq 27$ achieve this level of precision.

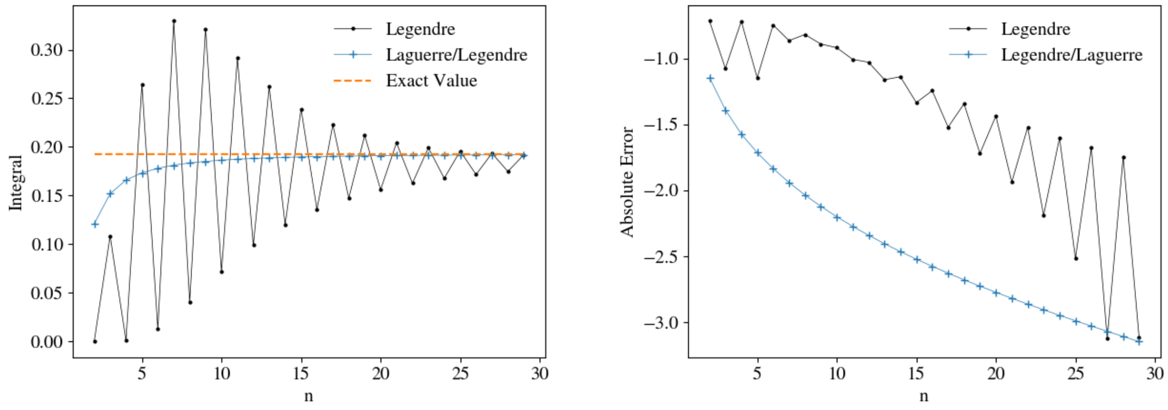

Figure 2: The expected value of particle separation in the Helium atom, determined using numerical integration. Shown in the left figure is the value of the integral, found using Gauss-Legendre, and Gauss-Laguerre quadrature methods, for different numbers of integration points n. Also inset is the analytical value of the expected separation, $5\pi^2/16^2$. The right hand figure shows the absolute error (log 10) of the computed integral, for both methods.

4.2 Monte Carlo Integration

Fig. 3 shows a typical result of calculating (11) using the brute force, as well as importance sampling Monte Carlo integration methods. For the brute force method, the infinity approximation $\lambda = 3$ was used, and the exact integral value $5\pi^2/16^2$ is also indicated. As we can judge from the plot, both methods tend toward the exact value, but oscillate rapidly around it. We can, however, see that the variance is consistently smaller for the importance sampling method.

Table 1: The estimated standard deviation for the brute force and importance sampling Monte Carlo integration algorithms, for different numbers of integration points, n. As the standard deviation tended to oscillate, each listing in the table was found by taking the average of the standard deviation of five runs.

	Standard Deviation (σ_n)	
n	Brute Force	Importance
10	0.25	0.1
100	0.05	0.04
10^3	0.02	0.01
10^4	0.007	0.003
10^5	0.0023	0.0012

Fig. 4 shows the corresponding execution time for the calculations shown in Fig. 3, for both the brute force, and importance sampling methods. While both methods require less than 10^{-4} seconds to compute a 1000-sample Monte Carlo integration, the brute force method is faster than the importance sampling method. In fact, for the entire profile shown in Fig. 4, the brute force method was, on average, approximately 1.4 times faster than the importance sampling method.

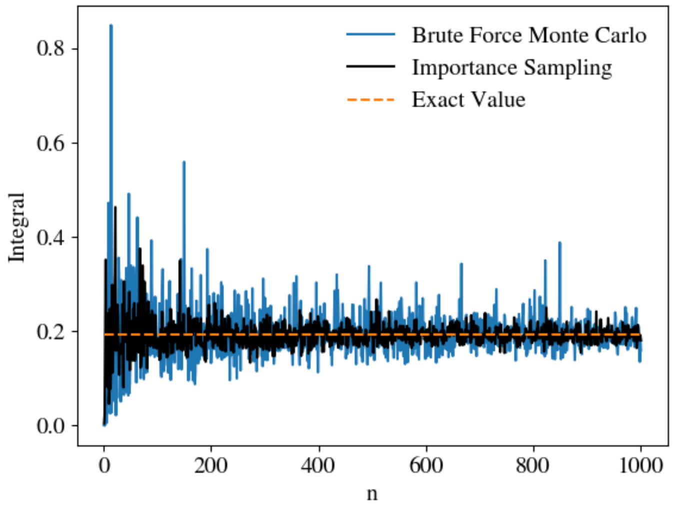

Figure 3: Calculated expected separation between particles in Helium atom, found by numerical Monte Carlo integration for different amounts of integration points, n. Shown is both a brute force approach, with all variables sampled from a uniform distribution. Also shown is the solution to the same integral found using importance sampling, where the radial variables were sampled from an exponential distribution. Also inset is the exact solution, $5\pi^2/16^2$. For the brute force approach, infinity was approximated as $\lambda = 3$.

Table 1 shows the standard deviation, calculated according to (8), for both the brute force, and importance sampling algorithms for different numbers of integration points, n. Each entry in the table is actually the average standard deviation of five independent runs, but this was only done to avoid any outlier values. Judging from the table, the standard deviation achieved using the importance sampling method is approximately half that of the brute force method, for the same number of integration points. Comparing this with the 1.4 times speedup achieved using the brute force algorithm, it appears that the importance sampling method tends to achieve the same accuracy using $1.4/2 \approx 70$ % of the time of the brute force method. In other words, the importance algorithm is roughly 30 % faster than the brute force method, if we take the error as a success measure.

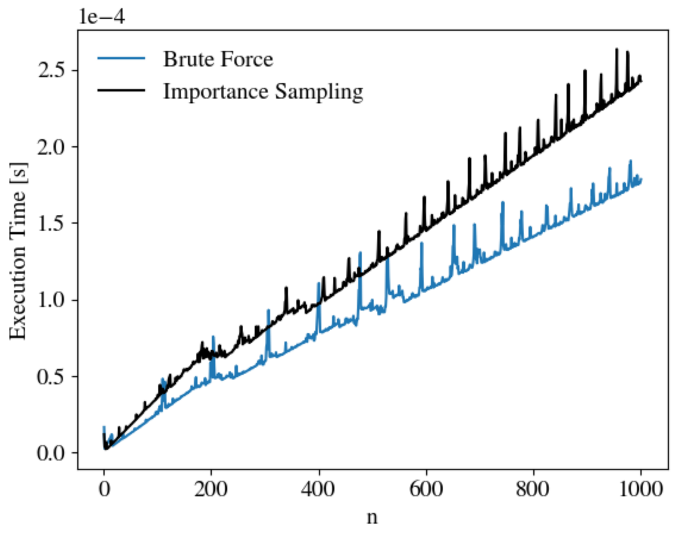

Figure 4: Execution time for the brute force, as well as importance sampling Monte Carlo algorithms, for different numbers of integration points, n.

Fig. 5 shows the execution time for the regular, and parallelized importance sampling Monte Carlo integration algorithms. Each indicated point represents the average of five runs using a given number of integration points, n. We can see that the serial algorithm takes longer to run for all values of n. By inspecting the ratio between the execution times of the programs, it was determined that the parallel program ran between approximately 2.5 and 3 times faster than the serial version (average ratio 2.71, standard deviation 0.05).

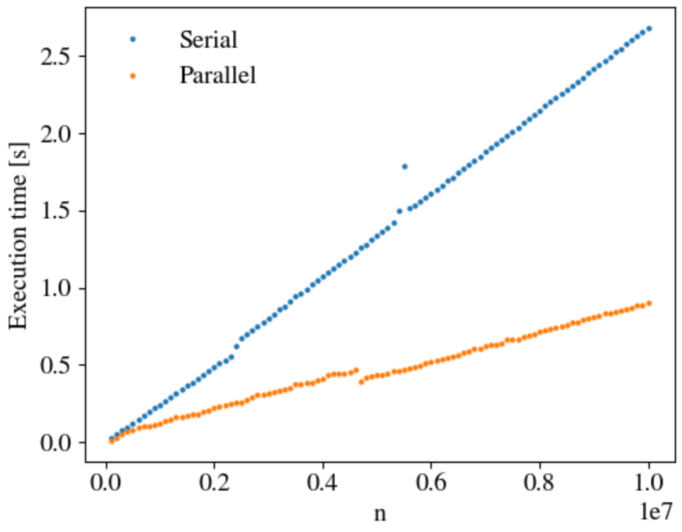

Figure 5: Execution time for the importance sampling Monte Carlo integration algorithm for different numbers of integration points, n. Inset are the execution time for the standard algorithm, as well as for a parallelized version, with an attempted total of 8 parallel processes to perform the same calculation. For each value, for both programs, each indicated execution time was taken as the average of five runs for the same value of n.

5 Discussion

5.1 Gaussian Quadrature

One obvious weakness of the pure Gauss-Legendre quadrature method, was the use of an infinity approximation $\lambda = 3$ in all calculations. While 1 shows that the single-particle wavefunction indeed is small beyond $\mathbf{r} \approx 3$, the nature of the word ”small” is highly subjective. For some applications, this approximation might be suitable, while in others it might lead to sub-par performance. Even though larger values of λ might better approximate infinity, we would also possibly require a corresponding increasing in integration points, at least in the radial dimension.

In stead of directly addressing this issue, we found a clever workaround, by applying the mixed Gauss-Lagurre and Gauss-Legendre quadratures approach. Judging from Fig. 2, one could argue that the mixed approach not only does away with the infinity approximation, it also appears to provide a method which converges faster towards, and oscillates much less around the exact value. However, we should point out that the Gauss-Legendre algorithm only appeared to oscillate when the number of points alternated between even and odd. This is most likely due to the fact that the Legendre polynomials are defined in the interval $[-1, 1]$, i.e. centered around zero. By selecting an odd number of points, one will therefore include the origin, which causes a divergence in the sum of (10). However, as mentioned in the methods section, this problem is avoided by simply skipping the contribution to the integral for very small particle separations r_{12}. As we can see from Fig. 2, the absolute error is lowest for the Gauss-Legendre method when the number of integration points is odd. In fact, the difference in absolute error between odd and even numbers of integration points, is almost a factor of two, for larger values of n.

One possible reason for this is that by skipping the zero-centered value which is included in the odd-valued calculations, what is left is symmetric about the origin. If we skip a contribution for even-numbered values of n, the resulting integration interval might not be as symmetric. While this is currently a bit of conjecture, it would be interesting to explore this subject as a future development.

5.2 Monte Carlo Integration

6 Conclusion

References

[1] M. Hjort-Jensen, ”Chapter 5 - Numerical Integration”, in Computational Physics - Lecture Notes Fall 2015, Oslo: Department of Physics, University of Oslo, 2015, p. 109-125.

[2] M. Hjort-Jensen, ”Chapter 11.4 - Improved Monte Carlo Integration”, in Computational Physics - Lecture Notes Fall 2015, Oslo: Department of Physics, University of Oslo, 2015, p. 364-370.

[3] M. Hjort-Jensen, ”Project 3”, Computational Physics I FYS3150, Oslo: Department of Physics, University of Oslo, Fall Semester, 2019

[4] W. Press, S. Teukolsky, W. Vetterling and B. Flannery, ”Numerical recipes - The Art of Scientific Computing”, 3rd ed. New York: Cambridge University Press, 2007, p. 183-186.

7 Appendix

All code is freely available at https://github.com/markusbp/fys3150/tree/master/project3