A study of numerical integration using various methods, with focus on accuracy, speed and optimization

Anders Bråte and Elin Finstad
Institue for Physics
University of Oslo
Norway
October 25, 2019

Integrating is paramount in many physical situations we may come across in physics. In addition, many calculations we happen upon are so complex that an analytical solution is impossible, or at least undesirable. Considering this, it is quite apparent why having a good grasp of numerical integration is important, as well as the traps that may arise if it is not done correctly. We have looked at four variations of two common integration methods, Gaussian Legendre and Laguerre, a brute force Monte Carlo and a rewritten case of Monte Carlo. We compare these by solving the same quantum mechanical integral, namely the ground state energy between two electrons in an atom. Our focus was numerical precision, computation time and various optimization methods that proved to be relevant. A Gaussian quadrature using Laguerre polynomials proved superior to the similar Legendre method. Laguerre achieved a three decimal accuracy after 69 seconds, while Legendre was not able to achieve this accuracy. The brute force Monte Carlo was quick to achieve a fair level of accuracy with a relative error of 0.0008 after 3.49 seconds, however this algorithm was not able to achieve better accuracy than this. On the other hand, the improved Monte Carlo method was able to achieve four leading digits, and much faster with parallellization.

I. INTRODUCTION

In computer science, and mathematics in general, the integral may be the most important operation used in all aspects of nearly any life science. It then follows naturally that being able to calculate integrals numerically is incredibly important in any aspect of science and mathematics. In this paper we go in depth looking at four variations of two different and fairly commonly used integration methods. The focus will be on numerical precision, effectiveness and optimisation, as this is the main focus in many numerical algorithms.

Initially, we take a look at the Gauss-Legendre method, then a similar Gauss-Laguerre method, and finally the famous Monte Carlo method of integration. As a part of the optimization, we also consider and compare parallellization for the Monte Carlo method of integration. We will use each method to integrate the ground state correlation energy between two electrons in a helium atom. This is a relatively ugly integral, which is preferably solved numerically. However it is also a six dimensional integral, which will prove a challenge.

II. THEORY

The general integration method the Gauss-Legendre and Gauss-Laguerre fall under is adaptive integration. This means that the discretized step length that is necessary for numerical integration is not a set length, as well as each step is not weighed equally. For example a typical step length is given by the total length of the interval given by the integration limits $[a, b]$, and the number of steps n written as $(b - a)/n$. Instead the length of the steps vary based on how it behaves in certain regions. If a function varies a lot in a certain area it needs more integration points to capture the detail, however if doesn't vary much in other areas this might be unnecessary in these areas. In addition to the varying step lengths we can also find how important an interval is, and how much it should count for the final sum.

Exactly how we find the integration points and weights varies with the given method. Mentioned earlier is the Laguerre and Legendre method, which while similar, have a different range and weight function.
In this paper we integrate a quantum mechanical function that describes the ground state correlation energy, for two electrons in a helium atom. This wave function

can be expressed by the following equation.

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

Here $\vec{r_1}$ & $\vec{r_2}$ are the vectors pointing to each electron and $\alpha = 2$, which is the charge of the helium atom.

A. Gaussian Quadrature

Gaussian quadrature is useful when dealing with integrals where the integrand varies slowly over a large interval, and other tailored methods are time consuming. In these cases the Gaussian quadrature can be both more precise and use fewer integration points. For this to happen, it is important that the integrand varies smoothly over the interval. If not, we need to split the interval into smaller pieces, and the gain achieved may be lost.
With this said, we can render a function smooth by writing it as the integral of the product of a new function g and a weight function W.

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

The weight function $W(x)$ which gives us the weights ω varies from Legendre to Laguerre, as will be explained later.

Other methods based on Taylor series use N mesh points while integrating exactly a polynomial P of degree $N-1$. This means that if we can approximate a function $f(x)$ with a polynomial P of degree $N-1$,

$$f(x) \approx P_{N-1}(x),$$

we should be able to integrate exactly the polynomial P_{N-1} with N mesh points. For the Gaussian quadrature rule we can yield an exact result for a polynomial of degree $2N-1$ with the same number of mesh points. In this case the equation reads

$$\int_a^b f(x)dx \approx \int_a^b P_{2N-1}(x)dx = \sum_{i=1}^{N} P_{2N-1}(x_i)\omega_i,$$

for N mesh points.
The reason why we can represent a function $f(x)$ with a polynomial of degree $2N-1$ is due to the fact that we have $2N$ equations, N for the mesh points and N for the weights.

The mesh points as which we evaluate the integral is found by finding the roots of the polynomial P.

B. Gauss-Legendre

The Gauss Legendre method uses Legendre polynomials, which limits us to the interval $x \in [-1, 1]$. This limitation can be worked around , since we can use a change of variables to rewrite the integral. If we use $t = \frac{b-a}{2}x + \frac{b+1}{2}$ we can end up with an integral on the form

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

In practice, the weights of each point as shown in equation 2 is given by the following equation

$$w_i = \frac{2}{(1-x_i^2)[P_n'(x_i)]^2}. \quad (3)$$

Here, the points x_i are the meshpoints in which we evaluate the integral, and P_n' is the solution to the following polynomial

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

These solutions can be written as the following recursive formula

$$(j+1)L_{j+1}(x) + jL_{j-1}(x) - (2j+1)xL_j(x) = 0 \quad (4)$$

for $j = 0, 1, 2, ...$ We pick $L_0(x) = 0$ and $L_1(x) = 1$ and generate values for a desired amount of meshpoints n. This recursive formula is simple to implement numerically.

C. Gauss-Laguerre

If we rewrite the equation 1 to spherical coordinates we can solve the integral by using nearly the same procedure, but using Laguerre polynomials and a different weight function W instead. Naturally, since we are using spherical coordinates we have the integration limits $\theta \in [0, \pi]$, $\phi \in [0, 2\pi]$ and $r \in [0, \infty]$. We can use Legendre for ϕ and θ since these are still limited, and can be rewritten to be within $[-1, 1]$. For r however, we don't need to approximate ∞. This is because Laguerre polynomials are valid in the interval $[0, \infty]$.

For Laguerre we rewrite our expression with the weight function $W(x) = x^\alpha e^{-x}$ where $x \in [-\infty, \infty]$. The polynomials we apply in this method is also different, since we use (you guessed it) Laguerre polynomials. These are similar to Legendre polynomials in that they are recursive, and can be found by the following equation

$$(n+1)L_{n+1}(x) = (2n+1-x)L(x) - nL_{n-1}(x) \quad (5)$$

D. Monte Carlo Integration

The Monte Carlo integration method is a numerical integration method which uses randomly generated variables within the integration limits, instead of evaluating

the integrand at a regular grid. As a result, the method is especially useful for higher order dimensional integrals.

A function $f(x)$ can be interpreted as calculating the area below the function's curve. Assume we want to integrate the function $f(x)$ in the area a to b. By picking a random number x within the interval $[a, b]$ and evaluating the function $f(x)$ at x, followed by multiplying with $(b-a)$, we get the area of a rectangle with length $(b-a)$ and height $f(x)$. If we repeat this several times and divide by the number of repetitions, we get an estimate of the actual area under the function's curve. This is illustrated in Figur 1

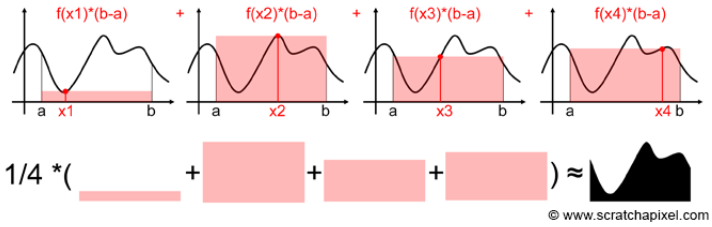

Figure 1. An illustration of how the Monte Carlo integration methods works.

We can formalize this with the following formula

$$\langle F^N \rangle = (b-a)\frac{1}{N}\sum_{i=0}^{N-1} f(x_i),$$

where N is the number of samples used in the approximation, and $\langle F^N \rangle$ is an approximation of F using N samples. The random point in the interval $[a, b]$ can easily be obtained by generating random uniformly distributed numbers in the interval $[a, b]$, or multiply a random number ξ from a uniform distribution in the interval $[0, 1]$ with $(b-a)$:

$$x_i = a + (b-a)\xi.$$

The probability distribution function (PDF) of x_i is $p(x) = 1/(b-a)$.

$\langle F^N \rangle$ is a random variable, since it's a summation of random numbers. We can prove that the expected value of $\langle F^N \rangle$ is equal to F, where we use that the expected value of a function of a random variable can be defined as

$$E[f(x)] = \int f(x)p(x)dx$$

$$\begin{aligned}
E[\langle F^N \rangle] &= E\left[(b-a)\frac{1}{N}\sum_{i=0}^{N-1} f(x_i)\right] \\
&= (b-a)\frac{1}{N}E[f(x)] \\
&= (b-a)\frac{1}{N}\sum_{i=0}^{N-1}\int_a^b f(x)p(x)dx \\
&= (b-a)\frac{1}{N}\sum_{i=0}^{N-1}\int_a^b f(x)\frac{1}{(b-a)}dx \\
&= \frac{1}{N}\sum_{i=0}^{N-1}\int_a^b f(x)dx \\
&= \int_a^b f(x)dx \\
&= F
\end{aligned}$$

The integral's rate of convergence is proportional to the integrand's variance σ^2. Variance means dispersion, and tells how scattered away from the mean the sample is. It is defined as the expected value of the squared difference between the outcome of the experience (in our case, the outcome of the integral) and its expected value, and can be expressed as

$$Var(X) = \sigma^2 = E[(X - E[X])^2] = E[X^2] - E[X]^2$$

We have so far been referring to random numbers, without explaining the complexity of actually finding and using real random numbers. When doing a Monte Carlo integration for many million steps we have to draw an extremely large amount of random numbers, which asks a lot from a given random number generator (RNG). Certain aglorithms for generating pseudo random numbers that aren't as complex have a tendency to 'loop' back when it runs out of new random numbers, meaning the sequence of numbers repeat. We therefore use the Mersenne twister RNG, which has a period of 2^{19937} [1] Despite being fairly old, this RNG will be sufficient for our use.

III. METHOD

A. Gauss-Legendre Quadrature

Our integral can be evaluated to a closed form expression, more precisely $5\pi^2/16^2$ [2]. When dealing with the problem numerically, we have to deal with the problem where $|\mathbf{r}_1 - \mathbf{r}_2| = 0$. In this case the potential will become very large, and diverge towards infinity. To deal with this, we skip the contribution when $|\mathbf{r}_1 - \mathbf{r}_2|$

becomes less than 10^{-10}.
There is also a conscideration to be made for the fact that the Gauss-Laguerre method is valid for the interval $[0, \infty)$. In practice this means we have to approximate infitiy. This can be a large problem, seeing as how computers generally don't take well to infinity. However if we look at the function we wish to integrate we can see that it rapidly nears zero.

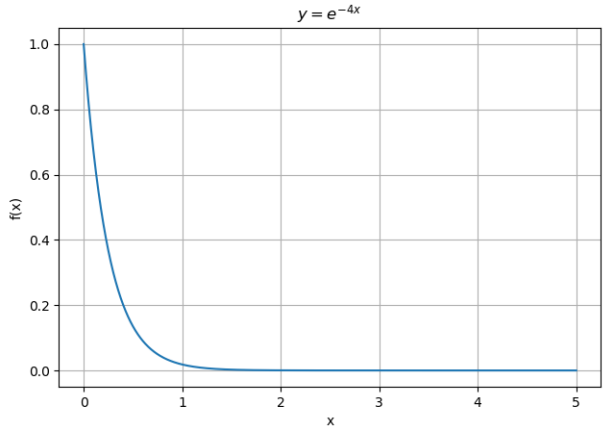

Figure 2. The quantum mechanical wave function for the ground state correlation energy between two electrons in a helium atom. Here we can see that the function value for $x = 1.9$ is nearly zero.

This means that we don't have to use a very large number to approximate infinity. In fact, by $x = 1.9$ the function is almost as close to zero as we could wish for, meaning we don't need to integrate further. We have shown this in fig 3.

B. Parallellization

One important aspect to consider is the option of parallellization. This basically mean we employ all the cpu-cores in the computer and assign each of them with their own calculation, which will improve accuracy and computation time. The way we do this in practice, is by using the numerical library 'open_ MPI'.

For the Monte Carlo method of integration, we call the function with random variables. This means that all the calls to the function are independent. Since they are independent, it is the same if we call the function once with $2n$ mesh points, or call the function twice with n mesh points and take the average. This is useful for parallellization, where we can call the function with n mesh points on several processors at the same time, and find the average of them.

IV. RESULT

A. Gauss-Legendre

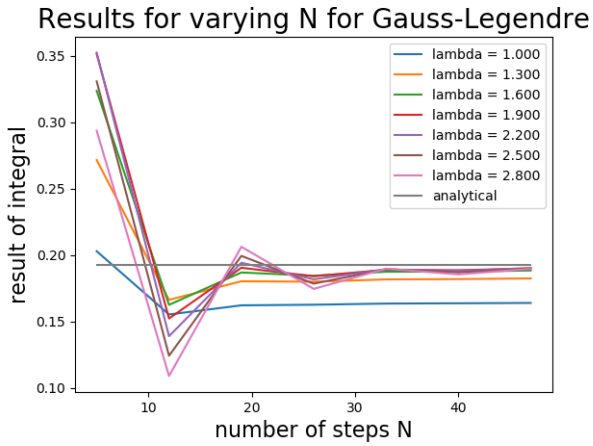

Figure 3. The result of the Gauss-Legendre method for varying N and λ. Here, λ is an approximation to infinity, chosen based on Figur 2. The result converges toward the analytical solution $5\pi^2/16^2$, with $\lambda = 1.9$ as the best approximation to infinity.

Table I. The sum, relative error and cpu time for different amount of mesh points n in the Gauss Legendre method. The results are calculated with $\lambda = 1.9$. The method requires $n = 55$ before the results converges at the level of two leading digits. With this many mesh points, the algorithm runs for 15 minutes.

n	**Sum**	**Relative Error**	**Time [s]**
5	0.351372	0.822791	0.000575797
10	0.13548	0.297176	0.0336064
15	0.196811	0.0209845	0.370313
20	0.178396	0.0745426	2.06743
25	0.188726	0.0209403	7.86556
30	0.186171	0.0342131	23.4714
35	0.189245	0.0182655	59.149
40	0.188735	0.0209118	131.417
45	0.189985	0.0144244	266.525
50	0.189883	0.0149532	501.176
55	0.190498	0.0117643	898.433
60	0.190496	0.0117746	1538.47
65	0.190839	0.00999733	2499.78

B. Gauss-Laguerre

Table II. The sum, relative error and time spent for different amount of mesh points n in the integral for the Gauss Laguerre method. This method converges to a level of three leading digits after 30 mesh points with a cpu time of 69 seconds.

n	Sum	Relative Error	Time [s]
5	0.173450	0.100205	0.00167836
10	0.186457	0.032726	0.101973
15	0.189759	0.015598	1.09328
20	0.191082	0.00873564	6.06895
25	0.191741	0.00531718	24.2661
30	0.192114	0.00338234	69.5402
35	0.192343	0.00219131	174.342

C. Monte Carlo

Table III. The sum, relative error and time spent for different amount of mesh points n in the integral for the brute force Monte Carlo integration method, with $\lambda = 1.9$. This method converges to a level of three leading digits after $1e7$ mesh points with a cpu time of 3.5 seconds. For $n = 1e8$, the precisions decreases, but it increases again for $n = 1e9$.

n	Sum	Relative Error	Time [s]
$1e2$	0.0454452	0.764247	$5.2507e-05$
$1e3$	0.182445	0.0535424	0.00035602
$1e4$	0.201564	0.0456403	0.00347774
$1e5$	0.18495	0.0405448	0.0361585
$1e6$	0.19787	0.0264798	0.352848
$1e7$	0.192922	0.000808746	3.49286
$1e8$	0.190522	0.0116396	35.7272
$1e9$	0.19221	0.00288541	353.574

Table IV. The sum, relative error and time spent for different amount of mesh points n in the integral for improved Monte Carlo integration method. This method converges to a level of three leading digits after $1e8$ mesh points with a cpu time of 45 seconds. For $n = 1e9$ it reaches four leading digits with a cpu time of 7.6 minutes.

n	Sum	Relative Error	Time [s]
$1e2$	0.234156	0.214719	0.00011669
$1e3$	0.205495	0.066033	0.000467745
$1e4$	0.2177545	0.078958	0.00447184
$1e5$	0.189173	0.0186392	0.046939
$1e6$	0.19121	0.00806933	0.44127
$1e7$	0.193029	0.00136711	4.55152
$1e8$	0.192598	0.000870809	45.8664
$1e9$	0.192748	$9.38345e-05$	457.074

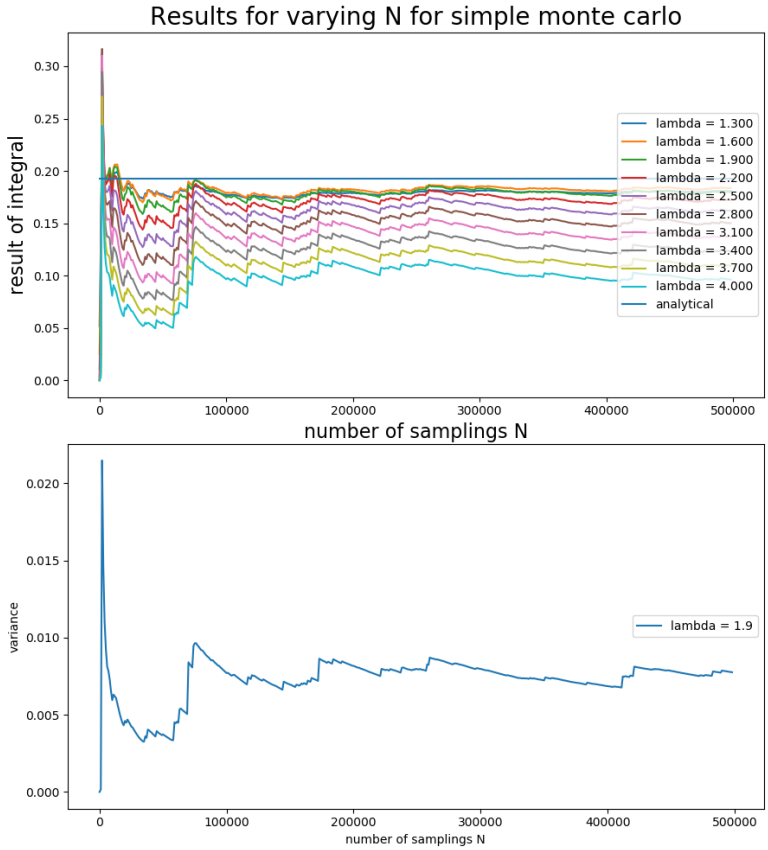

Figure 4. The first plot shows the results of the simple Monte Carlo integration with varying approximations for infinity λ as a function of samplings N. The second plot shows the variance for increasing N with $\lambda = 1.9$. The variance seems to increase when the result gets closer to the closed form solution.

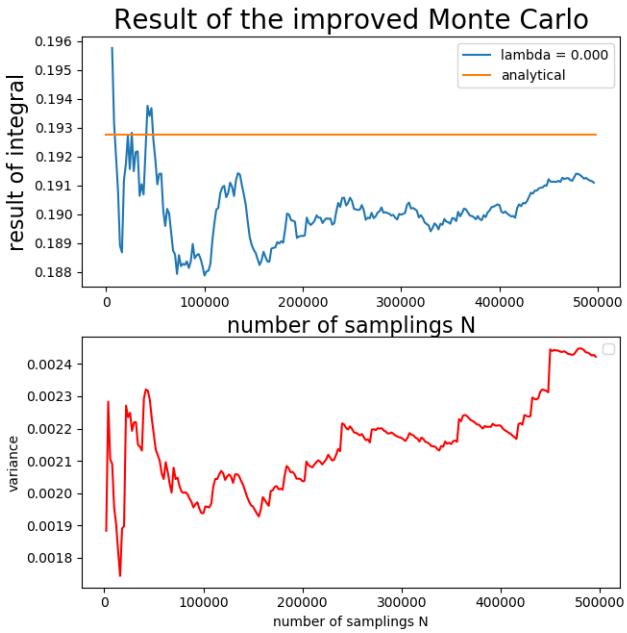

Figure 5. The results of the improved Monte Carlo integration as a function of samplings N. Compared to the figure for the simple Monte Carlo 4, we see that the variance is a whole order of magnitude smaller.

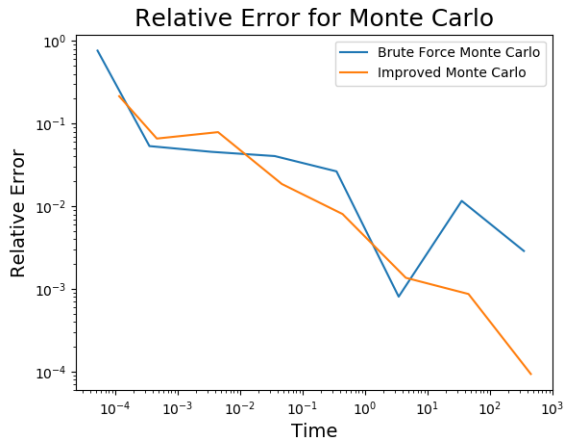

Figure 6. The relative error as the cpu time increases plotted logarithmically. The cpu time for the improved Monte Carlo integration method decreases faster than for the brute force Monte Carlo integration method for the same relative error.

D. Parallellization

Table V. The table shows the cpu time in seconds for the improved Monte Carlo integration, parallellized with two processors and different compiler flags. The different compiler flags are named in the header. -O0, -O1, -O2 and -O3 are optimization level 0, 1, 2 and 3 respectively, where optimization level 0 is the same as no optimization. We can see that the compiler flag -O3 is faster than all the previous compiler flags, but it is faster than -Ofast for smaller values of n. On the other hand, -Ofast is faster for larger values of n.

n	**-O0**	**-O1**	**-O2**	**-O3**	**-Ofast**
$2e2$	0.000259406	0.00021031	0.000260904	0.000175738	0.000285016
$2e3$	0.00167767	0.000585987	0.000715662	0.000649448	0.000676415
$2e4$	0.0166621	0.00544296	0.00635465	0.00678916	0.00670825
$2e5$	0.0518514	0.0299879	0.0318985	0.0311975	0.02579
$2e6$	0.449958	0.200555	0.225249	0.222562	0.175187
$2e7$	4.51406	2.33838	2.25121	2.2238	1.74681

Although it is not included in this rapport, one should note that the Gaussian quadrature also is fit for parallelization. The six for-loops necessary to integrate in all six dimensions are independent of one another, meaning groups of for-loops can be assigned to a cpu for better speed.

V. DISCUSSION

As mentioned in the method section for Gauss-Legendre (although valid for all the methods), we need to check that we don't integrate for values too close to $|\vec{r_1} - \vec{r_2}| = 0$ since this expression is in the denominator, which will lead to very large values (and perhaps division by zero). While doing this we need to be sure not to exclude too many values close to zero, since these should also ideally be included seeing as how it is included in the analytical integral.

1. *Gaussian Legendre*

In our Gauss Legendre function, the number of roots depends on the number of mesh points. This number is calculated by $(n+1)/2$ with integer division. That means we get a better results with an odd number of mesh points compared to an even number of mesh points, since an odd number results in one more root than an even number.

Looking at plot 3 we see that most values of λ approach a fairly close value, except for the very lowest values, which seem to 'miss out' on a lot of the area under the graph. This can be explained by looking at our wave function plot in figure 2. For a $\lambda = 1$ some of the area under the graph is left out of the integral limits.

2. Gaussian Laguerre

By comparing table I and table II, it is clear that the Gauss Laguerre method is way better than the Gauss Legendre method. The Gauss Laguerre method converges to a level of three leading digits after 30 mesh points with a cpu time of 69 seconds. For the Gauss Legendre method, we do not reach the same precision even after 65 mesh points. At this point it takes more than 15 minutes to compute.

3. Monte Carlo

Generally we need to know a little about how our function behaves if we're to use Monte Carlo integration. If we don't know anything about how our function looks, it is best to simply use uniform distribution. This is because if we don't know how our function look, applying a distribution that we don't know fits our function can have adverse effects, and make the algorithm faulty.

The spike seen in plot 4 can be explained by the fact that when r_1 and r_2 is very similar, the function value is very large. A simple 'unlucky' draw of random variables will leave the resulting sum for the integral very large, as per the spike. Drawing more function values with other more reasonable variables will eventually bring the integral down to the correct value. As we can see all the different approximations for λ eventually converge toward the analytical solution.

Compared to the brute force Monte Carlo method, the variance reduces with one order of magnitude for the improved Monte Carlo method. In the brute force Monte Carlo method, we use uniform distribution. For the improved Monte Carlo method, we use exponential distribution to estimate the integrals depending on r. The shape of the exponential distribution is more similar to the shape of the wave function than the uniform distribution, which results in a smaller variance.

Since the wave function is exponential, the variance is larger when the result is closer to the analytical value, as shown in plot 4. If we assume a random number x results in a high function value $f(x)$, while another random number y results in a small function value $f(y)$, then x and y will be far away from each other and cause higher variance. At the same time, the mean of the two rectangles created by x and y create (as explained in II D) will be close to the closed form solution. If x and y are almost equal, the function value of x and y will be almost the same. This results in a small variance, but the mean of the area of the rectangles will be far from the closed form solution.

VI. CONCLUSION

Looking at table I and II we see that Laguerre is by far the superior algorithm. Both in time and number of mesh points needed to compute the value. The Legendre method has a tendency to stagnate, and even for $n = 65$ we only achieve two decimals accuracy. This is partially due to the weight function W in Laguerre being a better fit for our function. The approximation to infinity was found to be good, yet the Legendre algorithm itself is lacking in our case.

The brute force Monte Carlo rapidly achieved a fair accuracy. looking at table III we see that it achieved a three decimal accuracy after only 3.5 seconds. Despite this, it stagnates and doesn't reach three decimal accuracy. The improved Monte Carlo is significantly slower, but is eventually able to beat the brute force Monte Carlo in accuracy, as it reaches four decimal accuracy after 457 seconds.

The use of parallellization for the Monte Carlo methods proved effective, as the speed of the improved Monte Carlo was more than doubled for larger amounts of mesh points. If we look at table V we see that for $n = 2E7$ the optimization flag '-Ofast' improved the integration time from 4.5 seconds, to 1.7 seconds, which is considerable. This proves that these kinds of algorithms lend themselves nicely to parallellization.

In general this paper only considered one fairly niche integral, which in some ways showed the many aspects of these methods. However, more of a broad view of applications might be considered for future research.

Appendix A: Rewrite to Laguerre Polynomials

First we change to spherical coordinates

$$d\mathbf{r}_1\mathbf{r}_2 = r_1^2 dr_1 r_2^2 dr_2 \sin\theta_1 d\theta_1 \sin\theta_2 d\theta_2 d\phi_1 d\phi_2$$

with

$$\frac{1}{r_{12}} = \frac{1}{|\mathbf{r}_1 - \mathbf{r}_2|} = \frac{1}{\sqrt{r_1^2 + r_2^2 - 2r_1 r_2 \cos\beta}}$$

and

$$\cos\beta = \cos\theta_1\cos\theta_2 + \sin\theta_1\sin\theta_2\cos\theta_1 - \theta_2.$$

By applying this we can rewrite our integral on the form

$$\int_0^{2\pi}\int_0^{2\pi}\int_0^{\pi}\int_0^{\pi}\int_0^{\infty}\int_0^{\infty} r_1^2 r_2^2 \sin\theta_1 \sin\theta_2 e^{-2\alpha(r_1+r_2)} \frac{1}{r_{12}} d\epsilon$$

with $d\epsilon = dr_1 dr_2 d\theta_1 d\theta_2 d\phi_1 d\phi_2$.
To be able to use the Laguerre polynomials, we want to rewrite the integrals

$$\int_0^{\infty} r_i^2 e^{-2\alpha r_i} dr_i,$$

where i is 1 and 2, on the form

$$\int_0^{\infty} x^{\alpha} e^{-x} g(x) dx$$

We apply the variable change $r_i' = 2\alpha r_i$ with $dr_i = \frac{dr_i'}{2\alpha}$.

$$\int_0^{\infty} \left(\frac{r_1'}{2\alpha}\right)^2 e^{-r_1'} \frac{dr_1'}{2\alpha} = \frac{1}{(2\alpha)^3}\int_0^{\infty} (r_i')^2 e^{-r_i'} dr_i'$$

We now have $W(r_i') = (r_i')^2 e^{-r_i'}$ and $g(r_i') = 1$.
We also need to apply the variable change to $\frac{1}{r_{12}}$.

$$\frac{1}{r_{12'}} = \frac{1}{\sqrt{\left(\frac{r_1'}{2\alpha}\right)^2 + \left(\frac{r_2'}{2\alpha}\right)^2 - 2\left(\frac{r_1'}{2\alpha}\right)\left(\frac{r_2'}{2\alpha}\right)\cos\beta}}$$

$$= \frac{2\alpha}{\sqrt{(r_1')^2 + (r_2')^2 - 2r_1' r_2' \cos\beta}}$$

In total this results in the integral

$$\int_0^{2\pi}\int_0^{2\pi}\int_0^{\pi}\int_0^{\pi}\int_0^{\infty}\int_0^{\infty} \frac{\sin\theta_1 \sin\theta_2}{(2\alpha)^5\sqrt{(r_1')^2 + (r_2')^2 - 2r_1' r_2' \cos\beta}} d\epsilon \tag{A1}$$

Here $d\epsilon = dr_1' dr_2' d\theta_1 d\theta_2 d\phi_1 d\phi_2$.

Appendix B: Improved Monte Carlo

To improve the Monte Carlo method, we want to rewrite the integral to use exponential distribution and transform to spherical coordinates. The transformation to spherical coordinates is the same as in A. In order to use exponential distribution, we want to factor out e^{-y} from the integrand.

$$I = \int_0^{\infty} F(y) dy = \int_0^{\infty} e^{-y} G(y) dy$$

To do this, we can use the same variable change as in A, with the result

$$\frac{1}{(2\alpha)^3}\int_0^{\infty} (r_i')^2 e^{-r_i'} dr_i'$$

This results in $G(r_i') = (r_i')^2$. We can then multiply this to equation A1

$$\int_0^{2\pi}\int_0^{2\pi}\int_0^{\pi}\int_0^{\pi}\int_0^{\infty}\int_0^{\infty} \frac{\sin\theta_1 \sin\theta_2 (r_1')^2 (r_2')^2}{(2\alpha)^5\sqrt{(r_1')^2 + (r_2')^2 - 2r_1' r_2' \cos\beta}} d\epsilon \tag{B1}$$

again, $d\epsilon = dr_1' dr_2' d\theta_1 d\theta_2 d\phi_1 d\phi_2$.

[1] Makoto Matsumoto and Takuji Nishimura. 1998. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. ACM Trans. Model. Comput. Simul. 8, 1 (January 1998), 3-30. DOI: https://doi.org/10.1145/272991.272995

[2] Jensen, M. H., Lecture in FYS3150 29/8-19.