Numerical integration methods

Henrik Lind Petlund | *github.com/henrikx2*

Date: October 21, 2019

Abstract

This report shows how the Monte Carlo integration algorithm is a superior algorithm in terms of calculating a multi-dimensional integral compared to the Gauss Guadrature integration method. The superiority is both in terms of accuracy and calculation speed. The report also discuss how both methods can be sped up and improved in terms of expressing the integrand in a different basis (spherical coordinates) or using a suitable *Probability Density Function*. The observations in this report are useful to determine which method to use when facing other integration problems.

Introduction

One important integral that finds place in many quantum mechanical systems is the six-dimensional integral defining the ground state corrolation energy between two electrons in a helium atom. This integral is derived by modelling the wave function of each electron as an single-particle wave function of the electron in the hydrogen atom. This is the integral which is to be solved using the four different methods and is chosen in terms of showing why this report has practical significance.

For an electron i in the 1s state, the dimensionless and unnormalized single-particle wave function can be expressed as

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

where α is a parameter, and

$$\mathbf{r}_i = x_i\mathbf{e}_x + y_i\mathbf{e}_y + z_i\mathbf{e}_z$$

with

$$r_i = \sqrt{x_i^2 + y_i^2 + z_i^2}$$

The parameter $\alpha = 2$ gives the charge of the helium atom ($Z = 2$). Further, the wave function of the two 1s electrons are given by

$$\Psi(r_1 + r_2) = e^{-\alpha(r_2+r_2)}$$

The integral which is to be solved is the expectation value of the corrolation energy between the two electrons in the helium atom. The corrolation energy depends on the classical Coluomb interactions of the two electrons, and is given by

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

This (unnormalized) integral can be solved on closed form to be $5\pi^2/16^2 \approx 0.19276571$ (UiO, 2019).

2 Methods

2.1 Gauss Quadrature

Gauss Quadrature is a method that uses orthogonal polynomials with weight functions to estimate integrals and are referenced in (Hjorth-Jensen, 2017). However, the topic is quite extensively to cover for this report and is therefore just explaned in short and otherwise sited.

2.2.1 Gauss-Legendre (GauLeg)

First off is using the Gaussian Quadrature with Legendre polynomials. These polynomials are defined at the interval $x \in [-1, 1]$ with the weight function $W(x) = 1$. The integral in Eq. (1) can be rewritten in terms of dx_i, dy_i and dz_i as

$$\langle \frac{1}{|\mathbf{r}_1 - \mathbf{r_2}|} \rangle =$$

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

Now, every variable is defined on the interval $[-\infty, \infty]$, but since infinity cannot be represented exactly from a numerical point of view, it is here necessery to define infinity as a finite number. This is done to get small enough mesh points, so that the integral becomes more "continous". Figure 1 shows how the function e^{-2r} is approximately zero (< 0.01) when the $r \approx \lambda = 3$. Here, λ is the eigenvalue of the ground state single particle system. This gives that the interval $[-3,3]$ should be sufficient to have three correct leading digits.

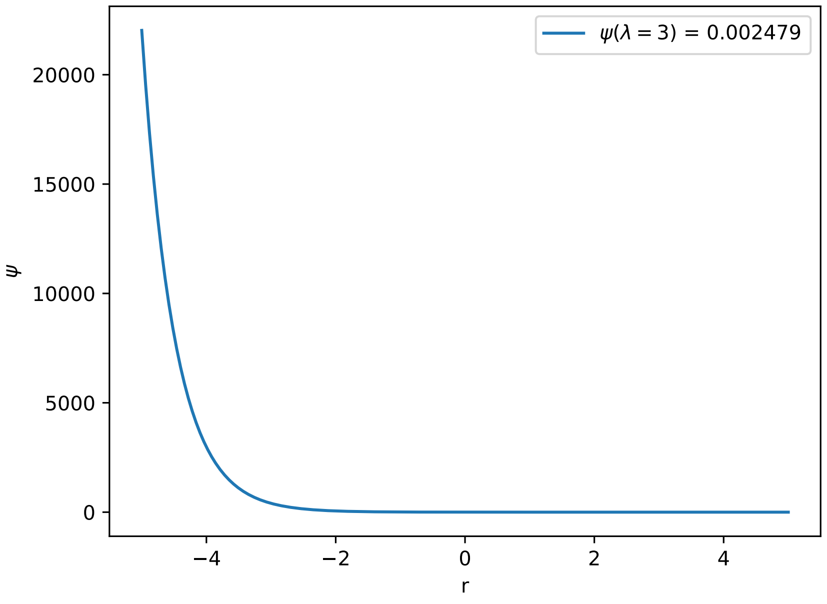

Figure 1: Plot of the wavefunction $\psi = e^{-2r}$ of a single particle in ground state. It's easy to see how the function converges to zero as r increases.

The integral to solve with Gauss-Legendre Quadrature is then given by

$$\int\int\int\int\int\int_{-3}^{3} \frac{dx_1 dx_2 dy_1 dy_2 dz_1 dz_2 e^{-2\alpha(\sqrt{x_1^2+y_1^2+z_1^2}+\sqrt{x_2^2+y_2^2+z_2^2})}}{\sqrt{(x_1-x_2)^2+(y_1-y_2)^2+(z_1-z_2)^2}} \tag{3}$$

This integral is solved in the program `gaussLeg.cpp`.

2.2.2 Gauss-Laguerre (Improved Gauss Quadrature/GauLag)

The Gaussian Quadrature with Laguerre polinomials is defined at the interval $x \in [0,\infty]$ and has the corresponding weight function $W(x) = x^{\alpha'} e^{-x}$ $(\alpha' \neq \alpha)$. By changing to spherical coordinates

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

with

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

and

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

it is possible to rewrite the integral with different integration limits ($\theta \in [0, \pi]$, $\phi \in [0, 2\pi]$ and $r \in [0, \infty\rangle$). This reads

$$\langle \frac{1}{|\mathbf{r}_1 - \mathbf{r_2}|} \rangle =$$

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

where

$$dcos(\theta_1) = -\sin(\theta_1)d\theta$$

such that

$$\langle \frac{1}{|\mathbf{r}_1 - \mathbf{r_2}|} \rangle =$$

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

Among these integrals, it is easiest to map ϕ_1, ϕ_2, θ_1 and θ_2 using Legandre polynomials and r_1 and r_2 using Laguerre polynomials. This is because $\theta \in [0, \pi]$ and $\phi \in [0, 2\pi]$ is easily transformed to $[-1, 1]$ and r is already defined at the interval $[0, \infty]$. Using the weight function, $W(x) = x^{\alpha'} e^{-x}$, with $\alpha' = 0$ (but $\alpha = 2$), the total integrand becomes

$$f(r_1, r_2) = sin(\theta_1)sin(\theta_2)\frac{e^{-3(r_1+r_2)}r_1^2 r_2^2}{r_{12}}$$

where of course there is also possible to set $\alpha' = 2$ and absorb the r's in the weights, such that

$$f(r_1, r_2) = sin(\theta_1)sin(\theta_2)\frac{e^{-3(r_1+r_2)}}{r_{12}}$$

There is also a way of getting rid of the whole exponential expression in the integrand by defining a new variable $r'_i = 4r_i$. Such that

$$f(r'_1, r'_2) = \frac{1}{16} sin(\theta_1) sin(\theta_2) \frac{\frac{1}{16} r'^2_1 \cdot \frac{1}{16} r'^2_2}{\frac{1}{4} \cdot \sqrt{r'^2_1 + r'^2_2 - 2r'_1 r'_2 cos(\beta)}}$$

gives

$$f(r'_1, r'_2) = \frac{1}{1024} \frac{r'^2_1 r'^2_2}{r'_{12}}$$

And with $\alpha' = 2$ the r^2_i's would also be absorbed by the weights, and the final integral value would only have to be multiplied with a factor 1/1024. To avoid loss of numerical precision, integration points where the value $r'^2_1 + r'^2_2 - 2r'_1 r'_2 cos(\beta) < 10^{-10}$ do not contribute to the integration sum.

The integral is solved using namely this last procedure in the program `gaussLag.cpp`.

2.2 Monte Carlo Integration

When using Monte Carlo integration, the descreete integration values are defined using a probability distribution. As long as a sufficient number of psudo-random integration points are chosen, this is supposed to make the numerical approximation of the integral have less error. It is the choice of the probability distribution function (PDF) that determines the precision of the Monte Carlo integration. A thorough explanation of the Monte Carlo methods can found in the lecture notes (Hjorth-Jensen, 2019) of FYS3150.

2.2.1 Brute force Monte Carlo Integration (MCBF)

The brute force Monte Carlo integration uses the uniform PDF given by

$$p(x) = \frac{1}{b-a} \Theta(x-a) \Theta(b-x)$$

where Θ is the Heaviside function and which at the interval $[a, b] = [0, 1]$ gives the function $p(x) = 1$. In the case of Eq. (3) the interval is not $[0, 1]$, but a change of variables such that

$$y(x) = a + (b-a)x$$

where $x \in [0, 1]$ would make it possible to generate random numbers on the general interval $[a, b]$. In a multidimensional integral the change of variable is expressed using the indices i

$$x_i = a_i + (b_i - a_i)t_i$$

Using the integral from Eq. (3), the brute force integrand is given by

$$g(r_1, r_2) = \frac{e^{-2\alpha(\sqrt{x_1^2+y_1^2+z_1^2}+\sqrt{x_2^2+y_2^2+z_2^2})}}{\sqrt{(x_1-x_2)^2+(y_1-y_2)^2+(z_1-z_2)^2}} \tag{4}$$

And the Jacobi-determinant given by

$$\prod_{i=1}^{d}(b_i - a_i) = (b-a)^6$$

which must be multiplied with the integral in the end.

This integral is solved in the program `monteCarloBF.cpp`

2.2.2 Improved Monte Carlo Integration (MCIS)

The improved Monte Carlo method introduces one new aspects to improve the results, namely; the *importance sampling*. In general, when doing importance sampling, one uses a PDF that has similarities with the integrand itself so that parts (or the whole) of this expression can be absorbed in the weights function. In this case (when transforming to spherical coordinates) the integrals with r-dependance would satisfy the exponential distribution given by

$$p(y) = e^{-y}$$

From (Hjorth-Jensen, 2019) this function gives the change of variable as

$$y(x) = -ln(1-x)$$

where x is a random number generated by i.e. the `ran()`-function. Since the exponential expression in the integrand has a factor of $4 = 2\alpha$, it is also necessary to alter the "change of variable"-expression such that $y = 2\alpha y'$ and

$$y'(x) = -\frac{1}{2\alpha}ln(x-1)$$

As for the other integrands with θ and ϕ dependance, the change of variable follows the uniform distribution with $x \in [0,1]$ as follows

$$y(x) = a - (b-a)x = bx$$

After applying this, the integrand will have the form

$$\frac{r_1^2 r_2^2 sin(\theta_1) sin(\theta_2)}{r_{12}}$$

and in the end it's important to multiply this with the Jacobi determinant which reads

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

This integration is solved in `monteCarloIS.cpp`.

2.2.3 Improved Monte Carlo Integration with Parallelization

Parallelization of the program `monteCarloIS.cpp` is done with the use of openMP to see if this gives a considerable speed up.

This program is found in `monteCarloISPar.cpp`.

3 Resulsts

3.1 Speed and error

In Figure 3.1.1, the absolute error is plotted against time usage. The reason for this choice of plot is because the four integration methods have different numbers of integration points and these don't really say much in the combined picture. Table 3.1.1 shows the time usage of each method in order to have the error less than 10^{-3}.

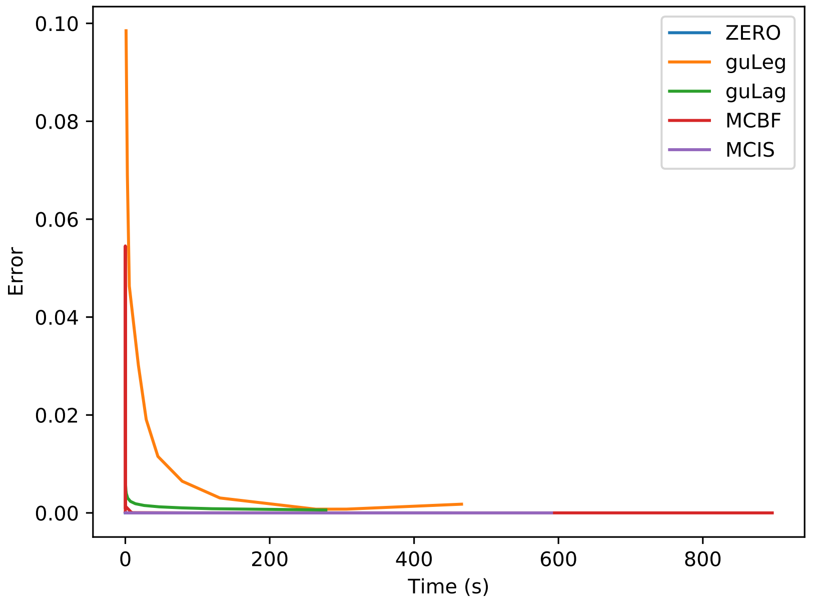

Figure 3.1.1: Plot of the convergance of the error as function of algorithm time usage in the four different cases. This kind of plot reflects how much time is needed to achieve a certain level of accuracy and gives to some extent the superiority of certain algorithms.

Table 3.1.1: Minimal number of steps and time usage to achieve an error less than 10^{-3} for the four different algorithms.

Method	Error	Time (s)	Steps
GaussLeg	0.00075868	263.98	n = 27
GaussLag	0.00085222	120.10	n = 27
MCBF	0.00099854	87.53	n = 10^8
MCIS	0.00024829	0.04978	n = 10^5
MCIS Par.	0.00024829	0.03174	n = 10^5

3.2 Variance

Figure 3.2.1 presents a loglog plot of the variance in the Monte Carlo Brute Force and Importance Sampling methods as function of the number of integration points n.

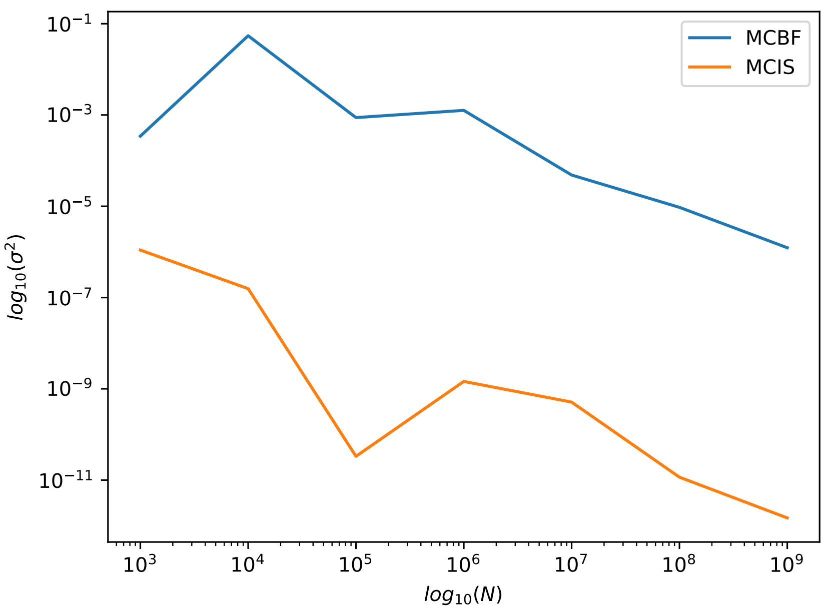

Figure 3.2.1: Loglog plot of the variance in the Brute Force and Importance Sampling Monte Carlo methods as function of integration points.

3.3 Parallelization

Table 3.3.1: The time usage of Monte Carlo Importance Sampling method with and without parallelization (2 threads) as function of n.

Step	Time MCIS (s)	Time MCIS Par (s)	Relative improvement
n = 1000	0.0006171	0.0053874	-7.73
n = 10000	0.0059055	0.0035225	0.37
n = 100000	0.0497836	0.0317402	0.36
n = 1000000	0.4544431	0.1913138	0.58
n = 10000000	5.1644614	1.7690288	0.66
n = 100000000	66.3662704	18.328646	0.72
n = 1000000000	896.1961175	344.31507	0.62

4 Discusson

4.1 Speed and error

According to the Tables 6.1.1-6.1.4, it is obvious that all four approximation methods have the possibility to give precise results with good decimal precision (at least < 0.001). Looking at Figure 3.1.1, it is trivial to see that the Monte Carlo methods are the ones where the error is decreasing fastest as one spends more time (and therefore more integration points) doing the calculations. This is quite naturally a consequence of the fact that there are six for-loops (nested loops) in the Gaussian Quadrature algorithms, and only two loops in the Monte Carlo algorithms. Though, this doesn't mean that both the Monte Carlo algorithms are superior.

In fact, looking at Table 6.1.3, which contains the values of the Brute Force Monte Carlo algorithm, and comparing the minimum error to the both the GaussLeg and GaussLag algorithm, it actually has a larger minimum error. So, the Gaussian Quadrature methods are more prescise than the Brute Force Monte Carlo. Though, looking at the time it takes to reach these values, the table turns somewhat. GaussLeg and GaussLag uses approximately 5 and 2 minutes respectively to calculate the integral with an error of $\sim 10^{-3}$. While MCBF uses approx. 1.5 minutes to achieve almost the same accuracy.

It seems that the MCBF algorithm won't give an error smaller than ~ 0.001, but when applying the importance sampling, the results change rigorously. The MCIS gives an with the error lying in the 6th decimal ($\sim 9 \cdot 10^{-6}$). None of the other algorithms are able to match this accuracy, at least not with the same speed. It seems GaussLag has the potential to reach a smaller error, but when it uses approx. 2 min to have 3 digits precision, one can only imagine what it needs to match MCIS.

MCIS is not only superior when it comes to decimal precision, it also has an impressive speed. Take in example Table 3.1.1, where the time used to have the error in the 4th decimal is given for all algorithms. Simple calculation makes MCIS ~ 2400 times faster than GaussLag! And even faster than GaussLeg.

Compared to the above mentioned speed up from Gaussian Quadrature to Monte Carlo with imporance sampling, the speed up from the use of parallelizaton is not that extreme. Table 3.3.1 shows the relative improvement of speed as function of steps. From this, it reads that parallelizaton may give a total speed up of the factor 0.72. If the goal is to have a large decimal precision, then parallelization is definetly an advantage, but with if less decimal precision is wanted, then the speed of the algorithm is already very good, so the parallelization wouldn't spare that much time.

4.2 Variance

When it comes to the Monte Carlo methods, there is one aspect evolving from the use of Probability Density Functons that is relevant to discuss. This is the behavior of the variance, σ^2.

5 Conclusion

Concluding remarks of the whole projects and all its methods. What can be done further? What waas not satisfying?

6 Appendix

6.1 Tables

In this section is the tables with all experimental data listed. The error values are given as the deviation from the exact solution.

Table 6.1.1: Results from the Gauss-Legendre algorithm for different n-values.

Steps	Integral	Error	Time (s)
n = 5	0.264249	0.071483	0.0086
n = 7	0.329525	0.136759	0.0708
n = 9	0.321518	0.128753	0.4082
n = 11	0.291261	0.098495	1.0169
n = 13	0.261821	0.069055	2.8220
n = 15	0.239088	0.046323	12.228
n = 17	0.222933	0.030167	18.086
n = 19	0.211832	0.019066	28.949
n = 21	0.204307	0.011541	45.114
n = 23	0.199232	0.006466	78.820
n = 25	0.195817	0.003051	130.986
n = 27	0.193524	0.000759	263.981
n = 29	0.191995	0.000771	306.378
n = 31	0.190985	0.001781	465.758

Table 6.1.2: Results from the Gauss-Laguerre algorithm for different n-values.

Step	Integral	Error	Time (s)
n = 5	0.17345	0.0193161	0.006
n = 7	0.18129	0.0114714	0.040
n = 9	0.18520	0.0075633	0.155
n = 11	0.18743	0.0053341	0.508
n = 13	0.18883	0.0039392	1.394
n = 15	0.18976	0.0030068	3.341
n = 17	0.19041	0.0023523	7.092
n = 19	0.19089	0.0018753	13.954
n = 21	0.19125	0.0015171	26.113
n = 25	0.19174	0.0010250	78.434
n = 27	0.19191	0.0008522	120.087

Step	Integral	Error	Time (s)
n = 31	0.19217	0.0005974	277.808

Table 6.1.3: Results from the Monte Carlo Brute Force algorithm for different n-values.

Step	Integral	Error	σ	Variance	Time (s)
n = 1000	0.04532403	0.14744168	0.01853285	0.00034347	0.001179
n = 10000	0.36204871	0.16928300	0.23353019	0.05453635	0.012395
n = 100000	0.15401721	0.03874850	0.02961495	0.00087705	0.097135
n = 1000000	0.21934475	0.02657904	0.03553336	0.00126262	0.886847
n = 10000000	0.18044779	0.01231792	0.00695426	4.836e-05	8.765628
n = 100000000	0.19376425	0.00099854	0.00307761	9.472e-06	87.53368
n = 1000000000	0.19460153	0.00183581	0.00111148	1.235e-06	896.19612

Table 6.1.4: Results from the Monte Carlo Imortance Sampling algorithm for different n-values.

Step	Integral	Error	σ	Variance	Time (s)
n = 1000	0.16817468	0.02459103	3.313e-05	1.098e-06	0.0006171
n = 10000	0.19593839	0.00317268	3.956e-06	1.565e-07	0.0059055
n = 100000	0.19301400	0.00024829	1.821e-08	3.317e-11	0.0497836
n = 1000000	0.19354966	0.00078395	3.810e-08	1.451e-09	0.4544431
n = 10000000	0.19285047	8.476e-05	7.153e-09	5.117e-10	5.1644614
n = 100000000	0.19275082	1.490e-05	3.404e-10	1.159e-11	66.3662704
n = 1000000000	0.19275638	9.327e-06	3.853e-11	1.485e-12	590.4009978

Rererences

Hjorth-Jensen, M., 2017. *Numerical integration, from newton-cotes quadrature to gaussion quadrature. Computational Physics Lectures*, pp.10–32.

Hjorth-Jensen, M., 2019. *Introduction to monte carlo methods. Computational Physics Lectures.*

UiO, 2019. *Project 3; https://github.com/CompPhysics/ComputationalPhysics/tree/master/doc/Projects/2019/Project3.*