Computational Physics - FYS3150
Project 3

Håvard Sinnes Haughom and Ida Risnes Hansen

Oct 22, 2019

Abstract

The integral for the correlation energy between the two electrons in a Helium atom is solved numerically with the quadrature methods Gauss-Legendre and Gauss-Laguerre, as well as with Monte Carlo integration. For the latter both uniform and importance sampling was used. Importance sampling Monte Carlo outperformed the other methods. It reaches three digit accuracy ~ 100 times faster than Gauss-Legendre and $\sim 1\,000$ times faster than uniform sampling Monte Carlo, while Gauss-Legendre did not converge within the scope of the program.

1 Introduction

Modelling microscopic systems often means handling many free variables. Take the two electrons of a simple helium atom. Their positions can be written in terms of dimensionless parameters,

$$\mathbf{r}_i = x_i\mathbf{e}_x + y_i\mathbf{e}_y + z_i\mathbf{e}_z, \qquad r_i = \sqrt{x_i^2 + y_i^2 + z_i^2}.$$

The wave function of the system will then depend on six variables,

$$\Psi(\mathbf{r}_1, \mathbf{r}_2) = e^{-\alpha(r_1+r_2)},$$

where $\alpha = 2$ in the case of the helium atom. This means that solving for the properties describing this small system, often will involve six-dimensional integrals. As an example, the expectation value of the correlation energy between the two electrons,

$$\left\langle \frac{1}{|\mathbf{r}_1 - \mathbf{r}_2|} \right\rangle = \int_{-\infty}^{\infty} \frac{1}{|\mathbf{r}_1 - \mathbf{r}_2|} \Psi^*\Psi \; d\mathbf{r}_1 d\mathbf{r}_2$$

Although a closed form solution exists for this specific problem, we want to explore different numerical methods for solving multi-dimensional integrals on

this form. We want to look at methods with better precision to computational cost ratios than the most intuitive equal-step methods.

In this spirit, we will start by looking at the Gauss-Legendre and Gauss-Laguerre quadrature methods. By optimizing the choice of mesh points and weights, these methods require half the number of iterations for the same accuracy produced by corresponding equal step methods.

While quadrature methods often outperform more primitive methods, they remain slow for high dimensional integrals. In these cases, random sampling and Monte Carlo techniques become better alternatives. Here, we will test Monte Carlo with a uniform distribution, as well as with importance sampling.

We will compare the performance of all four methods, both in terms of precision and computational cost. Finally, we want to explore potential improvements from parallelisation using openMP.

Apart from the challenge of high dimensionality, we are looking at infinite limits and an integrand that is singular wherever $\mathbf{r}_1 = \mathbf{r}_2$. Problems both of which will be addressed.

2 Theory

2.1 Properties of the integral

2.1.1 Cartesian coordinates

The integral in Cartesian coordinates is

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

In its current form, the integral is not particularly nice to deal with; neither singularities nor infinite limits are numerically friendly. While the exponential numerator approach zero fast as the distances $|\mathbf{r}_1|$ and $|\mathbf{r}_2|$ increase, the denominator goes to zero whenever their positions get very close, $\mathbf{r}_1 \approx \mathbf{r}_2$. However, we observe that the bigger r_1 and r_2 is, the closer together the electrons have to be for the integrand values to be meaningful. As an illustration, let $r_1 \approx r_2 = 1$. Then the numerator $e^{-4\cdot 2} \sim 10^{-4}$. Meanwhile, for $r_1 \approx r_2 = 5$, $e^{-4\cdot 10} \sim 10^{-18}$. In the latter case, the electrons have to be about 10^{14} times closer together for a similar contribution to the integral. See Figure 1 below for a two-dimensional qualitative illustration.

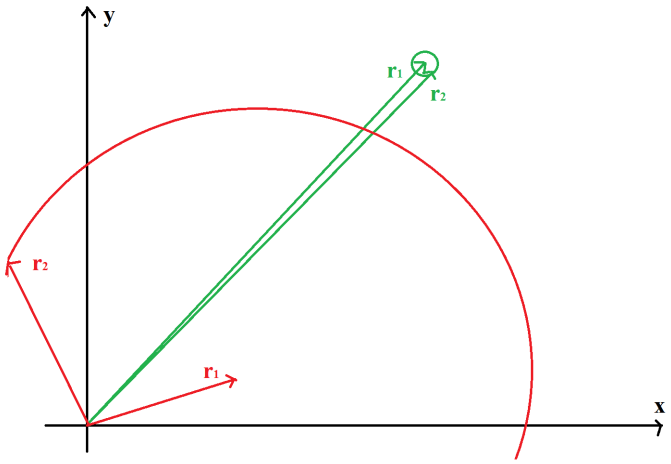

Figure 1: A projection of the electron positions. The bigger the length of $\mathbf{r_1}$, the closer together the electrons have to be for a significant integral contribution.

When the radius of the circles in Figure 1 gets close to zero, so does the relevant domain where the integrand value is significant. The hope is that the effect of the shrinking circle outweighs the blow up of the function value.

While this is far from a proof, it does motivate the change of limits

$$I = \int_{-\infty}^{\infty} \frac{e^{-2\alpha(r_1+r_2)}}{|\mathbf{r}_1 - \mathbf{r}_2|} d\mathbf{r}_1 d\mathbf{r}_2 \approx \int_{-5}^{5} \frac{e^{-2\alpha(r_1+r_2)}}{|\mathbf{r}_1 - \mathbf{r}_2|} d\mathbf{r}_1 d\mathbf{r}_2.$$

A crude way of addressing the remaining singularities, is simply discarding contributions where the denominator is very close to zero, using the same argument as above. While perhaps not the most elegant solution, the idea is straightforward and easy to implement.

2.1.2 Spherical coordinates

Another way of approaching the integral, is to rewrite it in spherical coordinates.

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

where

$$\frac{1}{r_{12}} \equiv \frac{1}{\sqrt{r_1^2 + r_2^2 - 2r_1 r_2 \cos(\beta)}}, \qquad \cos(\beta) \equiv \cos(\theta_1)\cos(\theta_2) + \sin(\theta_1)\sin(\theta_2)\cos(\phi_1 - \phi_2).$$

While we still have to deal with singularities, we will see that the infinite radial limits can be handled in a natural way in several of the methods.

2.2 Gaussian quadrature

The implementation of Gaussian quadrature methods is on the form,

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

$W(x)$ is called a weight function. It is continuous and non-negative in the domain $[a, b]$ and give rise to a system of polynomials $\mathcal{P}$ that are orthogonal in the same interval.

$$\int_a^b \mathcal{P}_i(x)\mathcal{P}_j(x) = 0 \quad \text{for } i \neq j, \tag{4}$$

A polynomial of degree $N-1$ can be expressed in terms of these polynomials,

$$Q_{N-1}(x) = \sum_{k=0}^{N-1} \alpha_k \mathcal{P}_k(x), \tag{5}$$

where the α_k's are constants. Together with equation 4, this gives

$$\int_a^b \mathcal{P}_N(x)Q_{N-1}(x) = \sum_{k=0}^{N-1} \alpha_k \int_a^b \mathcal{P}_N(x)\mathcal{P}_k(x) = 0.$$

With this logic, an integral of a polynomial of degree $2N-1$ can be expressed

$$\int_a^b P_{2N-1}(x)dx = \int_a^b (\mathcal{P}_N(x)P_{N-1} + Q_{N-1}(x))dx = \int_a^b Q_{N-1}(x)dx. \tag{6}$$

In particular, at the zeros x_k of $\mathcal{P}_N$,

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

In other words, if we choose gridpoints defined by the zeros x_k of $\mathcal{P}_N$, this amounts to approximating the integrand $f(x)$ with a polynomial of degree $2N-1$ and getting an exact integral over the polynomial, using only N gridpoints.

$$\int_a^b f(x)dx \approx \int_a^b P_{2N-1}(x)dx = \sum_{i=0}^{N-1} P_{2N-1}(x_i)\omega_i \tag{7}$$

It only remains to find the appropriate weights ω_i. Using equation 5, we can evaluate Q_{N-1} at the zeros x_k.

$$Q_{N-1}(\mathbf{x}_k) = \begin{bmatrix} \mathcal{P}_0(x_0) & \mathcal{P}_1(x_0) & \dots & \mathcal{P}_{N-1}(x_0) \\ \mathcal{P}_0(x_1) & \mathcal{P}_1(x_1) & \dots & \mathcal{P}_{N-1}(x_1) \\ \dots & \dots & \dots & \dots \\ \mathcal{P}_0(x_{N-1}) & \mathcal{P}_1(x_{N-1}) & \dots & \mathcal{P}_{N-1}(x_{N-1}) \end{bmatrix} \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \dots \\ \alpha_{N-1} \end{bmatrix} \equiv \hat{\mathcal{P}}\alpha$$

This gives the constants

$$\alpha = \hat{\mathcal{P}}^{-1} Q_{N-1}(\mathbf{x}_k) \tag{8}$$

Using equations 4 and 5, we can express

$$\int_a^b Q_{N-1}(x)dx = \sum_{i=0}^{N-1} \alpha_i \int_a^b \mathcal{P}_0(x)\mathcal{P}_i(x)dx = \alpha_0 \int_a^b \mathcal{P}_0(x)\mathcal{P}_0(x)dx \equiv \alpha_0\beta,$$

where β is a constant that depends on the polynomial $\mathcal{P}_0$. Using equation 8,

$$\int_a^b P_{2N-1}(x)dx = \int_a^b Q_{N-1}(x)dx = \alpha_0\beta = \sum_{i=0}^{N-1} \left(\beta \hat{\mathcal{P}}_{0i}^{-1}\right) P_{2N-1}(x_i)$$

Comparing it with equation 7, the weights must be $\omega_i = \beta\hat{\mathcal{P}}_{0i}^{-1}$.

Then for a given weight function $W(x)$ associated with the polynomials $\mathcal{P}$, the integral can be solved according to equation 3, where the N gridpoints and N weights are defined by the properties of $\mathcal{P}$. This amounts to approximating the integrand by a $2N-1$ polynomial, while only using N points (Jensen, 2017).

2.2.1 Gauss–Legendre

For Gauss–Legendre quadrature, the weight function is,

$$W(x) = 1 \qquad x \in [-1, 1],$$

while the associated polynomials are defined by

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

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

2.2.1.1 Changing the limits

An arbitrary finite domain $[a, b]$ can be changed to fit Gauss-Legendre.

$$t = \frac{b-a}{2}x + \frac{b+a}{2}$$

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

2.2.2 Gauss–Laguerre

For Gauss–Laguerre quadrature, the weight function is

$$W(x) = x^{\alpha}e^{-x} \qquad x \in [0, \infty),$$

and the Laguerre polynomials can be found through

$$\int_0^{\infty} e^{-x}\mathcal{L}_n(x)^2 = 1$$

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

2.3 Monte Carlo

Monte Carlo integration is a non-deterministic method based on random sampling of integrand values. In simple terms, the idea is based on the mean of a variable with a probability distribution $p(x)$,

$$\langle x \rangle = \int xp(x)dx.$$

From this, we can rewrite the integral

$$I = \int_a^b f(x)dx = \int_a^b P(x)\frac{f(x)}{P(x)}dx = \left\langle \frac{f(x)}{P(x)} \right\rangle \approx \frac{1}{N}\sum_{i=1}^{N}\frac{f(x_i)}{P(x_i)} \tag{9}$$

where $P(x)$ is a probability distribution function with domain $[a, b]$. The associated variance is

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{f(x_i)}{P(x_i)}\right)^2 - \left(\frac{1}{N}\sum_{i=1}^{N}\frac{f(x_i)}{P(x_i)}\right)^2$$

(Jensen, 2019).

2.3.1 Uniform sampling

A general uniform distribution in the interval $[a, b]$ is given by

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

Equation 9 is then on the form

$$\int_a^b f(x)dx \approx \frac{1}{N} \sum_{i=1}^{N} \frac{f(x_i)}{b-a} \tag{10}$$

where the $x_i \in [a, b]$ are sampled from the uniform distribution.

$$x_i = a + (b-a)y_i, \qquad y_i \in [0, 1] \tag{11}$$

2.3.2 Exponential sampling

For exponential integrands and limits on the form $[0, \infty]$, uniform sampling is not ideal. By instead using an exponential distribution function

$$p(x)dx = \alpha e^{-\alpha x} dx, \tag{12}$$

the sampling should on average give more relevant integrand values.
The integral is then

$$\int_0^\infty f(x)dx \approx \frac{1}{N} \sum_{i=1}^{N} \frac{f(x_i)}{\alpha e^{-\alpha x_i}} \tag{13}$$

With $x_i \in [a, b]$ are sampled from an exponential distribution.

$$x_i = -\frac{1}{\alpha} \ln(1 - y_i), \qquad y_i \in [0, 1] \tag{14}$$

3 Method

All four methods are implemented in the file *Integrate.cpp* [1]. Most of the functions are copied or closely based on similar ones from the FYS4150 course repository (Jensen, 2019). See footnotes for address to each function.

3.1 Gauss-Legendre

The relevant functions are

- double integrandCartesian(double r1[3], r2[3])
  Returns the integrand value for given electron positions according to the Cartesian integrand in equation 1.
- void pointsGaussLegendre(double a, b, double* x, w, int N)[2]
  Finds the Legendre polynomials up to L_N, calculates the meshpoints and weights, given an integral [a,b], and lets x* and w* point at them.
- double gaussLegendre(int N)[2]
  Calls pointsLegendre() to get the relevant meshpoints and weights. Loops over the integrand for each of the six variables according to equation 15 below and returns the integral estimate.

For the Gauss-Legendre method, we use the Cartesian integrand. As discussed in Section 2.1.1, we let $\int_{-\infty}^{\infty} \rightarrow \int_{-5}^{5}$, so the limits $[a, b] = [-5, 5]$ for all six variables. Through change of variables in pointsGaussLegendre(),

$$\int_{-5}^{5} f(x_i) dx_i = 5 \int_{-1}^{1} f\left(5\tilde{x}_i\right) d\tilde{x}_i.$$

The meshpoints and weights are the same for all variables, and the calculation becomes

$$I \approx \sum_{i,j,k,l,m,n=1}^{N} w_i w_j w_k w_l w_m w_n \, f((x_i, x_j, x_k), (x_l, x_m, x_n)) \tag{15}$$

where $f =$ integrandCartesian.

We record the result and time usage for $N = 5, 10, \ldots, 35, 40$.

[1] github.com/f0rmIdabel/FYS4150-Project3/blob/master/IntegralSolver/Integrate.cpp

[2] `https://github.com/CompPhysics/ComputationalPhysics/blob/master/doc/Projects/2019/Project3/CodeExamples/exampleprogram.cpp/`

3.2 Gauss-Laguerre

We use the functions

- double integrandSpherical(double r[2], theta[2], phi[2])
  Returns the integrand value for given electron positions according to the spherical integrand in equation 2.
- void pointsGaussLaguerre(double* x, w, int N, double α)[2]
  Finds the Laguerre polynomials up to $\mathcal{L}_N$, calculates the meshpoints and weights, given the parameter $\alpha = 2$, and lets x* and w* point at them.
- void pointsGaussLegendre(double a, b, double* x, w, int N)[2]
  Finds the Legendre polynomials up to L_N, calculates the meshpoints and weights, given an integral [a,b], and lets x* and w* point at them.
- double gaussLaguerre(int N)[2]
  Calls pointsLaguerre() and pointsLegrende() to get the relevant meshpoints and weights. Loops over the integrand for each of the six variables according to equation 16 below and returns the integral estimate.

We use the spherical form of the integrand. We handle the angular parts with Gauss-Legendre, while we use Gauss-Laguerre for the radial part.

$$I \approx \sum_{i,j,k,l,m,n=1}^{N} w_{r,i} w_{r,i} w_{\theta,k} w_{\theta,l} w_{\phi,m} w_{\phi,n} \, f((r_i, r_j), (\theta_k, \theta_l), (\phi_m, \phi_n)) \qquad (16)$$

The radial weights $w_{r,i}$, $w_{r,j}$ are found by pointsGaussLaguerre(), while the remaining angular weights are found by pointsGaussLegrende(), but with $[a,b]_\theta = [0, \pi]$ and $[a,b]_\phi = [0, 2\pi]$.

We record the result and time usage for $N = 5, 10, \ldots, 35, 40$.

3.2.1 Parallelisation

- double gaussLaguerrePara(int N)
  Similar to gaussLaguerre(), but with openMP parallelisation of innermost for-loop

3.3 Monte Carlo, uniform sampling

The relevant functions are

- double integrandCartesian(double r1[3], r2[3])

- double bruteMonteCarlo(int N)[3]
  Generates 6 random variables in the interval $[a, b]$ with the uniform distribution function given in equation (11), using the ran0 function (Press et al., 2002). Loops over N times to evaluate integral I_i and square of integral I_i^2 for variance.

We use the Cartesian integrand and the same approximation as before, $\int_{-\infty}^{\infty} \to \int_{-5}^{5}$, so for each variable we divide by $\frac{1}{b-a} = \frac{1}{10}$. The integral calculation becomes

$$I \approx \frac{1}{N} \sum_{i,j,k,l,m,n=1}^{N} \frac{f(x_i, x_j, x_k, x_l, x_m, x_n)}{10^6}, \tag{17}$$

while the variance

$$\sigma^2 = \frac{1}{N} \sum_{i=1}^{N} (I_i)^2 - I^2. \tag{18}$$

We record the result and time usage for $\log 10(N) = 1, 2, \ldots, 8$.

3.4 Monte Carlo, exponential sampling

We use

- double integrandSpherical(double r[2], theta[2], phi[2])
- double impsampMonteCarlo(int N)[3] Generates 2 radial variables from the exponential distribution given in equation (14), using the ran0 function (Press et al., 2002). While 4 radial variables are generated from a uniform distribution, where $[a, b]_\theta = [0, \pi]$ and $[a, b]_\phi = [0, 2\pi]$. Loops over N times to evaluate integral I_i and square of integral I_i^2 for variance.

We use the spherical integrand. For each of the θ and ϕ variables, we divide by $\frac{1}{\pi}$ and $\frac{1}{2\pi}$, respectively. For the radial parts, we use $p(x) = 4e^{-4x_i}$. The calculation becomes

$$I \approx \frac{1}{N} \sum_{i,j,k,l,m,n=1}^{N} \frac{f((r_i, r_j), (\theta_k, \theta_l), (\phi_m, \phi_n))}{(2\pi)^2 \pi^2 4e^{-4r_i} 4e^{-4r_j}}, \tag{19}$$

while the variance is on the same form as equation 18. We record the result and time usage for $\log 10(N) = 1, 2, \ldots, 8$.

[3] `http://compphysics.github.io/ComputationalPhysics/doc/pub/mcint/html/mcint.html`

4 Results

4.1 Accuracy

4.1.1 Quadrature methods

Below is a plot of the integral results from the quadrature methods for different number of steps. Gauss-Legendre does not reach three digit accuracy within the range of $N \leq 40$ we tested. Gauss-Laguerre converges more quickly and is accurate within three digits when $N = 30$.

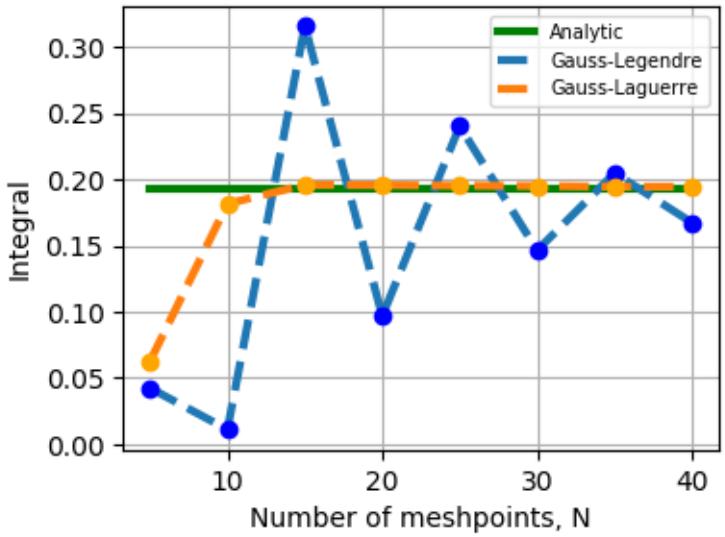

Figure 2: Integral values calculated by the Gauss-Legendre and Gauss-Laguerre quadrature methods for different number of steps. Gauss-Laguerre converges quickly after N=10, while Gauss-Legrende behaves like a slowly converging dampened oscillation.

We were not able to get the parallelisation to work properly. For a long time, we struggled to make openMP run at all. When it finally did, the time usage decreased, but the results blew up massively.

4.1.2 Monte Carlo methods

Below is a plot of the results from the Monte Carlo methods. The brute-force case gets accurate to three digits at $N = 10^8$, but is still unstable at this point. With importance sampling, the integral converges quickly and is accurate within three digits when $N = 10^4$.

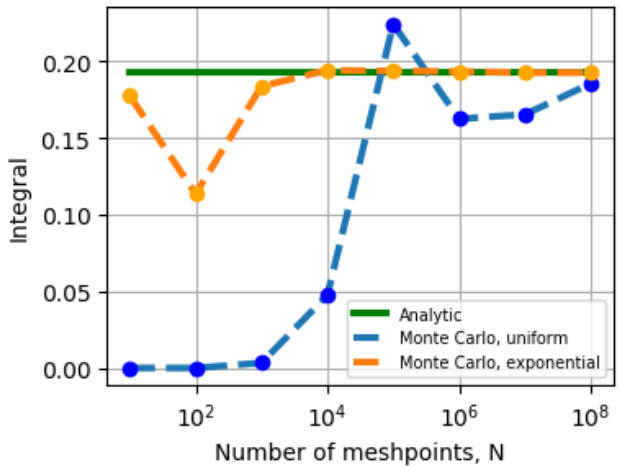

Figure 3: Integral values calculated by the Monte Carlo integration with uniform sampling and importance sampling. The latter converges quickly after $N = 10^3$, while the uniform sampling case is still unstable at $N = 10^8$.

In figure 4, we see the variance with exponential sampling decreases for all N. For uniform sampling, it starts out small, but blows up after $N = 10^2$. It starts to decrease after $N = 10^5$, just when the error decreases significantly (see figure 3).

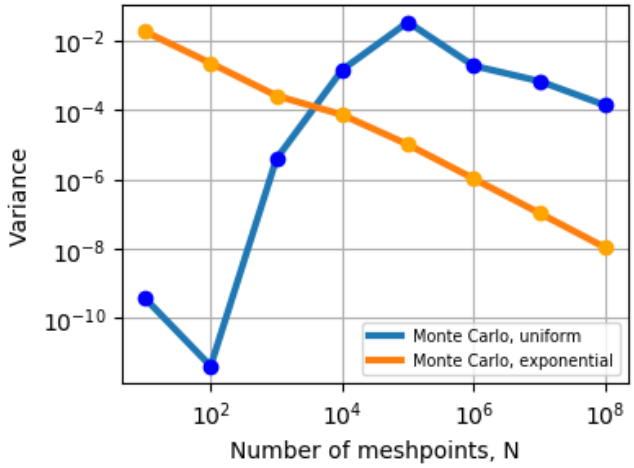

Figure 4: The variance of the Monte Carlo results for different numbers of steps. With importance sampling, the variance decrease consistently with N. With uniform sampling, the variance first blows up, and decreases for big $N > 10^5$.

4.2 Efficiency

Figure 5 shows error development over time for each of the four methods. Importance sampling Monte Carlo is the clear winner. It is ~ 100 times faster than Gauss-Laguerre and $\sim 1\,000$ times faster than brute force Monte Carlo.

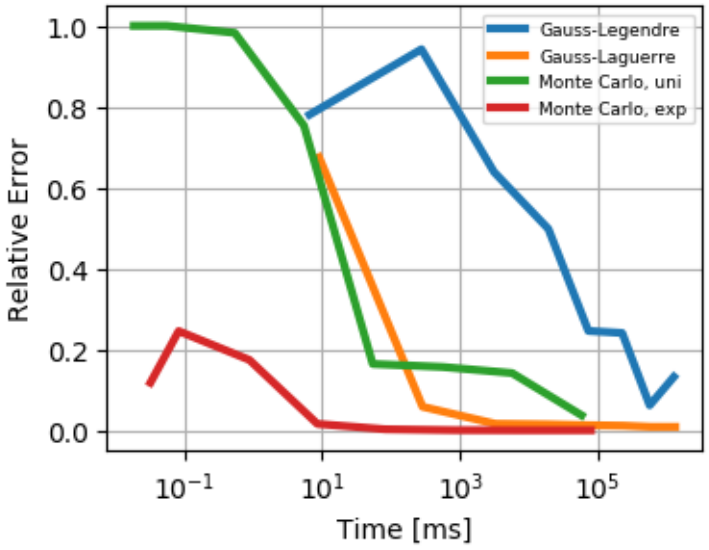

Figure 5: The relative error of each method over time. The importance sampling Monte Carlo clearly performs best, followed by Gauss-Laguerre. Gauss-Legendre performs worst.

5 Discussion

In many ways, figure 5 is a good summary of this project. Gauss-Laguerre and importance sampling Monte Carlo outperform their more primitive counterparts. Both take the integrand into consideration, something which pays of in accuracy. They both avoid the infinite limit problem and evaluate smoother integrands.

The points and weights from the Laguerre polynomials give a better approximation to the radial integral than the Legendre polynomials does to the Cartesian ones. Meanwhile, the exponential sampling Monte Carlo gives a better distribution of integral points than the uniform one.

The Monte Carlo methods are significantly faster than the quadrature ones. This is likely due to the high dimensionality of the integral. The variance of the sample mean decreases continuously in importance sampling Monte Carlo. This is expected as $\sigma_N^2 \propto N^{-1}$. In the brute-force case it is more unstable. This is another manifestation of the choice of the exponential PDF.

6 Conclusion

The brute-force methods, Gauss-Leguerre and uniform sampling Monte Carlo, perform worse than their counterparts. The former does not reach three digit accuracy within the 21 minutes it took to run for $N = 40$, while the latter takes $\sim 10^4$ times longer than importance sampling Monte Carlo. This should maybe be no surprise as they are implemented without considering the properties of the integrand. There is no one-method-fits-all when it comes to solving integrals, and there is a lot to be gained by giving the problem in question a little thought before writing a solver. While the Laguerre quadrature method performs well within the scope of this project, our results suggest that Monte Carlo is the better choice for solving this six dimensional integral.

References

[1] Jensen, M. H., *Computational Physics Lectures: Numerical integration: Newton-Cotes quadrature to Gaussian Quadrature* (2017). `http://compphysics.github.io/ComputationalPhysics/doc/pub/mcint/html/mcint.html`

[2] Jensen, M. H., *Computational Physics Lectures: Introduction to Monte Carlo methods* (2019), GitHub repository: `github.com/CompPhysics/ComputationalPhysics/`

[3] Jensen, M. H., *Computational Physics* (2019), GitHub repository: `github.com/CompPhysics/ComputationalPhysics/`

[4] Press, W.H. Teukolsky, S.A. Vetterling, W.T. Flannery, B.P. (2007) *Numerical Recipes: The Art of Scientific Computing* (Section 7.1.) (2nd ed.), New York: Cambridge University Press.