FYS4150 Project 3

Jens Bratten Due

October 2019

Abstract

Contents

1 Introduction

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

When $\lambda = 2$, the solution to this integral is $5\pi^2/16^2$, and we will be using two different methods for approximating this value, as well as evaluating their accuracy and efficiency. The first method used is the Gaussian Quadrature using Legendre polynomials, and later, Laguerre polynomials. The second method is Monte Carlo integration, firstly using a brute force approach and lastly using importance sampling.

2 Method

2.1 Gaussian Quadrature

The general idea of Gaussian Quadrature is approximating an integral as a sum of function values at mesh points x_i multiplied with specific weights ω_i:

$$\int_a^b W(x)f(x) = \sum_{i=1}^{N} \omega_i f(x_i) \tag{2}$$

Where $W(x)$ is the weight function. If the integrand is a polynomial of a degree $2N-1$, the Gaussian Quadrature will give the exact solution. By using different polynomials, we can then determine the weights and mesh points such that the solution is approached. These polynomials are orthogonal, and we will be using two different types, namely Legendre and Laguerre polynomials.

2.2 Gauss-Legendre

Using Legendre polynomials corresponds to setting the weight function $W(x) = 1$, as well as demanding a finite integration interval (as Legendre polynomials are defined for [-1,1]). Therefore, we must approximate infinity. This is done with a constant λ, by using the fact that the single-particle wave function $e^{-\alpha r_i}$ approaches zero for a sufficiently large $r_i \approx \lambda$. Beyond this point, the integral will essentially be zero, and can therefore be ignored. Furthermore, we must account for the factor $|\mathbf{r}_1 - \mathbf{r}_2|$ being close to zero. When this occurs, the sum will blow up, leading to useless numbers. With great audacity, and a fragment of guilt, we thereby omit these contributions completely and carry on. To find which λ is large enough, we will make a plot of the function against a set of λ-values. From here, we will then investigate how many integration points N are needed to arrive at a satisfactory approximation of the analytical value, as well as measuring CPU time.

2.3 Gauss-Laguerre

Our second approach is to use Laguerre polynomials. These are defined for $[0, \infty]$ and correspond to the weight function $W(x) = x^a e^{-x}$. To achieve the correct interval, we transform the integral 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

$$\frac{1}{r_{12}} = \frac{1}{\sqrt{r_1^2 + r_2^2 - 2r_1r_2cos(\beta)}}$$

and

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

Our integral now takes the form

$$\int_0^\infty \int_0^\infty \int_0^\pi \int_0^\pi \int_0^{2\pi} \int_0^{2\pi} r_1^2 r_2^2 \frac{e^{-4r_1}e^{-4r_2}}{r_{12}} dr_1 dr_2 dcos(\theta_1) dcos(\theta_2) d\phi_1 d\phi_2 \quad (3)$$

The angles θ, ϕ are still defined on finite intervals, so we will use Legendre polynomials for these.

2.4 Brute force Monte Carlo integration

Our second method of solving our integral involves gambling, in a sense. By randomly picking points in the interval [a,b], and evaluating our function at these points, we are able to approximate the integral. The integral

$$I \approx \frac{V}{N} \sum_{i=1}^{N} f(x_i) \quad (4)$$

for large numbers. Here, $V = (b - a)^6$ is the volume of the domain of the integral, and x_i is a random, uniformly distributed, sample from the domain.

2.5 Improved Monte Carlo

We will now switch the distribution of our random sample from uniform to exponential as our integrand contains an exponential function. We then transform to spherical coordinates and approximate as we did for the brute force method.

2.6 Parallelization

Lastly, we will be using the MPI library for parallelizing the Monte Carlo integration with importance sampling and investigate how much time we can save by doing these optimizations.

3 Results and discussion

3.1 Gauss-Legendre

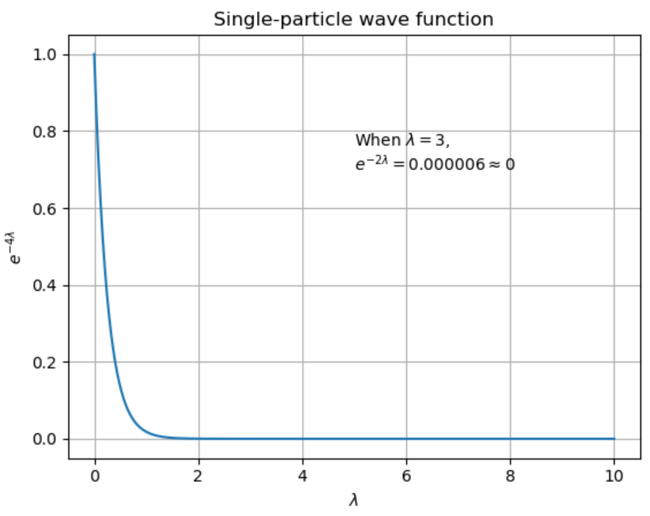

Figure 1: Figure showing the single-particle wave function plotted against different lambdas. At $\lambda = 3$ the function is essentially zero. The integration limits for Gauss-Legendre are then chosen to be -3 and 3, as contributions beyond this interval will have a negligible effect on the end result.

Table 1: Results from Gauss-Legendre integration. The analytical value is gradually approached, but is not able to reach the exact value to a satisfactory degree as number of integration points increase. Even at N = 50, resulting in the program needing up to an hour of run time, the method only reaches a value of 0.18756.

N	Approximation	Analytical	CPU time
10	0.07198	0.19277	0.27 s
15	0.23909	.	3.0 s
20	0.15614	.	14.9 s
25	0.19582	.	58.1 s
30	0.17728	.	163.8 s

3.2 Gauss-Laguerre

Table 2: Results from Gauss-Laguerre integration. Here we see an impproved trend over the method using only Legendre polynomials. The approximations are better and much more stable. The CPU time is slightly longer for every N, but the accuracy is significantly greater. At N = 30, we reach an approximation that is correct to the third decimal point.

N	Approximation	Analytical	CPU time
10	0.18646	0.19277	0.4 s
15	0.18976	.	4.1 s
20	0.19108	.	22.5 s
25	0.19174	.	85.7 s
30	0.19211	.	259.5 s

3.3 Brute force Monte Carlo

Table 3: Results from Brute force Monte Carle integration. The method reaches a satisfactory approximation as N increases, with some fluctuations and instability however. The values do change for each run, due to the inherent randomness. Demanding a large number of integration points is crucial for relying on this method. On the bright side, this method is significantly faster than the Gauss quadratures.

N	Approximation	Analytical	σ	CPU time
10^5	0.22267	0.19277	0.085552	0.04 s
10^6	0.19019	.	0.017682	0.29 s
10^7	0.17908	.	0.007875	3.0 s
10^8	0.19275	.	0.003639	31.1 s

3.4 Improved Monte Carlo

3.5 Parallelization

Table 4: Results from Brute force Monte Carle integration. Very stable and accurate approximations as N increases. The fluctuation seen in the brute force method is drastically reduced, and a satisfactory value is reached for smaller N. This modification of Monte Carlo integration is highly beneficial. The computation time has doubled, but is still vastly superior compared with the Gauss quadratures.

N	Approximation	Analytical	σ	CPU time
10^5	0.19437	0.19277	0.003288	0.099 s
10^6	0.19298	.	0.001012	0.58 s
10^7	0.19272	.	0.000335	6.0 s
10^8	0.19279	.	0.000104	60.1 s

Table 5: Results from parallelizing with 4 threads, as well as using compiler flag -O3 on the improved Monte Carlo integration. We can see there is roughly a 10-fold increase in speed for all N. A regrettable consequence of this procedure is that we lose accuracy, the approximations are now not nearly as close as before. This may be due to a strange anomaly from the fact that the programs have been run in Linux in a virtual machine, or more likely, a wrong implementation of the parallelization. It was assumed that the random number generator was not giving different values for each thread, but this was pesumably fixed when using the id of each thread to seed the RNG. The system time was also attempted as a seed, to no avail. Alas, the problem endured and the solution was not found.

N	Approximation	CPU time	Non-parallelized
10^5	0.18062	0.01 s	0.099 s
10^6	0.17138	0.05s	0.58 s
10^7	0.17473	0.5 s	6.0 s
10^8	0.17677	6.2 s	60.1 s

4 Conclusion

5 Bibliography

This report is made possible by lecture slides and code examples in the course, made available by Morten Hjorth-Jensen.

[1] Hjorth-Jensen, M., n.d. Overview of course material: Computational Physics. `http://compphysics.github.io/ComputationalPhysics/doc/web/course` (accessed 21.10.19).

6 Appendix

All programs and results can be found here: `https://github.com/jensbd/FYS4150/tree/master/Project3`