FYS3150 – Computational Physics
Project 3

Magnus Ulimoen

October 19, 2015

1 Introduction

The goal of this project is to find the correlation energy between two electrons in the helium atom ground state. We do not have the analytical solution to this problem, so it is taken as an ansatz that the electrons can be superpositioned from hydrogen 1s-orbitals.

The program made to solve this problem can be found under the folder `Project3` in the github repository at `https://github.com/mulimoen/FYS3150CompPhy`, where all necessary information on how to run the program is included.

2 Method

2.1 Wave function

The 1s-orbital for hydrogen has a wave function given by

$$\psi(\mathbf{r}) = R_n(r)Y_l^m(\theta, \phi) \tag{2.1}$$

For hydrogen in 1s, the wave function does not depend upon angles,

$$\psi(r) = Ce^{-\alpha\frac{r}{a_0}} \tag{2.2}$$

Where C is a normalisation constant and α is a parameter determing the strength of the potential. For our helium-atom this is chosen $\alpha = Z = 2$, corresponding to a charge of $2e$ for the nucleus of the helium atom. r will be used in natural units, where $a_0 = 1$, and the normalisation constant is neglected.

The wave function for the electrons can not be found exact, but an ansatz can be taken, that the wavefunctions is simply two overlapping hydrogen 1s-orbitals.

$$\psi(r_1, r_2) = e^{-\alpha(r_1+r_2)} \tag{2.3}$$

2.2 Interaction energy

Since the electrons are not localized, but rather spread out over all the space, it is necessary to find the expectation value of the energy. Classically the energy between two charged objects are given by

$$V = k\frac{qQ}{|\mathbf{r}_1 - \mathbf{r}_2|} = k\frac{qQ}{|\mathbf{r}_{12}|} \tag{2.4}$$

This suggest that the quantum mechanical interaction energy is the ensemble over all the space,

$$E_{\text{interaction}} \propto \left\langle \frac{1}{|\mathbf{r}_1 - \mathbf{r}_2|} \right\rangle \tag{2.5}$$

Which can be written out

$$E_{int} \propto \iint \psi^* \frac{1}{|\mathbf{r}_{12}|} \psi \, \mathrm{d}\mathbf{r}_1 \mathrm{d}\mathbf{r}_2 = I \tag{2.6}$$

An analytical solution of this is

$$I = \frac{5\pi^2}{16} \tag{2.7}$$

2.2.1 Cartesian coordinates

The integral (2.6) takes the following form in cartesian coordinates,

$$I = \iiiiint \frac{e^{-2\alpha(|\mathbf{r}_1|+|\mathbf{r}_2|)}}{|\mathbf{r}_{12}|} \mathrm{d}x_1 \mathrm{d}y_1 \mathrm{d}z_1 \mathrm{d}x_2 \mathrm{d}y_2 \mathrm{d}z_2 \tag{2.8}$$

with all the limits from $-\infty$ to ∞.

2.2.2 Polar coordinates

Since the integral constains a large amount of radial elements, a change to polar coordinates leads to better algorithms and solutions.

In order to change into polar coordinates we need to have the Jacobi determinant, also called volume element $\mathrm{d}V$. In polart coordinates this takes the form

$$\mathrm{d}V = r^2 \sin\theta \mathrm{d}r\mathrm{d}\theta\mathrm{d}\phi \tag{2.9}$$

The term $|\mathbf{r}_1 - \mathbf{r}_2|$ is expressed in these coordinates

$$|\mathbf{r}_1 - \mathbf{r}_2| = |r_1\mathbf{e}_{r_1} - r_2\mathbf{e}_{r_2}| = \sqrt{(r_1\mathbf{e}_{r_1} - r_2\mathbf{e}_{r_2}) \cdot (r_1\mathbf{e}_{r_1} - r_2\mathbf{e}_{r_2})} \tag{2.10}$$

$$= \sqrt{r_1^2\mathbf{e}_{r_1}^2 + r_2^2\mathbf{e}_{r_2}^2 - 2r_1r_2\mathbf{e}_{r_1} \cdot \mathbf{e}_{r_2}} \tag{2.11}$$

The inner product of $\mathbf{e}_{r_i}$ with itself is one. It is necessary to find the dot product $\mathbf{e}_{r_1} \cdot \mathbf{e}_{r_2}$ which is the angle between the two unit vects, $\cos\beta$. Writing out the radial unit vector in polar coordinates,

$$\mathbf{e}_r = \cos\phi\sin\theta\mathbf{i} + \sin\phi\sin\theta\mathbf{j} + \cos\theta\mathbf{k} \tag{2.12}$$

The inner product of the two radial unit vectors are

$$\cos\beta = \cos(\theta_1)\cos(\theta_2) + \sin(\theta_1)\sin(\theta_2)\cos(\phi_1 - \phi_2) \tag{2.13}$$

And the integral (2.6) takes the form

$$I = \int_0^{2\pi}\int_0^{2\pi}\int_0^{\pi}\int_0^{\pi}\int_0^{\infty}\int_0^{\infty} \frac{r_1^2 r_2^2 e^{-2\alpha(r_1+r_2)}\sin(\theta_1)\sin(\theta_2)}{\sqrt{r_1^2 + r_2^2 - 2r_1r_2\cos\beta}} \mathrm{d}r_1\mathrm{d}r_2\mathrm{d}\theta_1\mathrm{d}\theta_2\mathrm{d}\phi_1\mathrm{d}\phi_2 \tag{2.14}$$

3 Numerical approximations

To solve this integral some precautions must be taken. A straight forward method requires all of the space to be integrated, which is unfeasable numerically. The function should only be integrated where it is interesting, which is close to origo to due the exponential part. A suitable limit is selected based on where the wave function is approximately zero.

Another problem is the singularity that arises when $\mathbf{r}_{12}$ is zero. This is solved by not letting this part contribute if this occurs, as this volume is of very limited size and should not contribute much to the value of the integral.

If weights and mesh points are used we obtain a large amount of intertwined variables that needs to be permuted. This gives a complexity of N^6 calculations for the integral. This requires effective weights and mesh points to solve, and Gauss Quadrature is used. Another method is Monte Carlo integration which does not depend on weights.

3.1 Gauss Quadrature

The Gauss Quadrature uses orthogonal polynomials to capture the function in its space and gives weights and meshpoints which is summed over. The two methods used are based on Legendre polynomials and Laguerre polynomials.

3.1.1 Legendre

The Legendre quadrature uses the weight function

$$I = \int_a^b f(y)dy = \int_{-1}^{1} W(x)f(x)dx \approx \sum_i w_i f(x_i) \tag{3.1}$$

and meshpoints as the zeros of the Legendre polynomial. The legendre polynomials are defined for integrals with limits $[-1, 1]$. A linear mapping is used, which maps the limits to $[0, \text{limit}]$.

3.1.2 Laguerre

This Laguerre quadrature uses the weight function

$$I = \int f(y)dy = \int_0^\infty \frac{W(x)}{x^{\tilde{\alpha}}e^{-x}} f(x)dx \approx \sum_i w_i f(x_i) \tag{3.2}$$

In (2.14) we have a factor which strongly resembles the dividing factor of the above equation. If the choice $x_i = 2\alpha r_i$ is made and $\tilde{\alpha} = 2$, the integral simplifies to

$$I = \int\cdots\int \frac{r_1^2 e^{-x_1}}{x_1^2 e^{-x_1}} \frac{r_2^2 e^{-x_2}}{x_2^2 e^{-x_2}} \frac{W(x_1)W(x_2)}{|\mathbf{r}_{12}|} \cdots \mathrm{d}r_1 \mathrm{d}r_2 \cdots \tag{3.3}$$

Changing the variables and considering

$$|\mathbf{r}_{12}| = \frac{|\mathbf{x}_{12}|}{2\alpha} \tag{3.4}$$

the integral simplifies further,

$$I = \frac{1}{(2\alpha)^5} \sum_{i,j,\cdots} \frac{w(x_{1,i})w(x_{2,j})}{|\mathbf{x}_{12,ij}|} \sin\theta_{1,k} \sin\theta_{2,l} w(\theta_{1,k})w(\theta_{2,l})w(\phi_{1,m})w(\phi_{2,n})$$

With the sum taken over all the permutations of $i, j, \cdots$.

3.2 Monte Carlo

The Monte Carlo approach is picking out the numbers randomly from a range and then calculating the average value of the function with these numbers. As the number of guesses increases the integral should approach the true value. Since the Monte Carlo is random there is also a need to get the variance in order to get an error estimate. The variance is calculated through

$$\mathrm{Var}(X) = \left\langle x^2 \right\rangle - \langle x \rangle^2 \tag{3.5}$$

To compare over several values of N the standard deviation is used,

$$\sigma = \sqrt{\frac{\mathrm{Var}(X)}{N}} \tag{3.6}$$

The random number generators should produce a uniform distribution in the range $[0, 1]$. Since the integral selected here does not have these limits a change of variables is needed. This could either be used in a straight forward method of linear mapping, or some more clever approches could be employed.

3.2.1 Brute force

The integral (2.8) can be solved by setting the artifical limits $[-\mathrm{limit}, \mathrm{limit}]$ and doing a linear mapping to these limits. This causes the volume $\mathrm{d}x_1\mathrm{d}y_1\cdots$ to depend on the limit chosen as $(2\,\mathrm{limit})^6$, which can be imaginged as the size of the "box" which is integrated over. A pseudo-code for this is

```
gen = random_number(-limit, limit)

repeat N times {
x1 = gen()
x2 = gen()
y1 = gen()
  ...

r1 = x1*x1 + y1*y1 + z1*z1
r2 = x2*x2 + ...

r12 = sqrt((x1-x2)*(x1-x2) + ...)
f = exp(-2*alpha*(r1 + r2)/sqrt(r12)
sum = sum + f
```

```
square_sum = square_sum + f*f
} // end repeat

norm_factor = (2*limit)^6

sum = sum*norm_factor
square_sum = square_sum*norm_factor^2

mean = sum/N*norm_factor
Var = square_sum/N  - mean*mean
std_dev = sqrt(Var/N)
```

Another way to implement the brute force Monte Carlo is using the polar coordinates. The sphere is then mapped to a cube with the linear mapping $\theta = [0, \pi]$, $\phi = [0, 2\pi]$, $r = [0, \text{limit}]$. The volume of this box is now $\pi^2(2\pi)^2(2\text{limit})^2$.

3.2.2 Importance sampling

The brute force method does not follow the rapidly varying nature of the function which is integrated. The sampling of the random variables should therefore be adapted to take this into account. Both the angular part and the radial part is considered.

Simplifying the spherical part is done by considering the factor $\sin\theta \mathrm{d}\theta$. Comparing this with $\cos\theta$,

$$\frac{d\cos\theta}{d\theta} = \sin\theta \tag{3.7}$$

$$d\cos\theta = \sin\theta d\theta \tag{3.8}$$

If now replacing $\cos\theta$ with x, and let this variable be selected by the uniform distribution in the range $[-1, 1]$ we can rewrite (2.13) to use this choice of variable,

$$\cos\beta = x_1x_2 + \sqrt{1-x_1^2}\sqrt{1-x_2^2}\cos(\phi_1 - \phi_2) \tag{3.9}$$

Which also give the normalisation factor 2 to the integral.

For the radial part an exponential distribution is chosen. This distribution transforms the integral to

$$\int_0^\infty p(y)f(y)dy = \int_0^1 \frac{f(y(x))}{e^{-y}}dx \tag{3.10}$$

With y given by the inverse of the distribution

The integral in (2.14) has the form

$$\int_0^\infty dr\ r^2 e^{-2\alpha r} f(r) \cdots \tag{3.11}$$

Making the change of variables

$$y = 2\alpha r = \frac{r}{\lambda} \tag{3.12}$$

$$y = -\lambda \ln(1-x) \tag{3.13}$$

Where x is the random number generated by a RNG. The integral needs to be transformed to be usable with these new numbers,

$$I = \lambda^2 2^2 \int_0^{2\pi}\int_0^{2\pi}\int_0^{1}\int_0^{1}\int_0^{1}\int_0^{1} \frac{y_1^2 y_2^2}{|\mathbf{y}_{12}|} \mathrm{d}x_1 \mathrm{d}x_2 \mathrm{d}x_3 \mathrm{d}x_4 \mathrm{d}\phi_1 \mathrm{d}\phi_2 \tag{3.14}$$

Or as a sum

$$I = \frac{(2 \cdot 2\pi \cdot \lambda)^2}{N} \sum \frac{y_1^2 y_2^2}{|y_1^2 + y_2^2 - 2y_1 y_2 \cos\beta|} \tag{3.15}$$

4 Implementation

The programming language used is C++, and the code is available in the github repository listed in section 1 which also contains a README on how to use this program. The program source code is under `src`. A short description of the algorithms used follows in this section.

4.1 Gauss Quadrature

4.1.1 Pure Legendre

Mapping of the integral is done by a call to a function which gets the weights and mesh points by finding the roots of the appropriate polynomail. This is integrated by looping over the six variables which all uses the same weights and meshpoints, and summing the appropriate sum.

4.1.2 Laguerre and Legendre

The weights and meshpoints are found by a call to the appropriate function. The angles θ and ϕ are still mapped using the Legendre method. The summing of the appropriate sum gives the integral.

4.2 Monte Carlo

In order to extract the random numbers a RNG is used. The function used is the Mersenne Twister from the standard library, defined in the <random> header. To avoid initializing the same seed only one generator is used in this program. This might cause a loss of performance if operating with several threads, but this is assumed to be negligible.

These algorithms are coded so that they can be run in parallel using OpenMP. This should give a speedup of the program execution.

4.2.1 Brute force

The RNG is bound with the real uniform distribution defined in the standard library. For the cartesian coordinates the mapping $[-\text{limit}, \text{limit}]$ is used. In polar coordinates three different distributions are made to get the appropriate limits.

4.2.2 Importance sampling

The RNG is bound with the exponential distribution defined in the standard library. A lambda function is used so a call to this function will combine four random numbers into $\cos\beta$. The sum is then calculated and the mean and standard deviation is calculated.

5 Results

The analytical result is known to be

$$I = \frac{5\pi^2}{16} = 0.192765711 \tag{5.1}$$

5.1 Gauss Quadrature

The results for the Gauss Legendre approach is in table 1. The Gauss Laguerre approach are listed in table 2. Both of these are plotted against the relative error in figure 1.

Table 1: Results for the Gauss-Legendre approach

N	Result	Relative error	Limit
10	0.071979677	0.627	3
10	0.011164658	0.942	5
10	0.000011218	1.000	10
15	0.239088291	0.240	3
15	0.315862997	0.639	5
15	0.135901531	0.295	10
20	0.156139391	0.190	3
20	0.096788762	0.498	5
20	0.009799632	0.949	10
25	0.195816565	0.016	3
25	0.240135013	0.246	5
25	0.307075670	0.593	10
30	0.177282956	0.080	3
30	0.146370604	0.024	5
30	0.051521493	0.733	10
35	0.189923139	0.014	3
35	0.204704919	0.062	5
35	0.298264908	0.547	10

Table 2: Results for the Gauss-Laguerre approach

N	Result	Relative error
10	0.186457345	0.0327
15	0.189758982	0.0156
20	0.191081780	0.0087
25	0.191740740	0.0053
30	0.192113712	0.0033
35	0.192343301	0.0021

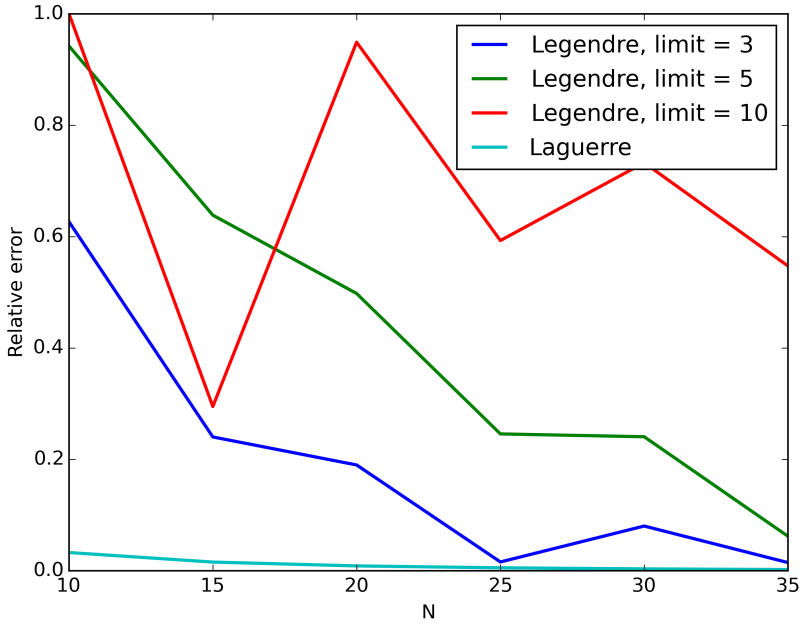

Figure 1: Plot of the relative error for the Gauss Legendre and Gauss Laguerre approach with different values of N and the limit

5.2 Monte Carlo

A table over the result of typical Monte Carlo runnings is included in table 3, table 4 and table 5. A figure which shows the estimated value of the integral for the different algorithms are included in figure 2.

The implementation of OpenMP leads to a decrease of time for the execution. It is however not optimal, as the speedup is 1.5 when using 4 threads, compared to an optimal speedup of 4.

6 Discussion

The algorithms used here do all tend to converge to the analytical solubtion if suitable parameters are used. But since the rate of convergence for some of these are too low, they will never converge to a meaningful value before roundoff errors and computational complexity overtakes it.

The approach based on Gauss-Legendre is unstable and good parameters must be selected to ensure that it comes close enought to the solution. The Laguerre approach gives far more satisfiable approximations, and quicly reaches three digit precision.

Table 3: Results for Monte Carlo in cartesian coordinates

N	Result	Standard deviation
10	0.000000008	0.000000007
100	0.001133467	0.001125540
1000	0.024285372	0.023123862
10000	0.014531879	0.005365095
100000	0.194244256	0.086522718
1000000	0.117016553	0.027317216
10000000	0.206257811	0.040739848
100000000	0.185448419	0.012420672

Table 4: Results for Monte Carlo in polar coordinates

N	Result	Standard deviation
10	0.000376852	0.000145215
100	0.231511633	0.126265106
1000	0.157525515	0.027329668
10000	0.212161377	0.016812296
100000	0.197445741	0.004107642
1000000	0.206297434	0.007374732
10000000	0.198070682	0.000595095
100000000	0.200285570	0.000732391

Table 5: Results for Monte Carlo importance

N	Result	Standard deviation
10	0.240731444	0.133228794
100	0.412266740	0.177025271
1000	0.169150179	0.025264961
10000	0.190142914	0.006359748
100000	0.193869185	0.002517490
1000000	0.193403734	0.000843820
10000000	0.195505813	0.000325820
100000000	0.195270296	0.000270642

Table 6: Program time in seconds for the parallelisation in MC methods, run with N = 10000000, * marks number of virtual cores on the tested computer

Number of threads	Time MCC	Time MCP	Time MCI
1	1.50	2.75	2.17
2	1.26	2.12	1.85
4*	0.97	1.75	1.44
8	0.99	1.77	1.48

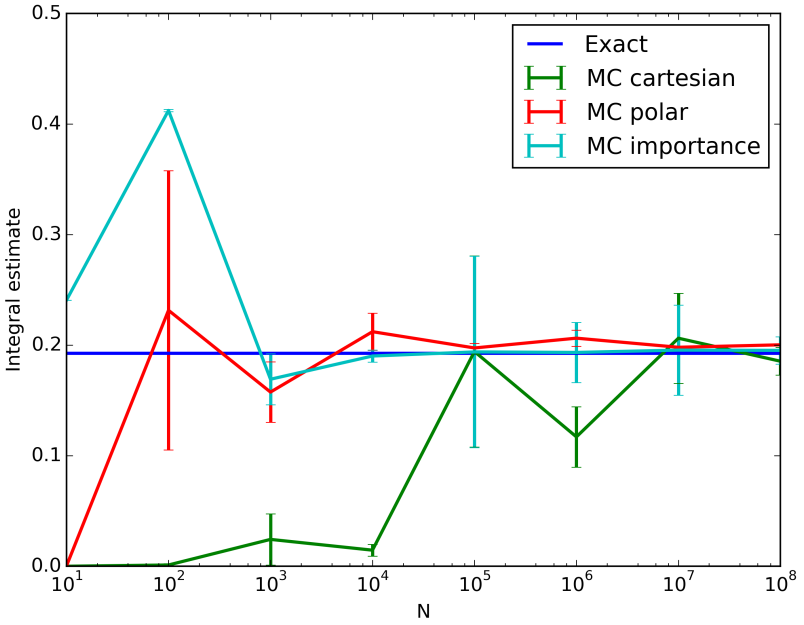

Figure 2: Values of the integral along with the standard deviation as error bars for different values of N for the three methods employed.

As the Monte Carlo methods rely on randomness the estimate can be quite off the true analytical solution. With higher values of N all the three algorithms converges to the true solution, but the rate of convergence is differing. The brute force method based on cartesian coordinates do not converge to more than a digit for $N = 10^8$ as the sampling does not capture the function. Changing to polar coordinates gives a better way to sample the space, and leads to better results. The importance sampling gives results which are better than both of these methods.

The standard deviation as a mesure of the error is according to the plot 2 sometimes connected to a high degree of randomness. For some of the data points the error estimate is far lower than the real value. A better approach to estimate the error should be sought after.

The time to run these programs shows that MCC is the fastest for a given N. This is explained by the operations used. MCC only uses three expensive[1] functions per calculation, MCP uses 9 expensive functions per calculation while MCI uses 4. This should be weighted against the accuracy gained from the different methods, which clearly favours MCI.

The speedup when using OpenMP is not optimal. Although a small speedup is achieved, the optimalisation does not use all the virtual cores optimally.

[1]Exponentions, trigonometric and square root

One reason for this could be the random number generator. Since there is only one RNG for all threads, here could be some idling of the threads when waiting for this to be available. Future implementations should create a thread-local RNG to avoid this idling.