University of Oslo

FYS4150

Helene Aune, Helle Bakke, & Kristine Moseid

Schroedinger's equation for two electrons in a three-dimensional harmonic oscillator well

October 2, 2016

Contents

Abstract

We used the Jacobi method and the C++ library *'armadillo'* to solve Schroedinger's equation for two electrons in both a non-interacting and an interacting case. The interacting case was approached with a repulsive Coulomb force. We found that *'armadillo'* was more time effective compared to the Jacobi method, the ideal number of mesh points was $N = 400$, and the ideal maximum potential was $\rho_{max} = 6$. Figures show that the width of the well narrowed with increasing ω_ρ values.

1 Introduction

The aim of this project was to solve Scroedinger's equation for two electrons in a three-dimensional harmonic oscillator, with and without a repulsive Coloumb interaction. We solved the equation by reformulating it in a discretized form as an eigenvalue equation to be solved by Jacobi's method. By implementing Jacobi's method to a code, we were able to achieve this.

We assumed that the electrons moved in a three-dimensional harmonic oscillator potential and repelled each other via the static Coloumb interaction. We also assumed spherical symmetry.

2 Methods

2.1 Schroedinger's equation for non-interacting electrons

The first thing we were interested in was the radial part of the Schroedinger equation. From there, we were able to reformulate it so that we got an eigenvalue equation:

$$-\frac{\hbar^2}{2m}\Big(\frac{1}{r^2}\frac{d}{dr}r^2 - \frac{l(l+1)}{r^2}\Big)R(r) + V(r)R(r) = ER(r)$$

We substituted $R(r) = (1/r)u(r)$, and set $l = 0$ for this project. We obtained the equation

$$-\frac{\hbar^2}{2m}\frac{d^2}{dr^2}u(r) + V(r)u(r) = Eu(r)$$

Next, we introduced the dimensionless variable $\rho = (1/\alpha)$ and obtained

$$-\frac{\hbar^2}{2m}\frac{d^2}{d\rho^2}u(\rho) + V(\rho)u(\rho) = Eu(\rho)$$

By inserting $V(\rho) = (1/2)k\alpha^2\rho^2$, we had the expression

$$-\frac{\hbar^2}{2m\alpha^2}\frac{d^2}{d\rho^2}u(\rho) + \frac{k}{2}\alpha^2\rho^2 u(\rho) = \frac{2m\alpha^2}{\hbar^2}Eu(\rho)$$

After multiplying with $2m\alpha^2/\hbar^2$ on both sides, and fixing α so that $(mk/\hbar^2)\alpha^4 = 1$, we could rewrite the Schroedinger equation as

$$-\frac{d^2}{d\rho^2}u(\rho) + \rho^2 u(\rho) = \lambda u(\rho),$$

where $\lambda = \frac{2m\alpha^2}{\hbar^2}E$. Using the standard expression for the second derivative of a function u, we had

$$u'' = \frac{u(\rho+h) - 2u(\rho) + u(\rho-h)}{h^2} + O(h^2)$$

We defined the minimum and maximum values for ρ. With given a number of mesh point N, we defined $\rho_{min} = \rho_0$ and $\rho_{max} = \rho_N$. The maximum value of ρ had to be checked, since $\rho_{max} = \inf$ was a value we could not see. The step length was defined as

$$h = \frac{\rho_N - \rho_0}{N}$$

To calculate the next values of ρ_i, we used that

$$\rho_i = \rho_0 + ih, \quad i = 1, 2, ..., N$$

Then the Schroedinger equation for value ρ_i was written as

$$-\frac{u(\rho_i+h) - 2u(\rho_i) + u(\rho_i-h)}{h^2} + \rho_i^2 u(\rho_i) = \lambda u(\rho_i),$$

or in a more compact way

$$-\frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} + \rho_i^2 u_i = \lambda u_i$$
$$-\frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} + V_i u_i = \lambda u_i$$

where $V_i = \rho_i^2$ was the harmonic oscillator potential. The first diagonal matrix element was defined as

$$d_i = \frac{2}{h^2} + V_i,$$

while the non-diagonal matrix elements were defined as

$$e_i = -\frac{1}{h^2}$$

We rewrote the last equation as a matrix eigenvalue problem

$$\begin{bmatrix} d_0 & e_0 & 0 & 0 & \dots & 0 & 0 \\ e_1 & d_1 & e_1 & 0 & \dots & 0 & 0 \\ 0 & e_2 & d_2 & e_2 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & \dots & \dots & \dots e_{N-1} & d_{N-1} & e_{N-1} \\ 0 & \dots & \dots & \dots & \dots & e_N & d_N \end{bmatrix} \begin{bmatrix} u_0 \\ u_1 \\ \dots \\ \dots \\ \dots \\ u_N \end{bmatrix} = \lambda \begin{bmatrix} u_0 \\ u_1 \\ \dots \\ \dots \\ \dots \\ u_N \end{bmatrix}$$

By inserting the values of d_i and e_i, the matrix looked like

$$\begin{bmatrix} \frac{2}{h^2} + V_1 & -\frac{1}{h^2} & 0 & 0 & \dots & 0 & 0 \\ -\frac{1}{h^2} & \frac{2}{h^2} + V_2 & -\frac{1}{h^2} & 0 & \dots & 0 & 0 \\ 0 & -\frac{1}{h^2} & \frac{2}{h^2} + V_3 & -\frac{1}{h^2} & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & \dots & \dots & -\frac{1}{h^2} & \frac{2}{h^2} + V_{N-2} & -\frac{1}{h^2} \\ 0 & \dots & \dots & \dots & \dots & -\frac{1}{h^2} & \frac{2}{h^2} + V_{N-1} \end{bmatrix}$$

We called this matrix $\mathbf{A}$, the non-diagonal matrix elements $\mathbf{a_{kl}}$, and the diagonal matrix elements $\mathbf{a_{ll}}$ and $\mathbf{a_{kk}}$.
By using s, c, and t is short for sin, cos and tan, respectively. We found the largest non-diagonal element and rewrote it as a new element $\mathbf{b_{kl}}$:

$$\mathbf{b_{kl}} = (\mathbf{a_{kk}} - \mathbf{a_{ll}})cs + \mathbf{a_{kl}}(c^2 - s^2) = 0$$

The algorithm used to program the rotation was as follows for i $\neq$ k and i $\neq$ l

$$
\begin{aligned}
\mathbf{b_{ii}} &= \mathbf{a_{ii}} \\
\mathbf{b_{ik}} &= \mathbf{a_{ik}}c - \mathbf{a_{il}}s \\
\mathbf{b_{il}} &= \mathbf{a_{il}}c + \mathbf{a_{ik}}s \\
\mathbf{b_{kk}} &= \mathbf{a_{kk}}c^2 - 2\mathbf{a_{kl}}cs + \mathbf{a_{ll}}s^2 \\
\mathbf{b_{ll}} &= \mathbf{a_{ll}}c^2 - 2\mathbf{a_{kl}}cs + \mathbf{a_{kk}}s^2 \\
\mathbf{b_{kl}} &= 0
\end{aligned}
$$

This was repeated until $\text{Off}(\mathbf{A})^{\mathbf{2}} \leq \epsilon \sim \mathbf{10^{-8}}$.

2.2 Preservation of orthogonality and dot product

We had a basis $\mathbf{v}_i$ that we assumed was orthogonal by

$$\mathbf{v}_j^T \mathbf{v}_i = \delta_{ij}$$

We then had a unitary transformation of $\mathbf{v}_i$ expressed like $\mathbf{w}_i = \mathbf{U}\mathbf{v}_i$, this would also preserve both the dot product and the orthogonality by

$$\mathbf{w}_j^T \mathbf{w}_i = (\mathbf{U}\mathbf{v}_j)^T \mathbf{U}\mathbf{v}_i = \mathbf{v}_j^T \mathbf{U}^T \mathbf{U}\mathbf{v}_i = I\mathbf{v}_j^T \mathbf{v}_i = \delta_{ij}$$

2.3 Unit tests

We implemented two unit tests to our program to test some mathematical properties of our algorithm. The first test made sure that the eigenvalues of $\mathbf{A}$ were correct. We used *'armadillo'* to calculate the eigenvalues quickly, then we compared the two. We added a tolerance to the comparison, since each number was unnoticably different.

The second test made sure that the orthogonality and dot product were preserved. By following the example above, we created an identity matrix of the, hopefully, orthogonal elements. Then, we used *'armadillo'* to create an exact identity matrix. We subtracted the two and compared them. Again, we used a tolerance since the elements of the identity matrix was not exact equal to 1 and 0.

2.4 Schroedinger's equation with interacting electrons

When solving Schroedinger's equation in subsection 2.1, we based our calculations on two non-interacting electrons. We then tried solving the Schroedinger equation where two electrons interact through a repulsive Coulomb interaction. This meant that we introduced a relative coordinat $\mathbf{r} = \mathbf{r}_1 - \mathbf{r}_2$, which gave

$$\left(-\frac{\hbar^2}{m}\frac{d^2}{dr^2} - \frac{\hbar^2}{4m}\frac{d^2}{dR^2} + \frac{1}{4}kr^2 + kR^2\right)u(r,R) = E^{(2)}u(r,R).$$

The original equations, shown in Section 2.1, could be rewritten[3] as

$$-\frac{d^2}{d\rho^2}\psi(\rho) + \omega_\rho^2\rho^2\psi(\rho) + \frac{1}{\rho} = \lambda\psi(\rho).$$

Where we treated ω_ρ as a parameter which reflected the strength of the oscillator potential.

3 Results

3.1 Jacobi's algorithm

The number of transformations needed for the $\text{Off}(\mathbf{A})^{\mathbf{2}} \leq \epsilon \sim \mathbf{10^{-8}}$, turned out to be 267515. The number of meshpoints needed to get the lowest three eigenvalues with four digit precision using the Jacobi algorithm, was measured to be 400. We chose $\rho_{max} = 6$ and got 2.9999, 6.9996, 10.9991 as the lowest three eigenvalues.

Table 1 shows the time spent using our Jacobi algorithm compared to the C++ *'armadillo'* package function *'eig_sym'*. It was clear from this that the *'armadillo'* function was a much better tool than the Jacobi algorithm for problems like this one. One of the reasons for this may be the incredible amount of iterations needed for each computation with Jacobi's method.

Matrix dimension	Jacobi's algorithm	*'armadillo'*
400	74.2503 s	0.05067 s

Table 1: Time comparison using *'armadillo'*s function *'eig_sym'* and Jacobi's algorithm.

N	i
50	3955
100	16238
150	36916
200	65899
250	103676
300	149636
350	203901
400	267515

Table 2: Dimensions of the matrix (N) compared with the number of transformations (i).

3.2 Interacting and non-interacting electrons

The following four figures show the probability distribution of electrons in a harmonic oscillator well with varying ω_ρ. From Figure 1 we saw that there was a noticable difference between the functions. This was explained by the range of the oscillator potential, where the electrons were free to move between all values of ρ. In the other three figures the electrons moved in a shorter range. The width of the wells were smaller, and therefore the potential for each case was similar. That was why there was no big difference between the functions.

We noted how the width of the well changed with increasing ω_ρ values. For bigger ω_ρ the width became smaller, and so did the range.

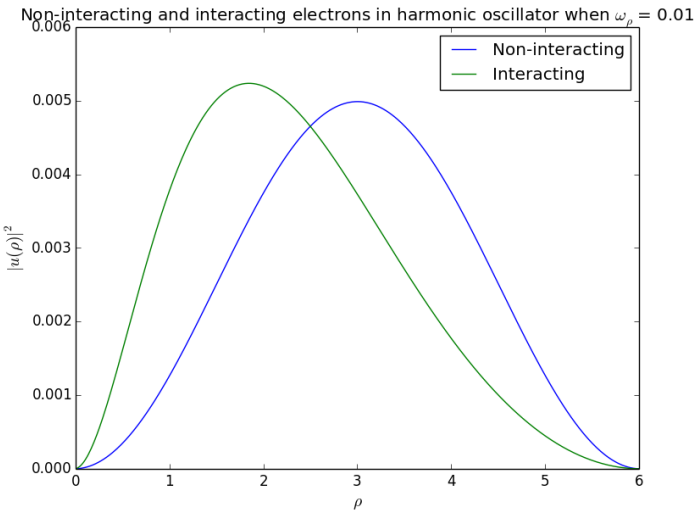

Figure 1: Probability distribution of electrons in a harmonic oscillator well with $\omega_\rho = 0.01$

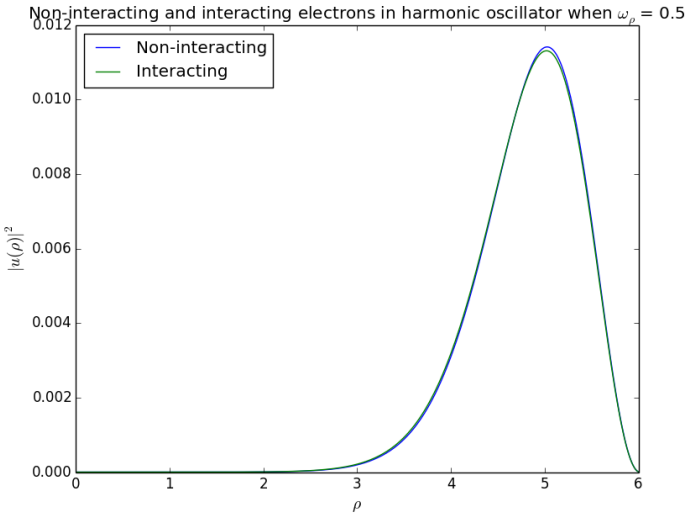

Figure 2: Probability distribution of electrons in a harmonic oscillator well with $\omega_\rho = 0.5$

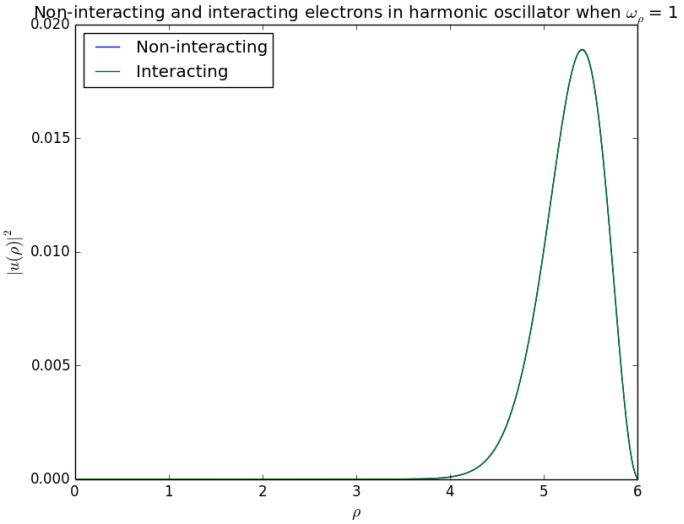

Figure 3: Probability distribution of electrons in a harmonic oscillator well with $\omega_\rho = 1.0$

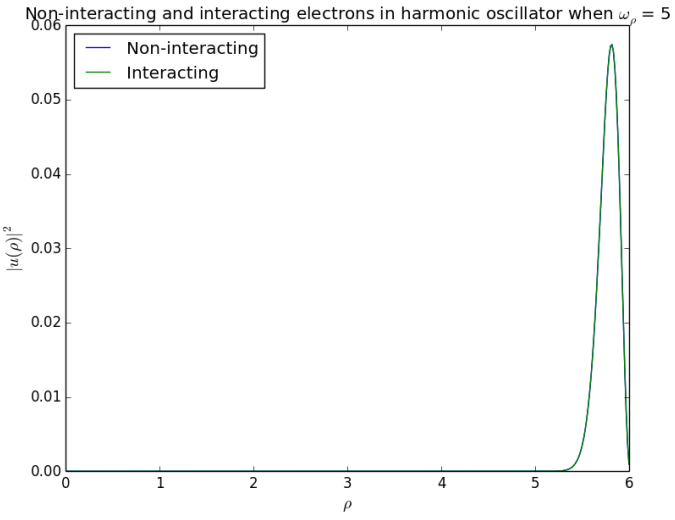

Figure 4: Probability distribution of electrons in a harmonic oscillator well with $\omega_\rho = 5.0$

4 Conclusion

In this project we experienced that *'armadillo'* was a much more efficient tool to solve the Schroedinger's equation compared to Jacobi's method. By implementing unit tests we verified that our solver preserved the dot product and orthogonality. We also made sure that the eigenvalues were correct after similarity transformations. After solving Jacobi's method for the interacting and non-interacting case, we found that for larger ω_ρ values the range of the well became smaller.

As an improvement, we can work on our debugging skills so that the programming process becomes easier. Our group consists of students from different departments, and we should work on sharing our knowledge in different subjects.

5 Appendix

The program *'main.cpp'* takes two input arguments. The first argument is the preferred ω_ρ value, while the second argument varies between *1*, *2* and *3*. *1* represents the potential for one electron, *2* represents the potential for two electrons that don't interact, and *3* represents two electrons that interact.

Be sure to have all files downloaded when running *'main.cpp'*, as it includes a few self-made header files.

All files and benchmarks can be found at the GitHub address: `https://github.com/hellmb/CompPhys/tree/master/Project%202%20-%20GitHub`

References

[1] Lecture notes, FYS3150 morten hjorth jensen. `https://github.com/CompPhysics/ComputationalPhysics/blob/master/doc/Lectures/lectures2015.pdf`. Accessed: 2016-09-16.

[2] Project 2 morten hjorth jensen. `https://github.com/CompPhysics/ComputationalPhysics/blob/master/doc/Projects/2016/Project2/project2_2016.pdf`. Accessed: 2016-09-30.

[3] Unit testing wikipedia. https://en.wikipedia.org/wiki/Unit_testing. Accessed: 2016-09-30.