FYS3150 Computational Physics – Project 2

Harald Moholt

October, 2016

Abstract

This project solves Schrödinger's equation for two electrons in a three-dimensional harmonic oscillator well using Jacobi's method for solving a set of linear equation in order to find the eigenvalues and eigenvectors. Standardized functions from the Armadillo library were used for faster computation. Both non-interacting case and interacting case were studied and wave functions were plotted for the different scenarios. After trial and error finding usable values for the different cases, a feasible result was achieved.

GitHub repository with code can be found at: github.com/harmoh/FYS3150_Project_2

1 Introduction

The aim of this project is to solve Schrödinger's equation for two electrons in a three-dimensional harmonic oscillator well. In order to do this, Jacobi's method is implemented to find eigenvalues and eigenvectors. The three lowest eigenvalues and eigenvectors are found and compared to given eigenvalues of $\lambda_0 = 3$, $\lambda_1 = 7$ and $\lambda_2 = 11$. Standard functions from the Armadillo library for solving eigenvalue problems are also used for reduced computation speed. Both non-interacting case and interacting case are studied, where the interacting case uses a change in potential to achieve a simulation of the difference. The interacting case focuses on four different frequencies. For both cases, wave functions are plotted.

2 Theory

We are interested in the solution of the radial part of Schrödinger's equation for a single electron:

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

where $V(r) = m\omega^2 r^2/2$ is the harmonic oscillator potential in our case. We can substitute $R(r) = (1/r)u(r)$, $\rho = (1/\alpha)r$ (where α is a constant) and use boundary conditions $u(0) = 0$ and $u(\infty) = 0$. With this, we get:

$$-\frac{\hbar^2}{2m\alpha^2}\frac{d^2}{d\rho^2}u(\rho)+\left(V(\rho)+\frac{l(l+1)}{\rho^2}\frac{\hbar^2}{2m\alpha^2}\right)u(\rho)=Eu(\rho) \tag{2}$$

In this project we set $l = 0$ and can insert $V(\rho) = (1/2)k\alpha^2\rho^2$. Then we can multiply with $2m\alpha^2/\hbar^2$ on both sides:

$$-\frac{d^2}{d\rho^2}u(\rho) + \frac{mk}{\hbar^2}\alpha^4\rho^2 u(\rho) = \frac{2m\alpha^2}{\hbar^2}Eu(\rho) \tag{3}$$

Since α is a constant, we can set

$$mk\alpha^4/\hbar^2 = 1 \qquad \text{and} \qquad \alpha = \left(\frac{\hbar^2}{mk}\right)^{1/4} \tag{4}$$

and define:

$$\lambda = \frac{2m\alpha^2}{\hbar^2}E \tag{5}$$

From Schrödinger's equation we then end up with a new equation:

$$-\frac{d^2}{d\rho^2}u(\rho) + \rho^2 u(\rho) = \lambda u(\rho) \tag{6}$$

By solving this equation numerically, we can set up the standard expression for the second derivative of the function u:

$$u'' = \frac{u(\rho+h) - 2u(\rho) + u(\rho-h)}{h^2} + O(h^2) \tag{7}$$

where the step length is defined as

$$h = \frac{\rho_n - \rho_0}{n} \qquad \text{and} \qquad \rho_i = \rho_0 + ih \qquad \text{for} \qquad i \in \{1, 2, \ldots, n\} \tag{8}$$

where n is the size of the square matrix $\mathbf{A}$. The three first eigenvalues are $\lambda_0 = 3$, $\lambda_1 = 7$ and $\lambda_2 = 11$. We also have to define the values $\rho_{min} = \rho_0 = 0$ and $\rho_n = \rho_{max}$, where ρ_{max} is set manually later. In this project, we use a tridiagonal matrix where all the non-diagonal elements are equal and given by:

$$e_i = -\frac{1}{h^2} \tag{9}$$

The diagonal elements are defined as:

$$d_i = \frac{2}{h^2} + V_i \tag{10}$$

where the the harmonic oscillator potential is $V_i = \rho_i^2$. By implementing this into the expression in eq. (7), we get:

$$d_i u_i + e_{i-1}u_{i-1} + e_{i+1}u_{i+1} = \lambda u_i \tag{11}$$

In this, u_i is unknown. If we rewrite the numerical expression into a matrix eigenvalue problem, we have $\mathbf{Au} = \lambda\mathbf{u}$, where $\mathbf{A}$ is an $n \times n$ tridiagonal matrix:

$$\begin{bmatrix} d_1 & e_1 & 0 & \ldots & \ldots & 0 \\ e_1 & d_2 & e_2 & & & 0 \\ 0 & e_2 & d_3 & & & 0 \\ \vdots & & & \ddots & & \vdots \\ 0 & & & & d_{n-1} & e_{n-1} \\ 0 & 0 & \ldots & \ldots & e_{n-1} & d_n \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_{n-1} \\ u_n \end{bmatrix} = \lambda \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_{n-1} \\ u_n \end{bmatrix} \tag{12}$$

3 Method

Orthogonal matrices are essential in Jacobi transformations. These matrices preserves the dot product and the orthogonality of the matrices. Let $\mathbf{S}$ be a 2×2 matrix:

$$\mathbf{S} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \qquad \text{and} \qquad \mathbf{u} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \qquad \text{and} \qquad \mathbf{v} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{13}$$

where $c = \cos(\theta)$, $s = \sin(\theta)$ and $\mathbf{S}^{\mathrm{T}}\mathbf{S} = \mathbf{I}$. In this case, $\mathbf{u}$ and $\mathbf{v}$ are orthogonal since $\mathbf{u} \cdot \mathbf{v} = 0$. We can check that a unitary transformation preserves the orthogonality by:

$$\mathbf{w} = \mathbf{S}\mathbf{u} = \begin{bmatrix} c \\ -s \end{bmatrix} \qquad \text{and} \qquad \mathbf{w}^{\mathrm{T}}\mathbf{w} = s^2 + c^2 = 1 \tag{14}$$

Two different methods are used for calculating a numerical solution for the eigenvalues. The Jacobi method is implemented for a brute force approach to find the eigenvalues and eigenvectors for any given matrix. The essential of the Jacobi method is to perform a series of transformations in order to set non-diagonal elements to zero while preserving the eigenvalues and eigenvectors. First, the largest off-diagonal values in the matrix is set to zero (or close to). Then we solve

$$(a_{kk} - a_{ll})c \cdot s + a_{kl}(c^2 - s^2) = 0 \tag{15}$$

with respect to a_{kk} and a_{ll}, where $c = \cos\theta$ and $s = \sin\theta$. We create τ:

$$\tau = \frac{a_{ll} - a_{kk}}{2a_{kl}} \tag{16}$$

By setting $t = \tan\theta$, we can write:

$$t^2 + 2\tau t - 1 = 0 \tag{17}$$

$$c = \frac{1}{\sqrt{1 + t^2}} \qquad \text{and} \qquad s = c \cdot t \tag{18}$$

This gives us:

$$t = \frac{1}{\tau + \sqrt{1 + \tau^2}} \tag{19}$$

The first step of the algorithm check that the off-diagonal elements are less than a tolerance value. Then the rotation can be performed and is shown in the snipped code below. Local variables are declared for reuse in order to achieve lower computation time. Lastly, eigenvectors are put in a separate matrix, $\mathbf{R}$.

```
1     // Declare and initialize local variables
2     double a_kk, a_ll, a_ik, a_il, r_ik, r_il;
3     a_kk = A(k,k);
4     a_ll = A(l,l);
5
6     // Changing the matrix elements with indices k and l
7     A(k,k) = a_kk * c * c - 2 * A(k,l) * c * s + a_ll * s * s;
8     A(l,l) = a_ll * c * c + 2 * A(k,l) * c * s + a_kk * s * s;
9
10    // Manually set elements to zero
11    A(k,l) = 0;
12    A(l,k) = 0;
13
14    // Perform rotation
15    for(int i = 0; i < n; i++)
```

```
16      {
17          if(i != k && i != l)
18          {
19              a_ik = A(i,k);
20              a_il = A(i,l);
21              A(i,k) = a_ik * c - a_il * s;
22              A(k,i) = A(i,k);
23              A(i,l) = a_il * c + a_ik * s;
24              A(l,i) = A(i,l);
25          }
26          r_ik = R(i,k);
27          r_il = R(i,l);
28          R(i,k) = r_ik * c - r_il * s;
29          R(i,l) = r_il * c + r_ik * s;
30      }
```

Finding eigenvalues and eigenvectors using the Jacobi method is compared to the function, `eig_sym()`, from the Armadillo library. This function was used later in the program due to the significantly lower computation time compared to the Jacobi method.

This project uses both a non-interacting case and an interacting case where the latter involves repulsive Coloumb interaction. For the interacting case, we define a "frequency", ω_r, when initializing the matrix $\mathbf{A}$. This way we change the potential from $V_i = \rho^2$ to:

$$V_i = \omega_r^2 \rho^2 + 1/\rho \tag{20}$$

The eigenvalues and eigenvectors are found the same way for the interacting case as the non-interacting case.

4 Implementation

The whole project code can be run from the python script `Run_main.py`. This runs `main.cpp` and plot the results from the computed text files. The function `jacobi_method()` runs the entire code for performing the Jacobi rotation which is implemented in the file `jacobi.cpp`. This function also produce the file `Results_jacobi.txt` with eigenvalues and computation time for Jacobi rotation compared to method from the Armadillo library.

The files `non_interacting.cpp` and `interacting.cpp` finds the eigenvalues and eigenvectors using the Armadillo library for the non-interacting case and the interacting case, respectively. They also produce files with the results, which are plotted in the python script. The file `unit_test.cpp` includes the function `test_max_off()`, which test the function `max_offdiagonal()` in `jacobi.cpp` with a 5×5 tridiagonal matrix with random values to check if it returns the maximum off-diagonal value.

5 Result

The Jacobi method was implemented and run with n-values ranging from 50 to 350, where higher n-values were avoided due to lack of patience. The eigenvalues λ_0, λ_1 and λ_2 are 3, 7 and 11, respectively. It can be seen in table 1 that calculated eigenvalues approximates these values

n	Eigenvalues	Time (Jacobi)	Time (Armadillo)	Transformations
50	{2.99687, 6.98434, 10.9619}	0.018096 s	0.0008600 s	3852
100	{2.99922, 6.99609, 10.9907}	0.294091 s	0.0061280 s	16217
150	{2.99965, 6.99827, 10.9960}	1.637430 s	0.0126660 s	36847
200	{2.99980, 6.99903, 10.9978}	5.852860 s	0.0158400 s	66065
250	{2.99988, 6.99938, 10.9987}	14.370100 s	0.0242680 s	193791
300	{2.99991, 6.99957, 10.9991}	30.812800 s	0.0436800 s	149879
350	{2.99994, 6.99968, 10.9994}	84.516500 s	0.0577560 s	204649

Table 1: Eigenvalues computed with Jacobi

with increasing n. The eigenvalues shown are computed using the Jacobi method, but showed no difference when compared to the results using the Armadillo library. Computation time was measure for each method. The Jacobi method also uses significantly more time than using the Armadillo library function for finding eigenvalues and eigenvectors. With $n = 350$ the Jacobi method uses more the 80 seconds while the function from the Armadillo library uses a little more than 50 milliseconds. The Jacobi method uses unnecessary long time for matrices larger than $n = 10^2$.

Figure 1 shows the plotted result for the non-interacting case with $n = 350$ and $\rho_{max} = 5$. This shows the wave functions corresponding to the eigenvalues in table 1.

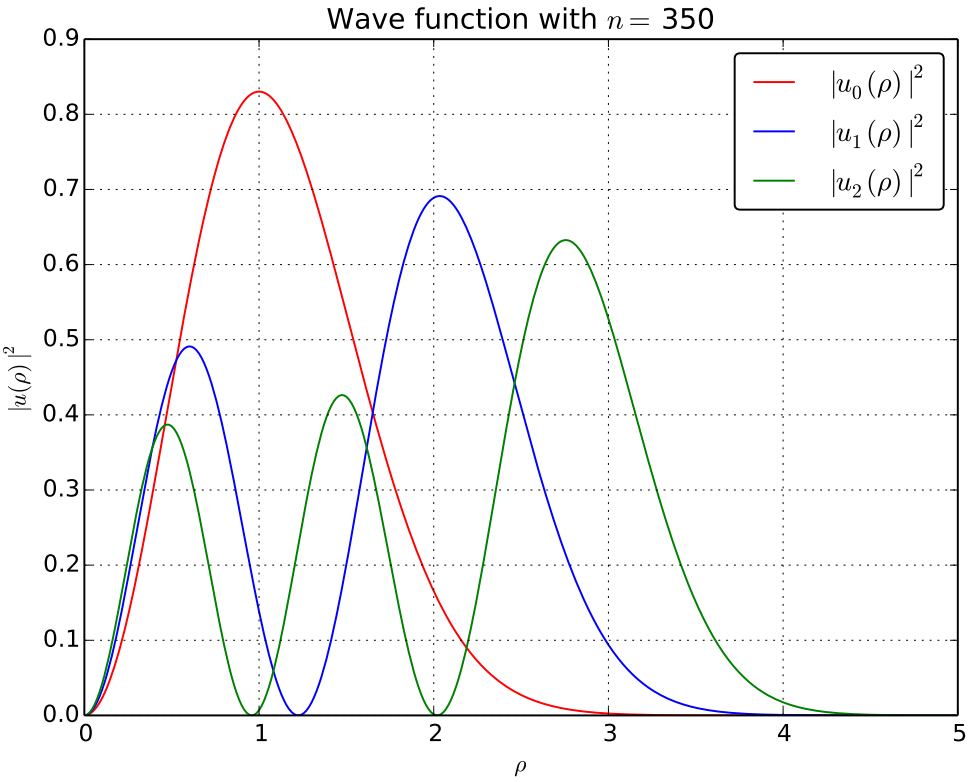

Figure 1: Non-interacting case

The interacting case used mostly the same code as for the non-interacting case. The only difference was a slight difference to the potential in the diagonal of the matrix **A**. A "frequency", ω_r, was added and set to the values 0.01, 0.5, 1 and 5. All the different ω_r used $n = 350$ in order to compare with the non-interacting case. ρ_{max} was set to different values for different ω_r in order to include the whole wave function. The values are shown below. These values can be seen used

ω_r	0.01	0.5	1	5
ρ_{max}	50	7	5	2

in figs. 2 to 5 as the wave functions vary depending on the chosen ω_r. Table 2 shows the ground state eigenvalues as a function of different ω_r with $n = 350$ for all frequencies.

Frequency, ω_r	Ground state eigenvalues
0.01	0.105774
0.50	2.230079
1.00	4.057810
5.00	17.448425

Table 2: Ground state eigenvalues as a function of varying strengths of ω_r

Figures 2 to 5 shows the wave function with different ω_r for the three lowest lying states. We can observe that $|u_o(\rho)|^2$ shows one top, $|u_1(\rho)|^2$ shows two tops and $|u_2(\rho)|^2$ shows three tops.

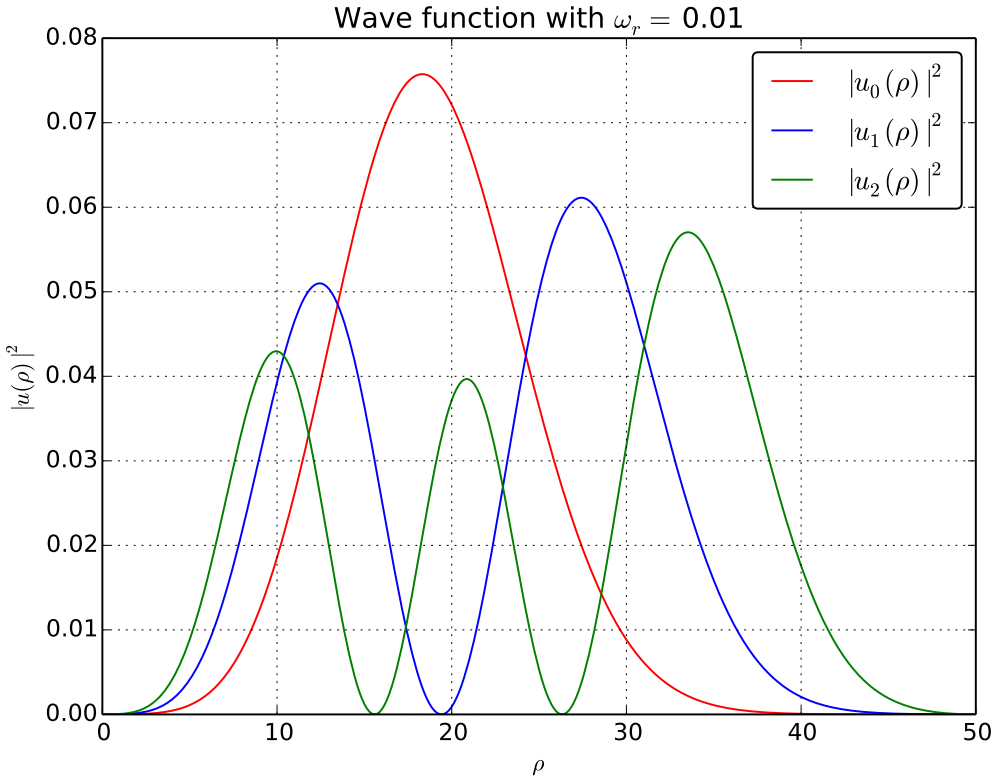

Figure 2: Interacting case with $\omega_r = 0.01$

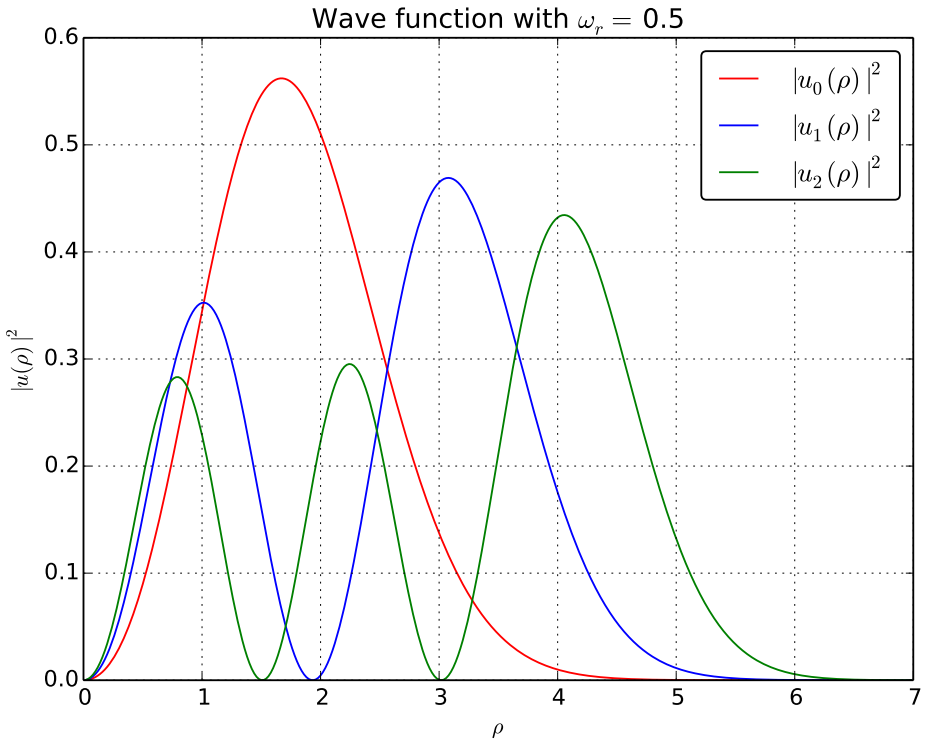

Figure 3: Interacting case with $\omega_r = 0.5$

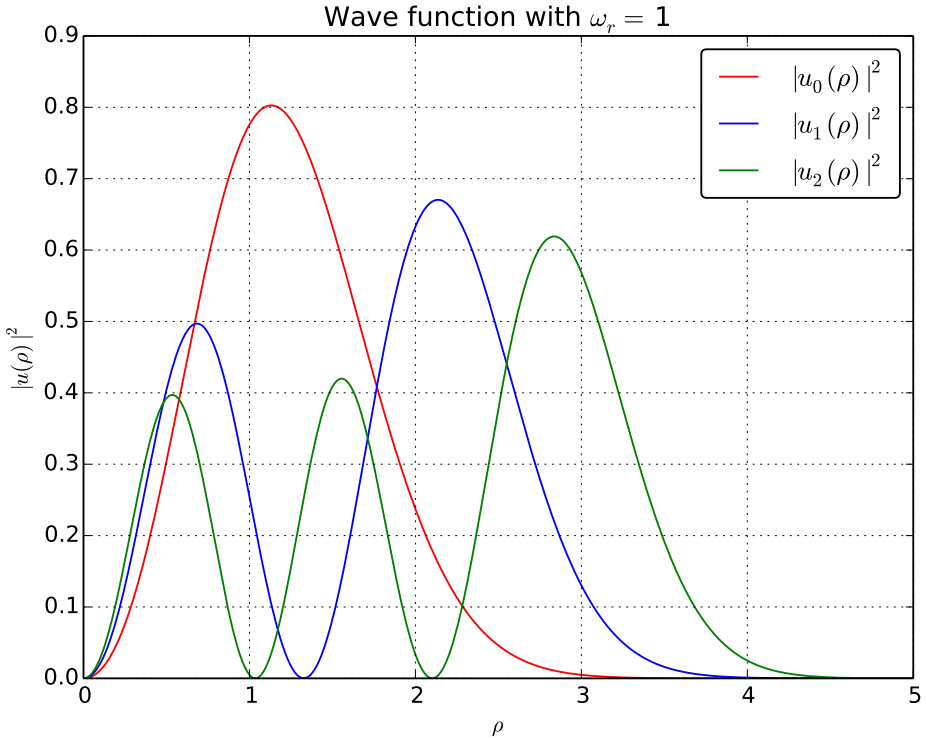

Figure 4: Interacting case with $\omega_r = 1$

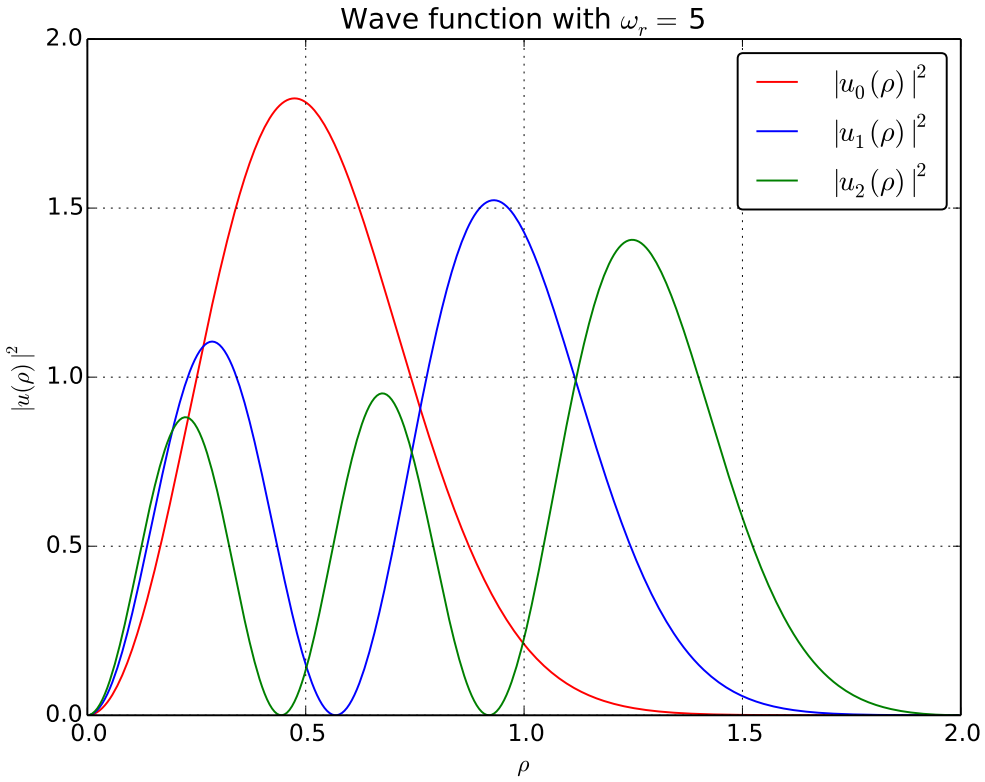

Figure 5: Interacting case with $\omega_r = 5$

The functions also shows higher values with higher ω_r. The wave functions are also closer to each other when at higher ω_r. This becomes clear when comparing ρ_{max}, where $\rho_{max} = 2$ for $\omega_r = 5$ and $\rho_{max} = 50$ for $\omega_r = 0.01$.

6 Discussion

As seen for the Jacobi rotation algorithm, it was possible to implement the method and achieve correct eigenvalues for the lowest lying states, λ_0, λ_1 and λ_2. However, this method used a huge amount of computation time that could easily be avoided by implementing the standardized function `eig_sym()` from the Armadillo library. If the matrix $\mathbf{A}$ had not been a tridiagonal matrix, the Jacobi method might have taken even more time. Since the Jacobi method goes through every element of the matrix, and we only use a tridiagonal where most of the elements are zero, the algorithm could have been simplified and tailored to fit our case. This could have reduced the computation time significantly. To go further, it is possible to reduce the tridiagonal matrix to a matrix with only diagonal elements using Householders method in order to decrease the number of FLOPS even more.

7 Conclusion

In this project we have studied Schrödinger's equation for two electrons in a three-dimensional harmonic oscillator. Jacobi's method was implemented to find the eigenvalues and eigenvectors for the three lowest lying states for a tridiagonal matrix **A**. Since the Jacobi method used a huge amount of computation time, a standardized solver in the Armadillo library were used further to find eigenvalues and eigenvectors for the non-interacting case and the interacting case. After trial and error finding usable ρ_{max} and n values for the different cases, a feasible result was achieved.