UNIVERSITY OF OSLO

COMPUTATIONAL PHYSICS

Project 2

UiO : University of Oslo

Authors:

Birgitte Madsen

Magnus Isaksen

Soumya Chalakkal

Autumn 2015

UiO : University of Oslo

Department of Physics
University of Oslo
Sem Sælands vei 24
0371 Oslo, Norway
+47 22 85 64 28
http://www.mn.uio.no/fysikk/english/

Course:

Computational Physics

Project number:

2

Link to GitHub folder:

`https://??`

Hand-in deadline:

Monday, October 5, 2015

Project Members:

Birgitte Madsen
Magnus Isaksen
Soumya Chalakkal

Copies: 1
Page count: ??
Appendices: 0
Completed: ??

TABLE OF CONTENTS

CHAPTER

1 INTRODUCTION

The aim of this project is to solve Schrödinger's equation numerically, using Jacobi's method, for a single electron with zero angular momentum in a harmonic oscillator potential, as well as for two electrons in a three dimensional harmonic oscillator well, both with and without Coulomb interaction. To solve Schrödinger's equation using Jacobi's method, it is first reformulated into an eigenvalue problem.

The Jacobi algorithm is implemented in c++, and the source codes developed in this project and selected results, can be found in the GitHub repository: `https://?` . [1]

In order to insure the credibility of the algorithm, various tests are run. Amongst these tests are solving the considered eigenvalue problem for a simple 2×2 case to check the correctness of the computed solution to the known values, and a comparison of the computed eigenvalues to the analytical solution.

The characteristics of the algorithm are, furthermore, examined by finding the optimal number of steps and interval that gives the best value of the lowest three eigenvalues for the single particle case. As a part of this characterization, the influence of number of steps on the number of iterations in the Jacobi method is investigated, and the consequence of changing the size of the considered interval and the number of steps both independently and dependently of each other is discussed. Furthermore, the computed Jacobi algorithm is compared to the precomputed Armadillo function for solving eigenvalue problems.

The report mainly consists of two sections. First section discusses the nature of the problem, the functionality of the algorithm and also the various tests on the algorithm. The last section is about the results, its interpretation and discussions.

[1] FiXme Note: correct the these lines

CHAPTER

2 METHOD

We present in this chapter a short discussion on the nature of the problem.

The source code itself can be found in the GitHub folder `https://??`. [1]

2.1 Nature of the problem

The aim of the first part of the project is to solving Schrödingers equations for one electron in a harmonic oscillator potential with angular momentum $l = 0$. The radial part of the Schrödingers equation is considered which is as follows

$$\left[-\frac{\hbar^2}{2m}\frac{1}{r^2}\frac{d}{dr}r^2\frac{d}{dr}+V(r)\right]R(r) = ER(r). \tag{2.1}$$

In order to solve this equation numerically, it is rewritten after a series of transformation and substitution as

$$-\frac{d^2}{d\rho^2}u(\rho)+\rho^2u(\rho) = \lambda u(\rho) \tag{2.2}$$

Eq. (2.2) is discretized by writing the second derivative of $u(\rho)$ as

$$\frac{d^2}{d\rho^2}u(\rho) = \frac{u(\rho+h)-2u(\rho)+u(\rho-h)}{h^2}+O(h^2) \tag{2.3}$$

In Eq. (2.3) h is the step length, and ρ_{max} and ρ_{min} are the maximum and minimum values of the variable ρ, respectively. For a given number of steps n, the step length is given as

$$h = \frac{\rho_{max}-\rho_{min}}{n} \tag{2.4}$$

In order to solve equation Eq. (2.2), it is transformed into a matrix eigenvalue problem

$$\mathbf{Au} = \lambda\mathbf{u} \tag{2.5}$$

[1] FiXme Note: correct the above lines

in which **A** is a tridiagonal matrix of the form

$$\mathbf{A} = \begin{pmatrix} \frac{2}{h^2}+V_1 & -\frac{1}{h^2} & 0 & 0 & \dots & 0 & 0 \\ -\frac{1}{h^2} & \frac{2}{h^2}+V_2 & -\frac{1}{h^2} & 0 & \dots & 0 & 0 \\ 0 & -\frac{1}{h^2} & \frac{2}{h^2}+V_3 & -\frac{1}{h^2} & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & \dots & \dots & \dots & \frac{2}{h^2}+V_{n-2} & -\frac{1}{h^2} \\ 0 & \dots & \dots & \dots & \dots & -\frac{1}{h^2} & \frac{2}{h^2}+V_{n-1} \end{pmatrix} \tag{2.6}$$

A is obtained from Eq. (2.2), with the approximation of the derivative of $u(\rho)$ given in Eq. (2.3) when omitting all later terms, by discretizing ρ by

$$\rho_i = \rho_{min} + ih \qquad i = 0,1,2,\dots,n \tag{2.7}$$

This leads to the following Schrödinger equation:

$$-\frac{u(\rho_i+h)-2u(\rho_i)+u(\rho_i-h)}{h^2} + \rho_i^2 u(\rho_i) = \lambda u(\rho_i) \tag{2.8}$$

which can be rewritten as

$$-\frac{u_{i+1}-2u_i+u_{i-1}}{h^2} + \rho_i^2 u_i = -\frac{u_{i+1}-2u_i+u_{i-1}}{h^2} + V_i u_i = \lambda u_i \tag{2.9}$$

in which $V_i = \rho_i^2$ is the harmonic oscillator potential. When comparing this relation with the general eigenvalue problem in Eq. (2.5), it is evident that the diagonal elements of the matrix **A** is given by

$$d_i = \frac{2}{h^2} + V_i \tag{2.10}$$

while all off diagonal elements are zero apart from those neighbouring the diagonal, which are all constants with the value

$$e_i = -\frac{1}{h^2} \tag{2.11}$$

This is exactly what is given in Eq. (2.6).

2.1.1 Change in Matrix Elements after Iterations and Choice of θ

The algorithm for solving the eigenvalue problem given in [2] contains of multiple similarity transformations of the matrix **A**, in which we assume a_{kl} to be the largest off-diagonal element. The matrix **B** constructed by the similarity transformation is given by

$$\mathbf{B} = \mathbf{S}^T \mathbf{A} \mathbf{S} \tag{2.12}$$

in which **S** is an orthogonal transformation matrix with its non-zero matrix elements:

$$s_{kk} = s_{ll} = \cos\theta$$
$$s_{kl} = -s_{lk} = -\sin\theta$$
$$s_{ii} = 1, \qquad i \neq k, i \neq l$$

[2] FiXme Note: eqref

After matrix multiplication with the orthogonal transformation matrix **S** and its transverse (as in (2.12)) the entrances of **B** becomes

$$
\begin{aligned}
b_{ii} &= a_{ii}, \qquad i \neq k, i \neq l \\
b_{ik} &= a_{ik}\cos\theta - a_{il}\sin\theta, \qquad i \neq k, i \neq l \\
b_{il} &= a_{il}\cos\theta + a_{ik}\sin\theta, \qquad i \neq k, i \neq l \\
b_{kk} &= a_{kk}\cos^2\theta - 2a_{kl}\cos\theta\sin\theta + a_{ll}\sin^2\theta \\
b_{ll} &= a_{ll}\cos^2\theta + 2a_{kl}\cos\theta\sin\theta + a_{kk}\sin^2\theta \\
b_{kl} &= (a_{kk} - a_{ll})\cos\theta\sin\theta + a_{kl}(\cos^2\theta - \sin^2\theta)
\end{aligned}
$$

Due to the symmetry in (2.12) with **A** being a tridiagonal symmetric matrix, $b_{lk} = b_{kl}$, $b_{ki} = b_{ik}$, and $b_{li} = b_{il}$.

Since θ can be chosen arbitrarily, we choose θ to be the angle at which b_{kl}, and hence b_{lk}, becomes zero. In this way, the largest element of **A** is eliminated, and it can be shown that this choice of θ reduces the norm of the off-diagonal elements of **A**, which ensures that the algorithm terminates towards the eigenvalues. [3]

This yields the equation

$$0 = (a_{kk} - a_{ll})\cos\theta\sin\theta + a_{kl}(\cos^2\theta - \sin^2\theta) \tag{2.13}$$

By introducing $\tan\theta = \sin\theta/\cos\theta$ and the quantity

$$\tau = \frac{a_{ll} - a_{kk}}{2a_{kl}} \tag{2.14}$$

(2.13) can be rewritten as the quadratic equation in $\tan\theta$

$$\tan^2\theta + 2\tau\tan\theta - 1 = 0 \tag{2.15}$$

which has the solutions

$$\tan\theta = -\tau \pm \sqrt{1+\tau^2} \tag{2.16}$$

From the solutions for $\tan\theta$ given in (2.16), $\cos\theta$ and $\sin\theta$ can be found using the formulas

$$\cos\theta = \frac{1}{\sqrt{1+\tan^2\theta}} \qquad \text{and} \qquad \sin\theta = \tan\theta\cos\theta$$

If $\tau < 0$, $\tan\theta$ is chosen to be

$$\tan\theta = -\tau - \sqrt{1+\tau^2} \tag{2.17}$$

whilst if $\tau \geq 0$, $\tan\theta$ is calculated as

$$\tan\theta = -\tau + \sqrt{1+\tau^2} \tag{2.18}$$

This choice is made to always make $\tan\theta$ the smaller of the two roots given in (2.16). Furthermore, this choice ensures that $|\tan\theta| \leq 1$, yielding that $|\theta| \leq \pi/4$.

[3] FiXme Note: this, I can write, right??

This is true since $|\tau| \leq 1$, because $|a_{kl}| \geq |a_{ij}|$ for all i, j, from which it follows that

$$|\tan\theta| = \left|-\tau - \sqrt{1+\tau^2}\right| \leq 1, \qquad \text{for } \tau < 0 \tag{2.19}$$

and

$$|\tan\theta| = \left|-\tau + \sqrt{1+\tau^2}\right| \leq 1, \qquad \text{for } \tau \geq 0 \tag{2.20}$$

since $\sqrt{1+\tau^2} \leq \sqrt{2}$.

The fact that $|\theta| \leq \pi/4$ ensures that $\cos\theta \geq 0$ which ultimately ensures that the difference between $\mathbf{A}$ and the new matrix $\mathbf{B}$ is minimized, since

$$||\mathbf{B}-\mathbf{A}||_F^2 = 4(1-\cos\theta)\sum_{i=1, i\neq k,l}^{n}(a_{ik}^2 + a_{il}^2) + \frac{2a_{kl}^2}{\cos^2\theta}. \tag{2.21}$$

When implementing the elements of $\mathbf{B}$ in the algorithm, the following source code is used:

```
// Computing the new matrix A
       for (int i = 0; i<n; i++)
       {
           if (i != row_number && i != column_number) // determining A(i,k) for new
               matrix
           {
               a_ik = A(i,row_number);
               a_il = A(i,column_number);
               A(i,row_number) = a_ik*c - a_il*s;
               A(i,column_number) = a_il*c + a_ik*s;
               A(row_number,i) = A(i,row_number);
               A(column_number,i) = A(i,column_number);
           }
       }
       A(row_number, row_number) = a_kk*pow(c,2) - 2*a_kl*c*s+a_ll*pow(s,2);
       A(column_number, column_number) = a_ll*pow(c,2) + 2*a_kl*c*s+a_kk*pow(s,2);
       A(column_number, row_number) = 0.00; //By choice of theta
       A(row_number, column_number) = 0.00; //By choice of theta
```

In the above source code lines *a_ij* is the ij'th element of matrix $\mathbf{A}$, whilst *A(i,j)* is the ij'th element of matrix $\mathbf{B} = \mathbf{S}^T\mathbf{A}\mathbf{S}$. *row_number* and *column_number* are the row number and column number of the element of $\mathbf{A}$ with the maximum absolute value. As seen in the source code, the elements $b_{kl} = b_{lk}$ with the largest absolute value are forced to zero by the choice of θ to save computational time.

2.1.2 Off-diagonal norm

Similarity transformations is used to reduce the off-diagonal norm. The norm found by Eq. (2.22), is wanted as small as possible and smaller than a given test value ε. Ideally the norm should get to zero, but that is difficult because when the elements gets small there can be problems with round-off errors.

The value ε is therefore set so that it gives the smallest values possible without problems round-off errors, typically set around 10^{-8}.

$$off(\mathbf{A}) = \sqrt{\sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n} a_{ij}^2} \tag{2.22}$$

This norm is compared after each transformation so that the new matrix **B** is as close to a diagonal matrix as possible. But this is a very time consuming approach and requires many calculations. So instead Eq. (2.23) is used as it is less time consuming.

$$max(a_{kl}^2) > \varepsilon \tag{2.23}$$

This is possible because if the biggest element squared is smaller than ε then all other values will be equally small or smaller. Which means that they are so small that the possibility for round-off errors is big. As it is not possible to get ensure that the process further gives correct values the transformation loop should stop.

2.2 Tests

Several tests are run to ensure that the algorithm runs correctly. The two test presented in this section is a test of the computed *find_max* function to find the off diagonal element with the largest absolute value of a symmetric matrix and a comparison of the computed solution for a simple 2×2 case. In Chap. 3 the results gained for the one particle case using the computed Jacobi algorithm is furthermore compared to the known analytical solution.

2.2.1 Test of the *find_max* function

To find the maximum absolute value of the elements of the matrix for which we want to solve the eigenvalue problem using the Jacobi method, the following c++ source code is used.

```
void find_max(mat &A, int &n, int &row_number, int &column_number)
// Set row_number = 0 and column_number = 1, when running the code.
// These are the initial guesses for max(A(i,j))
{
        double max = A(0,1);
        for (int i=0; i<n; i++)
        {
            for (int j=i+1; j<n; j++)
            {
            if (fabs(A(i,j)) > fabs(max))
            {
                max = A(i,j);
                row_number = i;
                column_number = j;
```

```
            }
            }
        }
        return;
}
```

The programmed function *find_max* finds the entrance with the maximal absolute value amongst the entrances above the diagonal. The initial guess of the maximum absolute value of the off diagonal elements is set to a_{12} (notice that the first row/column of the matrix in the code is 0, whilst it is 1 in the text). The two for loops then run through all the elements above the diagonal, and if the absolute value of that element is greater than the absolute value of the until then computed maximal value, the new value *max* is set equal to the value of that entrance. Since the matrix $\mathbf{A}$ for this project is symmetric, it is not necessary to run through the elements below the diagonal.

To check that the *find_max* function runs as expected, a random matrix $\mathbf{A}$, with the maximum absolute value above the diagonal being $a_{25} = 6$, is considered.

$$\mathbf{A} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 3 & 4 & 5 & 6 \\ 0 & -1 & -2 & -3 & -4 \\ 0 & -3 & -6 & -9 & 0 \\ -1 & 0 & 1 & 2 & 3 \end{pmatrix} \tag{2.24}$$

When running the function for the matrix $\mathbf{A}$ in Eq. (2.24) and an initial guess that the greatest absolute value can be found as the element $a_{12} = 2$, the function outputs the maximum value:

$$\begin{aligned} &\text{max value} = 6 \\ &\text{row number} = 2 \\ &\text{column number} = 5 \end{aligned}$$

which is exactly what was expected when considering the investigated matrix. When running the function for $-\mathbf{A}$ the element with the greatest absolute value is once again found to be a_{25}. In this case the element has a value of $a_{25} = -6$, which was to be expected.

2.2.2 Testing against simple 2×2 case

To check that the Jacobi function runs correctly, consider the case with matrix dimensionality 2×2 case with $\rho_{min} = 0$ and $\rho_{max} = 6$, yielding a step length of

$$h = \frac{\rho_{max} - \rho_{min}}{n+1} = \frac{6-0}{3} = 2$$

With the potential described by ρ^2, this case gives that the matrix $\mathbf{A}$, for which the eigenvalue problem is solved, takes the form

$$\mathbf{A} = \begin{pmatrix} \frac{2}{2^2} + 2^2 & -\frac{1}{2^2} \\ -\frac{1}{2^2} & \frac{2}{2^2} + 4^2 \end{pmatrix} = \begin{pmatrix} 4.5 & -0.25 \\ -0.25 & 16.5 \end{pmatrix}$$

The entrance with the greatest absolute value of the off diagonal element is the $a_{12} = a_{21}$, which means that τ introduced in Eq. (2.14)

$$\tau = -\frac{16.5 - 4.5}{2 \cdot 0.25} = -24$$

and hence

$$\tan\theta = 24 - \sqrt{1 + 24^2} \approx -0.0208$$

which yields that

$$\cos\theta = \frac{1}{\sqrt{(1 + 48^2}} \approx 0.9998 \qquad \text{and} \qquad \sin\theta = -0.0208 \cdot 0.9998 \approx -0.0208 \tag{2.25}$$

From the values for $\cos\theta$ and $\sin\theta$, the diagonal elements of the constructed matrix **B** after one similarity transformation described in Eq. (2.12) take the form

$$b_{11} \approx 4.5 \cdot 0.9998^2 - 2 \cdot 0.25 \cdot 0.0208 \cdot 0.9998 + 16.5 \cdot 0.0208^2 \approx 4.495$$
$$b_{22} \approx 16.5 \cdot 0.9998^2 + 2 \cdot 0.25 \cdot 0.0208 \cdot 0.9998 + 4.5 \cdot 0.0208^2 \approx 16.51$$

giving

$$\mathbf{B} \approx \begin{pmatrix} 4.495 & 0 \\ 0 & 16.51 \end{pmatrix}$$

Which means that the first and second eigenvalues are 4.495 and 16.51, respectively. When running the computed Jacobi function for this 2×2 example, this is exactly what is gained. [4]

[4] FiXme Note: ref. to result

CHAPTER

3 RESULTS

When running the code presented in Chap. 2.... blah blah blah.... Let's have an intro to this chapter...

The results from running the code ... can be found in the GitHub folder `https://??`. [1]

3.1 Dependence of ρ_{max} and n on Eigenvalue

For a single electron moving in a three-dimensional harmonic oscillator potential, the analytical solution for the first three eigenvalues to the rewritten Schrdinger's equation stated in Eq. (2.2) are $\lambda_0 = 3$, $\lambda_1 = 7$, and $\lambda_2 = 11$, for $l = 0$.

In the computed code for solving the eigenvalue problem numerically using the Jacobi method, the two parameters ρ_{max} and n can be modified to give a more or less accurate numerical solution to the problem. If not considering the computational time, limit in memory, and round-off errors, the obvious preferable choice of ρ_{max} and n would be to make both numbers infinite. This is, however, not a realistic possibility, and this sections is therefore dedicated to find and discuss on the optimal choices for ρ_{max} and n to obtain acceptable values for the eigenvalues of Eq. (2.5) with the matrix $\mathbf{A}$ defined in Eq. (2.6).

Changing ρ_{max} causes the interval $[\rho_{min}, \rho_{max}]$, in which the wave function is considered, to change. Since the wave function goes to zero as the distance goes to infinity, it is acceptable to neglect the contribution from some ρ_{max}. It is an advantage to decrease ρ_{max} and hence making the interval smaller, in the sense that then a smaller n is needed to create a sufficient step length and ultimately a good enough "resolution". However, if this ρ_{max} is too close to ρ_{min} the neglected part of the eigenfunction can actually not be neglected, if an acceptable result is wished for.

In the figure below, the dependence of different integer valued ρ_{max} on the three first eigenvalues gained by computed Jacobi method is plotted for $n = 100$.

[1] FiXme Note: correct the above lines

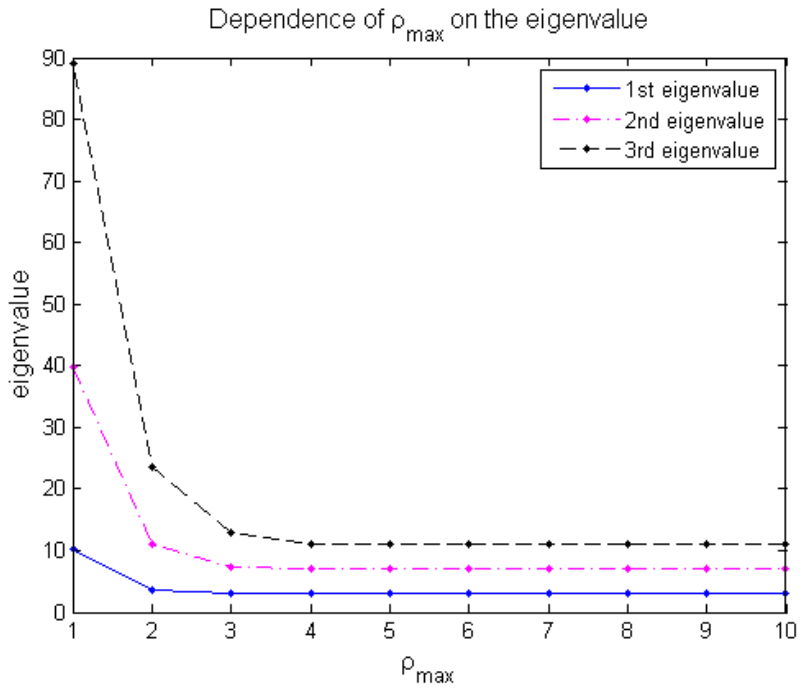

Figure 3.1. The computed eigenvalue for the ground state and the first two excited states of the one electron in a harmonic oscillator potential case plotted as functions of ρ_{max} for $\rho_{min} = 0$. For small values of ρ_{max} (that is $\rho_{max} \leq 3$ for this case) the eigenvalue is greatly varying with ρ_{max}, but for greater values of ρ_{max} the eigenvalue seems to stabilize as a function of ρ_{max} towards the analytical results for the eigenvalues of this problem.

From the it is evident that figure that if $\rho_{max} < 3$ the eigenvalues are varying dramatically. This happens due to neglect of strongly contributing parts of the eigenfunction. When ρ_{max} is so large that the greatest contributive parts of the eigenfunction is in the interval from ρ_{min} to ρ_{max}, the variation of the eigenvalue decreases strongly. However, when increasing ρ_{max} for the same value of n the step length is increased, as well, which once again influences the precision of the computed eigenvalue.

The ρ_{max} that, with $n = 100$, gives the most accurate result for all three of the first eigenvalues is $\rho_{max} = 5$. Since this ρ_{max} gives the most accurate result for a relatively small n, this is chosen as the optimal ρ_{max} in this and the following sections for this specific problem.

With this ρ_{max}, we wish to find the integer value of n that gives the first three eigenvalues with four leading digits. This optimal n is found by steady increment of n, as seen in Fig. 3.2.

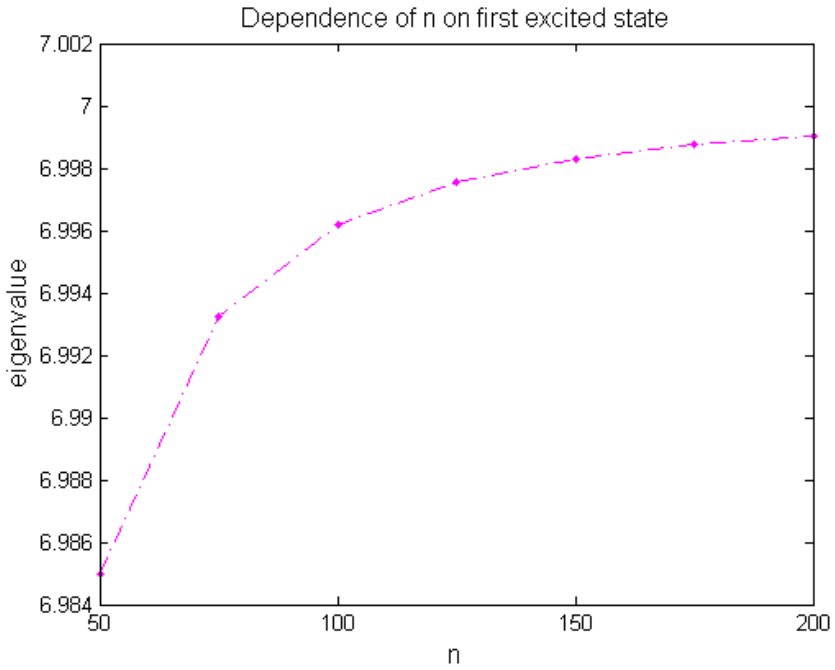

Figure 3.2. Plot of the second eigenvalue of the one electron in a simple harmonic oscillator case. The second eigenvalue is plotted as a function of the number of steps n for $\rho_{min} = 0$ and $\rho_{max} = 5$. The analytical solution for the first excited state of this case is 7, and from the graph it is evident that the numerical solution gained from the computed Jacobi method approaches the analytical solution, as n is increased. One can, however, suspect that a further increment of n can lead to greater round-off errors, and hence a worse result.

The eigenvalue of the first exited state is seen to be asymptotic to the analytical solution $\lambda_2 = 7$, and at a matrix size of $n = 200$, the eigenvalue of the first exited state has the numerical solution 6.99904. This yields an accuracy up to four leading digits, which is also found to be the case for the ground state and the third eigenstate. Hence, the optimal ρ_{max} and n is 5 and 200, respectively.

3.2 Computation Time Compared to Alternative Algorithm

When computing the time needed for the Jacobi rotation algorithm computed in this project to solve the eigenvalue problem with a matrix $\mathbf{A}$ of dimensionality 200×200, $\rho_{max} = 5$, and $\varepsilon = 10^{-8}$ the elapse time if found to be 17 sec. [2] This is a much greater value than the elapsed time for solving the same eigenvalue problem using the Armadillo function *eig_sym*, since the Armadillo function has a displayed computational time of 0 sec (the computational time is of course not 0 sec, but the precision of the displayed time is so low that the number is displayed as 0). It is hence clear that the computed Jacobi algorithm consumes more time than the precomputed Armadillo function and as we increases the size of the matrix the elapsed time to get the solution also increases. This makes Jacobi rotation algorithm less efficient.

The following figures show the relation between the number of steps n and the number of iterations of the while loop in the Jacobi algorithm computed for this project with the same ρ_{max} and ε as described in the section above.

[2] FiXme Note: ref to results

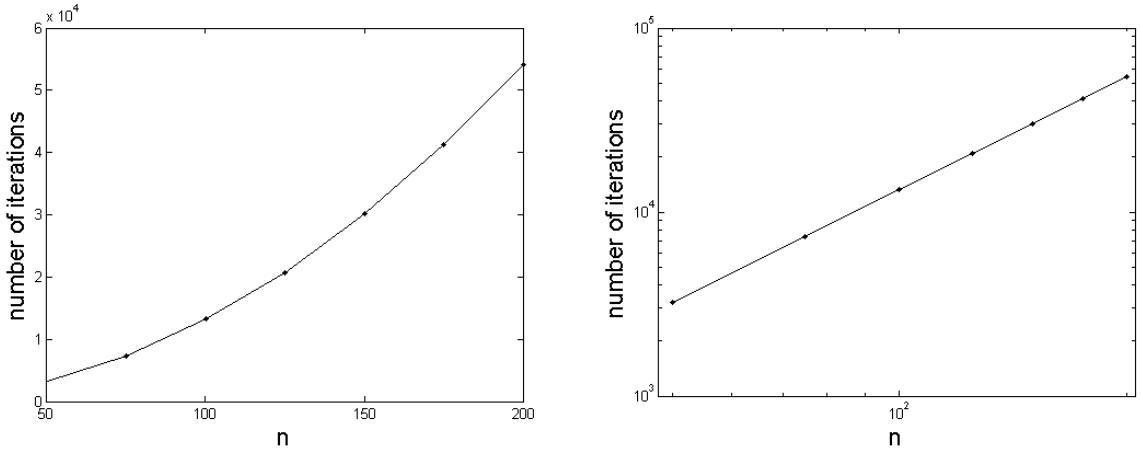

Figure 3.3. The number of iterations of the while loop in the computed Jacobi method as a function of the number of steps n plotted in both a normal graph and as a log-log plot. From the log-log plot it is evident that the number of iterations in the while loop is proportional to n^2, since the slope of the log-log plot is found to equal 2.

In the log-log plot, the relation is shown as a straight line with a slope of approximately 2, which is calculated directly from the data for $n = 100$ and $n = 200$. This yield that the number of iterations increases as the number of steps squared (that is, n^2), and hence an increment of n has a great impact on the number of iteration. [3] E.g. with $n = 100$ the number of iterations for the while loop in the Jacobi method is $13,200$, whilst when n is doubled to 200, the function runs through the while loop 54,071 times before the requirement of $max(a_{kl}^2) > \varepsilon$ in Eq. (2.23) is fulfilled.

The slowness of the Jacobi algorithm has the effect that it cannot be run for too large matrices, and hence there is a significant limit for how small the step length h can be made, and ultimately a limit to the precision of the algorithm.

It is, however, evident that the Jacobi method implemented in this project can be improved for solving the specific eigenvalue problem described by Eq. (2.5) and (2.6) by taking into account that the matrix $\mathbf{A}$ is tridiagonal and has constant values in the entrances adjacent to the diagonal. However, this improvement of the algorithm is not in the scope of this project.

New potential and repulsive Coulomb interaction

Introducing a new potential $V_\rho = \omega^2\rho^2 + \frac{1}{\rho}$ where ω is an oscillator frequency and $\frac{1}{\rho}$ is a repulsive Coulomb potential. First considering the value of ρ_{max} in the case without a repulsive Coulumb interaction. Where finding a value of ρ_{max} so that the eigenvalues are stable is the first step. [4] For this to be stable the change should be small, so $\rho_{max} = 10$ and the change is about 0.2 between the surrounding steps. Also the change between 10 and 10.1 is about 0.02 so small changes in ρ_{max} doesn't give big changes. Also the relation between ρ_{max} and n is kept constant so that [6] is constant. Running for different values of ρ_{max} with the ratio gives Tab. 3.1, which shows that when ρ_{max} is 10 the results are stable

Table 3.1. hei hei[5]

ρ_{max}	N	3rd eigenvalue
1	10	83.5237
2	20	21.8365
3	30	9.79411
4	40	5.5277
⋮	⋮	⋮
9	90	1.0983
10	100	0.890899
10.1	101	0.873488
11	110	0.737631

[3] FiXme Note: ikke sandt?
[4] FiXme Note: rewrite sentence
[6] FiXme Note: ref h

for the 3rd eigenvalue. The reason for choosing the 3rd eigenvalue is that it is more sensitive for change of ρ_{max} and it's the last printed eigenvalue in the program.

The Eigenvectors are used for determining the wave function, and in this case they are used squared so they show the probability density of the distance ρ between the two electrons. Which gives the most probable distance between the particles,the distance $\rho = (\frac{1}{\alpha})r$ where $\alpha = (\frac{\hbar^2}{mk})^{1/4}$. The reason for this is better described in Sec. 2.1. Plotting the distance as a function of ρ for different values of the frequency ω_r, Fig. 3.4 shows that for a higher frequency ω_r the distance between the particles are bigger and the spread in area which the particle can be in is also more stretched out.

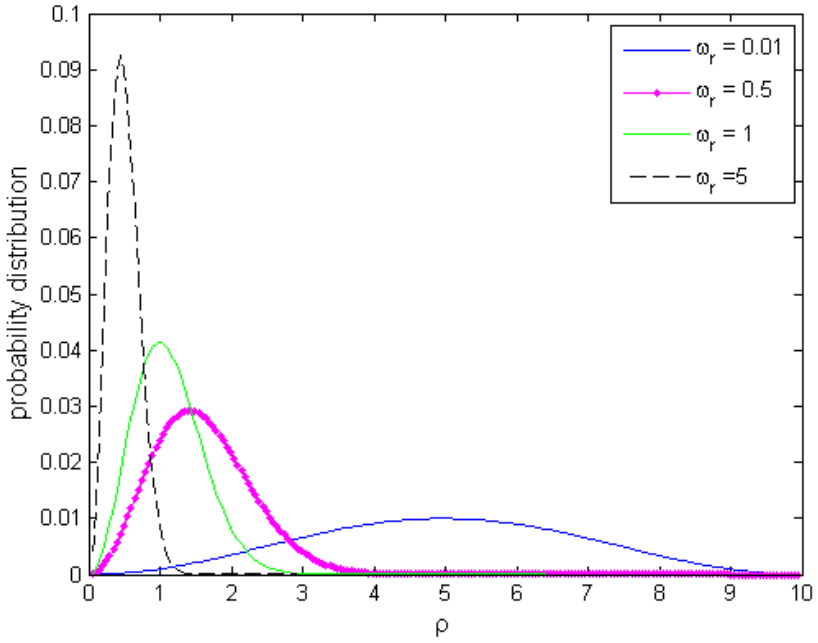

Figure 3.4. The probability density as a function of distance ρ without repulsive Coulomb interaction. Where $\rho_{min} = 0$, $\rho_{max} = 10$ and $n = 200$

When adding a repulsive Coulomb potential $\frac{1}{\rho}$ to the potential $V(\rho)$ in Fig. 3.5 the probability density is pushed to a longer distance, although it might be a bit difficult to see in Fig. 3.5 and Fig. 3.4.

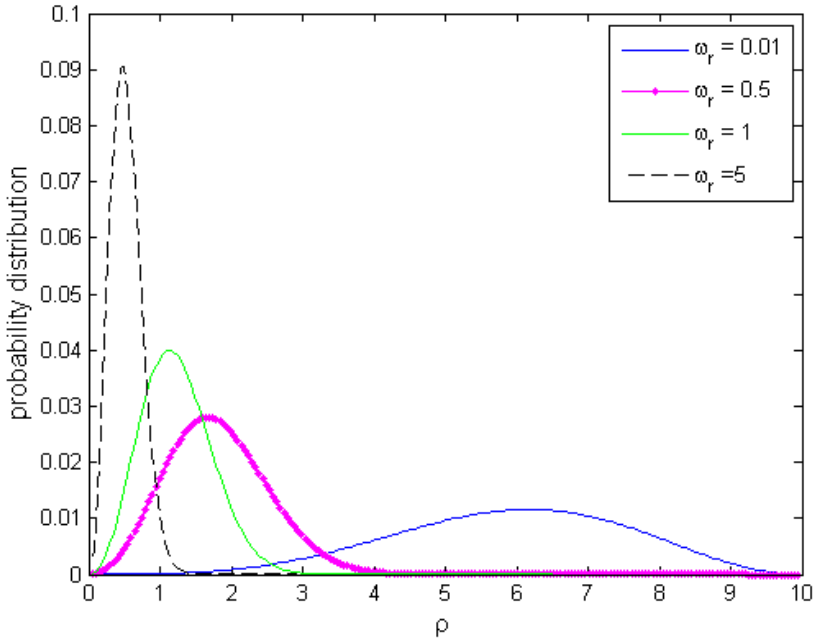

Figure 3.5. The probability density as a function of distance ρ with a repulsive Coulomb interaction. Where $\rho_{min} = 0$, $\rho_{max} = 10$ and $n = 200$

CHAPTER

4 CONCLUSION

Conclude.... conclude.... conclude....

BIBLIOGRAPHY

APPENDIX

A MATLAB CODE FOR SMT....

This is how, we write MatLab code in the report

```
close all
clear all
clc
%I am a comment

filename = 'Results.xlsx';
sheet = 4;
xlRange = 'B3:C12';

[v,T,vT] = xlsread(filename, sheet, xlRange);
x10=v(:,1);y10=v(:,2);

figure
plot(??)
legend(??)

xlim([??])
ylim([??])

title(??)
xlabel('x')
ylabel('y')
```