Computational Physics - Project 2

Johannes Scheller, Vincent Noculak, Lukas Powalla

October 5, 2015

Contents

1 Introduction and Motivation

In many fields of both mathematics and physics, we often come to the point that we have to solve so-called eigenvalue problems, which are equations of the form $\mathbf{A}\cdot\mathbf{v} = \lambda\mathbf{v}$, where $\mathbf{A}$ is a matrix of dimension $n \times n$ and v is a vector of dimension n. Equations of this kind occur not only in linear algebra, but also in mechanics and quantum mechanics and will also be a major part of this report. In this project, we are going to rewrite the Schrödinger's equation of one and two electrons in a harmonic oscillator potential in the form of an eigenvalue problem and solve it numerically by implementing Jacobi's method, an algorithm that can be used to solve any eigenvalue problem. This algorithm is based on simple rotations of a given matrix to diagonalize it. We will show how this algorithm works, set it up on our own using C++ and use this program to obtain the eigenvalues of the Schrödinger's equation of electrons in an harmonic oscillator potential that we are going to rewrite as an eigenvalue problem.

2 Theory

This theory part deals with the Schrödinger equation for one or two electrons in a tree dimensional harmonic potential. In particular, we want to solve the Schrödinger equation numerically. Therefore, we rewrite the Schrödinger equation as eigenvalue problem and solve the eigenvalue problem with Jacobi's method.

2.1 One Electron in a Harmonic Oscillator Potential

We are now looking at one electron in a harmonic oscillator potential. The time independent Schrödinger equation in general looks like:

$$\hat{H}|\Psi\rangle = E|\Psi\rangle \tag{1}$$

We are going to have a closer look at the Schrödinger equation of one electron in a harmonic oscillator. You can rewrite the Hamiltonian like it is done in eq. (2). Here, m is the mass of the particle, $V(r) = \frac{1}{2}k\cdot r^2$ is the potential, $\hat{p_r} = -i\hbar\frac{1}{r}\frac{\partial}{\partial r}r$ is the radial component of the momentum operator, $E = \hbar\omega(2n+l+\frac{2}{3})$ is the energy and $\hat{L}$ is the angular momentum operator.

$$\left(\frac{1}{2m}\left[\hat{p_r}^2 + \frac{\hat{L}^2}{r^2}\right] + V(r)\right)\Psi = E\Psi \tag{2}$$

We can now solve this equation with an product ansatz $\Psi = R(r)\cdot Y(\Theta,\phi)$. The angular-part provides spherical harmonics (as long $V(r) = \frac{1}{2}k\cdot r^2$ is only a function of radius). We are interested in the solution for R(r). After eliminating the angular-dependant part, we receive eq. (3). (for further information look at "The Physics of Atoms and Quanta -Introduction to Experiments and Theory " written by Herman Haken and Hans Christoph Wolf - Chapter 10: Quantum Mechanics of the Hydrogen Atom)

$$-\frac{\hbar^2}{2m}\left(\frac{1}{r^2}\frac{d}{dr}r^2\frac{d}{dr} - \frac{l(l+1)}{r^2}\right)R(r) + V(r) = ER(r) \tag{3}$$

We want to solve this equation numerically. In this project, we are only considering l=0. In order to have Dirichlet boundary conditions, we substitute $R(r) = \frac{1}{r}\cdot u(r)$. (where r is element of [0,∞)) The boundary conditions for the new variable are u(0)=0 and u(∞)=0.

$$-\frac{\hbar^2}{2m}\frac{d^2}{dr^2}u(r) + V(r) = Eu(r) \tag{4}$$

In order to have more general solution of our equation, we rewrite eq. (3) in dimensionless variables.
we define:

$$\alpha = \left(\frac{\hbar^2}{mk}\right)^{\frac{1}{4}} \tag{5}$$

$$\lambda = \frac{2m\alpha^2}{\hbar^2}E \tag{6}$$

$$\rho = \frac{1}{\alpha}r \tag{7}$$

Finally, we can rewrite the Schrödinger equation in a dimensionless form. (equation 10) We know already the values for λ since we know energy values. (re-substitution using equations 6, 7 and the $E_{l=0}$)

$$\lambda(n) = 4\cdot n + 3 \tag{8}$$

$$n = 0, 1, 2, 3... \tag{9}$$

$$-\frac{d^2}{d\rho^2}\cdot u(\rho)+\rho^2 u(\rho)=\lambda u(\rho) \tag{10}$$

We use the standard approximation for the second derivative of u(ρ) (formula 11)

$$\frac{du(\rho)^2}{d\rho^2}=\frac{u(\rho+h)-2u(\rho)+u(\rho-h)}{h^2} \tag{11}$$

We also have to define the minimum and maximum of our new variable ρ. We choose the minimum of $\rho_{min}=0$. For the maximum, we tried different values. The maximum value of ρ effects the numerical precision of the result. However, this impact will be discussed in chapter 4.

Since we want to solve the given equation numerically, we discretize ρ. We choose to have n grid points for ρ. In this case, ρ and the step length h are defined as follows:

$$h=\frac{\rho_{max}-\rho_{min}}{n_{step}} \tag{12}$$

$$\rho_i=\rho_{min}+i\cdot h \quad ,i=0,1,2... \tag{13}$$

We finally end up with eq. (14). This equation can be written as a Eigenvalue problem. In this case, λ is the eigenvalue and the left side of equation 14 can be rewritten as a matrix/vector-product.

$$\frac{u(\rho_i+h)-2u(\rho_i)+u(\rho_i-h)}{h^2}+V_i(\rho_i)\cdot u(\rho_i)=\lambda u(\rho_i) \tag{14}$$

$$V_i(\rho_i)=\rho_i^2 \tag{15}$$

$$A\cdot\mathbf{u}(\rho_i)=\lambda\cdot\mathbf{u}(\rho_i) \tag{16}$$

Where **A** is defined as:

$$\mathbf{A}:=\begin{pmatrix} \frac{2}{h^2}+V_1 & -\frac{1}{h^2} & 0 & 0 & \dots & 0 & 0 \\ -\frac{1}{h^2} & \frac{2}{h^2}+V_2 & -\frac{1}{h^2} & 0 & \dots & 0 & 0 \\ 0 & -\frac{1}{h^2} & \frac{2}{h^2}+V_3 & -\frac{1}{h^2} & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & \dots & \dots & \dots & \frac{2}{h^2}+V_{n_{\text{step}}-2} & -\frac{1}{h^2} \\ 0 & \dots & \dots & \dots & \dots & -\frac{1}{h^2} & \frac{2}{h^2}+V_{n_{\text{step}}-1} \end{pmatrix} \tag{17}$$

Now, we have rewritten the Schrödinger equation as an eigenvalue problem in eq. (16) with given Matrix **A** (Matrix 17). The given eigenvalue problem (eq. (16)) is now solvable with Jacobi's method, which is discussed in section 2.3.

2.2 Two interacting electrons in a harmonic oscillator potential

In this capture, we want to solve two electrons interacting in a harmonic oscillator potential. We can start with a single electron equation in formula 18. We did all the previous steps as in capture 2.1.

$$-\frac{\hbar^2}{2m}\frac{d^2}{dr^2}u(r)+\frac{1}{2}k\cdot r^2 u(r)=E^{(1)}u(r) \tag{18}$$

If we don't consider repulsive coulomb interaction between the two electrons, we get equation 19. In this case, we have a two Electron Energy and a two electron radial wave function ($u(r_1,r_2)$).

$$\left(-\frac{\hbar^2}{2m}\frac{d^2}{dr_1^2}-\frac{\hbar^2}{2m}\frac{d^2}{dr_1^2}+\frac{1}{2}k\cdot r_1^2+\frac{1}{2}k\cdot r_2^2\right)u(r_1,r_2)=E^{(1,2)}u(r_1,r_2) \tag{19}$$

In order to describe the system with more demonstrative variables, we choose to describe the system with a relative coordinate r and a center-of mass coordinate R instead of using r_1,r_2.

$$r=r_1-r_2 \tag{20}$$

$$R=\frac{r_1+r_2}{2} \tag{21}$$

We can now describe the equation 18 with these new variables. This equation is shown in equation 22

$$\left(-\frac{\hbar^2}{m}\frac{d^2}{dr^2}-\frac{\hbar^2}{4m}\frac{d^2}{dR^2}+\frac{1}{4}kr^2+kR^2\right)u(r,R)=E^{(1,2)}\cdot u(r,R) \tag{22}$$

This equation can be solved with a product ansatz for the wave function $u(R,r)=\phi(R)\cdot\xi(r)$. In this case, the Energy $E^{1,2}=E_r+E_R$ is the sum of the relative Energy and the center of mass energy. we also want to take the Coulomb interaction between the two electrons. Therefore, we ad the repulsive Coulomb interaction.

$$V_{Coulomb}=\frac{\beta e^2}{|\mathbf{r_1}-\mathbf{r_2}|}=\frac{\beta e^2}{r} \tag{23}$$

We can separate the equation 22 into a part, which is only dependant on r and one part, which is only dependant on R. Adding the coulomb term, we derive equation 24 for the r dependence of the Schrödinger equation.

$$\left(-\frac{\hbar^2}{m}\frac{d^2}{dr^2}+\frac{1}{4}kr^2+\frac{\beta e^2}{r}\right)\xi(r)=E_r\xi(r) \tag{24}$$

Analogue to chapter 2.1, we rewrite equation 24 and introduce dimensionless variables $\rho=\frac{1}{\alpha}$.

$$\left(-\frac{d^2}{dr^2}+\frac{1}{4}\frac{mk}{\hbar^2}\alpha^4\rho^2+\frac{\beta e^2\alpha m}{\rho\hbar^2}\right)\xi(r)=\frac{m\alpha^2}{\hbar^2}E_r\xi(r) \tag{25}$$

In order to make equation 25 dimensionless, we would like to rewrite it in such a way that we can perform the same steps to solve the equation like in chaper 2.1.

$$\omega_r=\frac{1}{4}\frac{mk}{\hbar^2}\alpha^4 \tag{26}$$

$$\alpha=\frac{\hbar^2}{m\beta e^2} \tag{27}$$

$$\lambda=\frac{m\alpha^2}{\hbar^2}E \tag{28}$$

The final, dimensionless equation is equation 29.

$$\left(-\frac{d^2}{d\rho^2}+\omega_r^2\rho^2+\frac{1}{\rho}\right)\xi(\rho)=\lambda\xi(\rho) \tag{29}$$

We can solve equation 29 numerically by approximation of the second derivative in the same way like in chapter 2.1. We just have to reintroduce the potential V_i. Afterwards, we can solve the equation 30 like we solved equation 14.

$$-\frac{\xi(\rho_i+h)-2u(\rho_i)+\xi(\rho_i-h)}{h^2}+V_i(\rho_i)\cdot\xi(\rho_i)=\lambda\xi(\rho_i) \tag{30}$$

$$V_i(\rho_i)=\left(\omega_r^2\rho_i^2+\frac{1}{\rho_i}\right) \tag{31}$$

2.3 Jacobi's method

Jacobis method is a iterative method to solve a eigenvalue-problem.

$$\mathbf{Ax}=\mathbf{b} \tag{32}$$

Assuming **A** is real and symmetric, meaning $\mathbf{A}\in\mathbb{R}^N\times\mathbb{R}^N$, then eq. (32) is solved by N eigenvalues $\lambda_1\ldots\lambda_N$ and there exists a transformation matrix **S** such that $\mathbf{S}^{\mathrm{T}}\mathbf{AS}=\mathbf{D}$. In this case, **D** is a diagonal matrix containing the eigenvalues in the following way:

$$\mathbf{D}=\begin{pmatrix}\lambda_1 & 0 & \ldots & \ldots & 0\\ 0 & \lambda_2 & 0 & \ldots & 0\\ \ldots & \ldots & \ldots & \ldots & \ldots\\ 0 & \ldots & \ldots & \ldots & \lambda_N\end{pmatrix} \tag{33}$$

Now, to obtain the eigenvalues of $\mathbf{A}$, we perform a series of similarity transformations on it to reduce it to a diagonal matrix. A matrix $\mathbf{A}'$ is a similarity transform of $\mathbf{A}$ if $\mathbf{S}^{\mathrm{T}}\mathbf{A}\mathbf{S} = \mathbf{A}'$ and $\mathbf{S}^{\mathrm{T}}\mathbf{S} = \mathbb{1}$. It has got the same eigenvalues as the original matrix, but in general different eigenvectors. The idea of Jacobi's method is to use a series of rotation matrices $\mathbf{S}_i$ to reach this goal. At each step, we look for the largest off-diagonal element a_{kl} of our matrix and set up a rotation matrix such that this element becomes zero in our transformed matrix. Assuming we start with the original matrix $\mathbf{A}_0 := \mathbf{A}$ and a_{kl} is the highest off-diagonal element of $\mathbf{A}_0$, the first transformation matrix $\mathbf{S_1}$ has the following form:

$$\mathbf{S} = \begin{pmatrix} 1 & 0 & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 1 & 0 & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & \dots & s_{kk} = \cos(\theta) & 0 & \dots & s_{lk} = \sin(\theta) & 0 & \dots \\ 0 & \dots & \dots & \dots & 1 & 0 & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & \dots & s_{kl} = -\sin(\theta) & 0 & \dots & s_{ll} = \cos(\theta) & 0 & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & 1 \end{pmatrix} \tag{34}$$

In the next step, we do the same for the transformed matrix $\mathbf{A_2} = \mathbf{S}^{\mathrm{T}}{}_1\mathbf{A_1}\mathbf{S}_1$. Since we always chose $\mathbf{S}$ such that the largest off-diagonal of $\mathbf{A}_i$ gets zero, we ensure that our matrix converges towards a diagonal matrix $\mathbf{D} = \mathbf{S}^{\mathrm{T}}{}_m \dots \mathbf{S}^{\mathrm{T}}{}_1\mathbf{A}\mathbf{S}_1 \dots \mathbf{S}_m$. We repeat these steps until the highest off-diagonal value of $\mathbf{A}_i$ is smaller than a chosen tolerance that we call ϵ.

To chose the right angle θ, we take a closer look at the largest off-diagonal element of our matrix, a_{kl}: It is supposed to become zero after the transformation, meaning that $a'_{kl} = (a_{kk} - a_{ll})\cos(\theta)\sin(\theta) + a_{kl}\left(\cos^2(\theta) - \sin^2(\theta)\right) \stackrel{!}{=} 0$. Hence, we get the following equation for $t := \tan(\theta) =: \frac{s}{c}$ and $\tau := \frac{a_{ll} - a_{kk}}{2a_7 kl}$:

$$t^2 + 2\tau t - 1 = 0 \tag{35}$$

, which results in

$$t = \pm\sqrt{1 + \tau^2} \tag{36}$$

Now we can also obtain $c = \frac{1}{\sqrt{1+t^2}}$ and $s = tc$. To ensure that $|\theta| \leq \frac{\pi}{4}$, we always take the smaller of the roots in eq (36). Summing up, the steps for Jacobi's method are the following:

1. Look for the highest off-diagonal element a_{kl} of $\mathbf{A}_i$.
2. If this element is higher than a chosen tolerance ϵ, set up a rotation matrix $\mathbf{S}_i$ such that $a'_{kl} = 0$ after the transformation.
3. Repeat.

After all, we end up with a matrix that is at least very close to the diagonal one containing all the eigenvalues. It can be shown that the matrix indeed converges towards a diagonal form, but the convergence is very slow. depending on some other parameters, we ended up with between $0.8n^2$ and $1.6n^2$ rotations (compare fig. 2) for a matrix of dimensions $n \times n$, which leads to a total number of about $8n^3$ operations!

3 Execution

3.1 Implementing the algorithm

Our main goal in this project was to implement our own algorithm for Jacobi's method to compute the eigenvalues of the equation discussed above. This was done in the source files that are found at our github repository. First, our program asks for the desired number of steps n and sets up the matrix and the vectors for the eigenvalue equation as described in section 2.1, as well as a matrix $\mathbf{R}$ which is used for to compute the eigenvectors respectively the eigenfunctions. Then, it starts the actual Jacobi's method: It looks for the absolutely highest non-diagonal element of the matrix in the function *maxoffdiag*, and, while this is higher than the chosen tolerance ϵ, it performs a rotation around the line and the column of this element in the function *rotation*. This function obtains the values of τ, $\sin(\theta)$, $\cos(\theta)$ and $\tan(\theta)$ by using the formulas that we introduced in section 2.3.

```
//The Jacobi algorithm gets executed
while ( fabs(max_offdiag) > epsilon && iterations < max_number_iterations )
{

        rotate ( A, R, k, l, n );
        max_offdiag = maxoffdiag ( A, &k, &l, n );
        iterations++;

}
// Function to find the maximum matrix element
double maxoffdiag ( double ** A, int * k, int * l, int n )
{
        double max = 0.0;
        for ( int i = 0; i < n-1; i++ )
        {
                for ( int j = i + 1; j < n-1; j++ )
                {
                        if ( fabs(A[i][j]) > max )
                        {
                                max = fabs(A[i][j]);
                                *l = i;
                                *k = j;
                        }
                }
        }
        return max;
}
```

```
// Function to find the values of cos and sin and then to rotate
void rotate ( double ** A, double ** R, int k, int l, int n )
{
        double s, c;
        if ( A[k][l] != 0.0 )
        {
                double t, tau;
                tau = (A[l][l] - A[k][k])/(2*A[k][l]);
                if ( tau > 0 )
                {
                        t = 1.0/(tau + sqrt(1.0 + tau*tau));
                }
                else
                {
                        t = -1.0/( -tau + sqrt(1.0 + tau*tau));
                }
                c = 1/sqrt(1+t*t);
                s = c*t;
        }
        else
        {
                c = 1.0;
                s = 0.0;
        }
        double a_kk, a_ll, a_ik, a_il, r_ik, r_il;
        a_kk = A[k][k];
        a_ll = A[l][l];
        // changing the matrix elements with indices k and l
        A[k][k] = c*c*a_kk - 2.0*c*s*A[k][l] + s*s*a_ll;
        A[l][l] = s*s*a_kk + 2.0*c*s*A[k][l] + c*c*a_ll;
        A[k][l] = 0.0;
        A[l][k] = 0.0;
        // and then we change the remaining elements
        for ( int i = 0; i < n-1; i++ )
        {
                if ( i != k && i != l )
                {
                        a_ik = A[i][k];
                        a_il = A[i][l];
                        A[i][k] = c*a_ik - s*a_il;
                        A[k][i] = A[i][k];
                        A[i][l] = c*a_il + s*a_ik;
                        A[l][i] = A[i][l];
                }
                //Compute new eigenvectors in its matrix
                r_ik = R[i][k];
                r_il = R[i][l];
                R[i][k] = c*r_ik - s*r_il;
                R[i][l] = c*r_il + s*r_ik;
        }
        return;
}
```

As a simple test case for our algorithm, we took a 2×2 matrix which it solved correctly within 2 steps. This indicates that the program is working correctly, a fact that is highlighted by the correct results for the eigenvalues that we obtained during this project. At the end, our program gives the user the possibility to print out all eigenvalues, a specific one or only the first three eigenvalues.

3.2 Setting and Testing of Parameters

In both cases, whether we deal with only one particle or with two, we have three parameters to be set, resulting in two degrees of freedom that have an effect on the accuracy of our results. We tested several set-ups for those three parameters for the one particle case to obtain the best and took slight changes to adjust it for the case with two interacting particles.

The first and most obvious parameter is n, the number of grid points we use. Using a higher value of n, we gain more precision as the step length decreases, but at the same time, we will end up with a larger matrix that needs more memory (proportional to n^2). Most important, the number of similarity transformations needed to calculate the eigenvalues goes like n^3, leading to a very long computation time for large matrices. The highest possible value we used for n was 1000, resulting in more than 45 minutes of computation time! Altering n had the most effect on our results, so we decided to take a relatively high value for this parameter.

The second parameter that we can alter is ρ_{max}, the maximum value of ρ. In theory, this value should be infinite, which is just not possible for this numeric solution. In our case, the higher an eigenvalue is, the more its calculation depends on the choice of ρ_{max}. Therefore the challenge was to set this parameter to a value which resulted in stable and consistent results for the first three eigenvalues without being to high, as a higher value would also increase our step length h if we don't change n accordingly. Choosing the right value for this parameter wasn't easy as a higher value would affect the lowest eigenvalue to become more imprecise due to the increasing step length, but would at the same time improve our results for the higher eigenvalues. This is the case since the eigenfunctions of higher eigenvalues extend to a larger value of ρ and are therefore affected more by a relatively small value of ρ_{max}.

The last degree of freedom is to set the tolerance for the non-diagonal matrix elements that are supposed to become zero. This value determines implicitly how many similarity transformations are being operated until the non-diagonal elements are considered zero. A smaller value can lead to higher precision in the eigenvalues, but will at the same time increase the computation time again. We found out that changing this value had a comparably small effect on the final results: We tried the values of $\epsilon = 10^{-8}$ and $\epsilon = 10^{-10}$, which led to no difference at all in five leading digits for the five lowest eigenvalues! Therefore, we decided to choose $\epsilon = 10^{-8}$, which obviously needed less computation time.

As said above, we tested different set-ups with different values of n and ρ_{max} with the results shown in tab. 1. As we wanted a precision of three leading digits for the three lowest eigenvalues, we decided to use the set-up with $n = 400$, $\rho_{max} = 6$ and $\epsilon = 10^{-8}$ for our final results of the one particle case, which seemed to be a good compromise between precision and computation time and led to the desired results. For the two particle case, we had to perform some changes to obtain stable results, which are explained in section 3.3.

Table 1: Comparing different set-ups of the parameters n and ρ_{max} for the one particle case

$\epsilon = 10^{-8}$	1. Eigenvalue	2. Eigenvalue	3. Eigenvalue	4. Eigenvalue	5. Eigenvalue
$n = 200$, $\rho_{max} = 5$	2.9998	6.9990	10.9979	15.0024	19.0751
$n = 200$, $\rho_{max} = 8$	2.9995	6.9975	10.9939	14.9887	18.9819
$n = 200$, $\rho_{max} = 10$	2.9992	6.9961	10.9905	14.9823	18.9717
$n = 500$, $\rho_{max} = 5$	2.99997	6.9999	10.9998	15.0055	19.0756
$n = 500$, $\rho_{max} = 8$	2.99992	6.9996	10.9990	14.9982	18.9971
$n = 500$, $\rho_{max} = 10$	2.99987	6.9994	10.9985	14.9972	18.9955
$n = 1000$, $\rho_{max} = 5$	2.99999	6.99996	11.0001	15.0059	19.0746
$n = 1000$, $\rho_{max} = 8$	2.99998	6.9999	10.9998	14.9995	18.9993
$n = 1000$, $\rho_{max} = 10$	2.99997	6.99984	10.9996	14.9993	18.9989
Exact values	3	7	11	15	19

3.3 Results

By calculating the first three eigenvalues with our Jacobi algorithm for $n = 400$, $\rho_{max} = 6$ and $\epsilon = 10^{-8}$,the values we obtain are 2.99993, 6.99965 and 10.9991. Those eigenvalues match good with the analytical eigenvalue which are 3, 7 and 11.

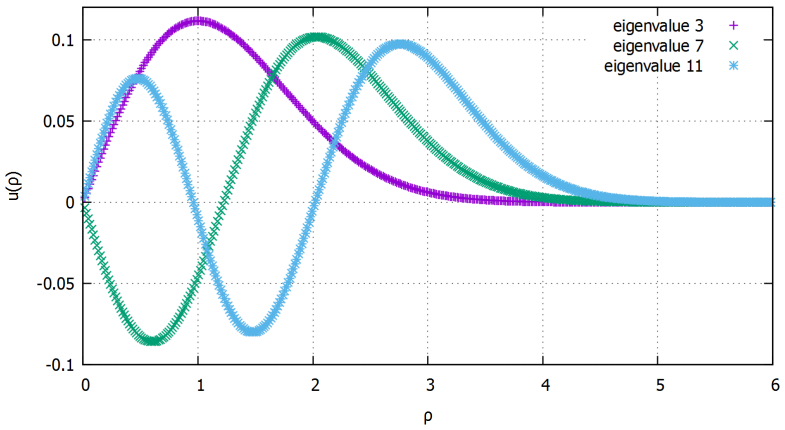

Figure 1: Functions to different eigenvalues

In figure 1 the functions to these eigenvalues can be seen.

Because the Jacobi algorithm is not specifically designed for triangular matrices like we have, calculating the eigenvalues for a high n requires a lot of iterations. Hence, for higher n, we need a lot of time to calculate the values. In our case it makes more sense to use an algorithm which is designed for a triangular matrix, like Householder's algorithm. Table 2 backs this up. While the Householder algorithm takes less than two seconds execution time for a matrix with $n = 500$, the time for the Jacobi algorithm increases quickly for bigger n and is already at 696 seconds for $n = 500$.

n	Jacobi	Householder
200	15s	<1s
250	38s	<1s
500	696s	1s
700		5s
1000		24s

Table 2: Execution time needed for the Jacobi and Householder algorithm with a $n \times n$ matrix

Figure 2 shows the needed iterations for a $n \times n$ matrix in our Jacobi algorithm for two different values of ρ_{max}. It can be seen that the number of iterations needed is proportional to n^2. This reflects why the execution time increases that quickly for higher n. While the number of the needed iterations can be approximated with $f(n) = 1.6 \cdot n^2$ for $\rho_{max} = 10$, we can approximate it just with $g(n) = 0.08 \cdot n^2$ for $\rho_{max} = 100$. This big decrease in iterations for bigger ρ_{max} is due to the fact that the step length is defined by $h = \frac{\rho_{max}}{n}$. Hence the first non-diagonal matrix elements are given by $e = \frac{n^2}{\rho_{max}^2}$. Because of that the first non-diagonals get rapidly smaller for bigger ρ_{max} and as a consequence all non-diagonal matrix elements are smaller than ϵ in less similarity transformations.

By computing the eigenvalues for two electrons in a harmonic oscillator for different ω_r with $n = 250$ and $\rho_{max} = 3$ we get the results seen in table 3. When looking at the calculated first eigenvalue for different ρ_{max} and n, it can be seen that the value is dependent on those variables. While the first eigenvalue quickly changes for lower n, it stays nearly stable for bigger n which give the better approximation for it. If we choose a ρ_{max} bigger than three, the eigenvalue stays nearly the same, but slowly decreases with increasing ρ_{max}. It has to be mentioned that the plot shows very big range of values for ρ_{max}. For at least $\rho_{max} = 4$, the function to the first eigenvalue can clearly be considered as 0 as you can see in figure 5. Hence, the approximation of our algorithm, that we assume $\psi(\rho_{max}) = 0$, is probably not the reason the eigenvalue changes. It is more likely that the direct influence of ρ_{max} on the first non-diagonal element e of the matrix A is the reason for the change of the eigenvalue (like we explained, when we looked at the number of iterations for $\rho_{max} = 100$)

In figure 4 and 5 the functions to the first eigenvalue for different ω_r can be seen. It can be observed that, the smaller ω_r is, the more the eigenfunction gets forced against zero. Hence the average distance between the two electrons gets smaller. One way to interpret this is, that through the increasing potential(which directly depends on ω_r) it gets harder for the Coulomb interaction between the electrons to push them apart.

Next we plotted the wave function depending on r in figure 6. In this plot it can be seen again that for a increasing

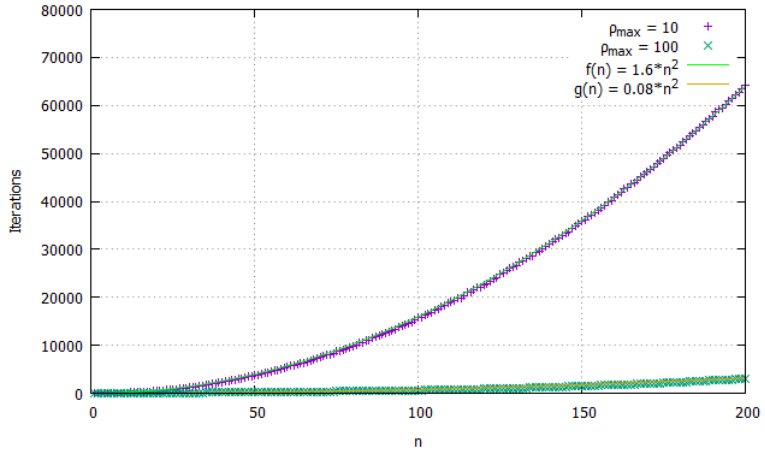

Figure 2: Number of iterations needed for a $n \times n$ matrix($\epsilon = 10^{-8}$)

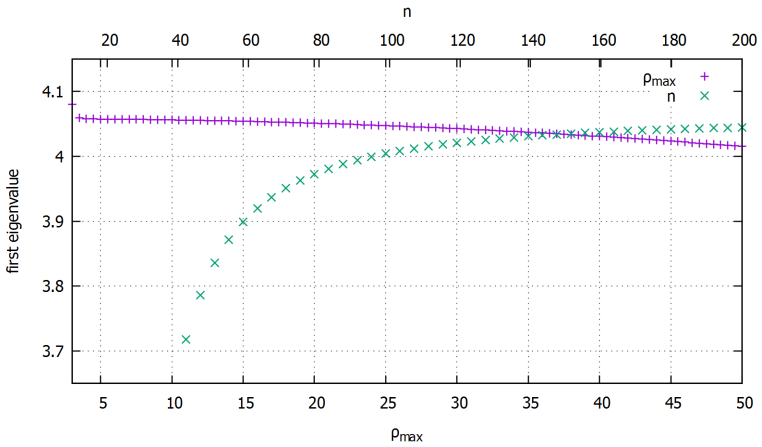

Figure 3: Dependence of the first eigenvalue of ρ_{max} and n

ω_r	Eigenvalue
0.01	0.10577
0.5	2.2300
1.0	4.0578
5.0	17.448

Table 3: Eigenvalues for different ω_r

potential (this time we varied it with the variable k), the electrons are more likely to be less distant to each other. It can also be observed that the peak of the wave function gets wider for smaller k.

When we now turn off the coulomb interaction between the two electrons (table 7), we observe that we obtain the same function like when we observed just one electron. In figure 8 the functions to the first eigenvalues of our problem with two electrons can be seen. Our calculated eigenvalues for these states are 1.5206 eV, 1.5253 eV and 1.5301 eV(with $n = 500$, $r_{max} = 3nm$, $k = \frac{eV}{nm2}$).

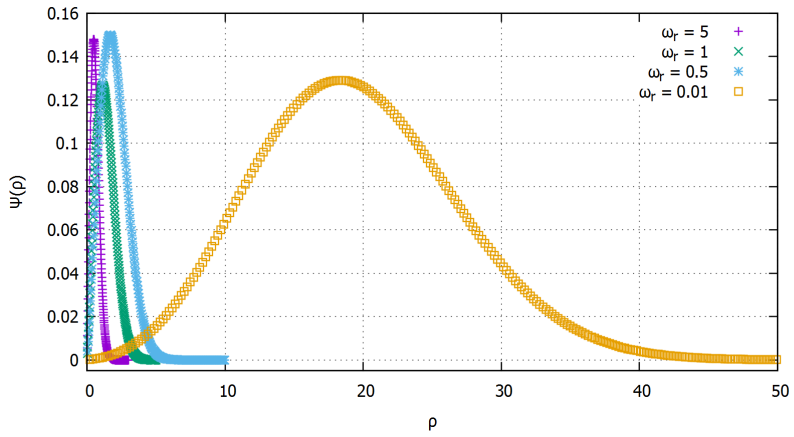

Figure 4: Two electrons in a harmonic oscillator for different values of ω_r (1)

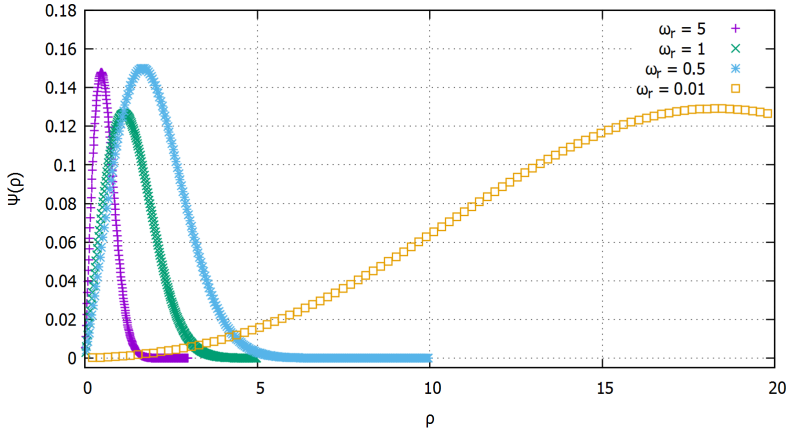

Figure 5: Two electrons in a harmonic oscillator for different values of ω_r (2)

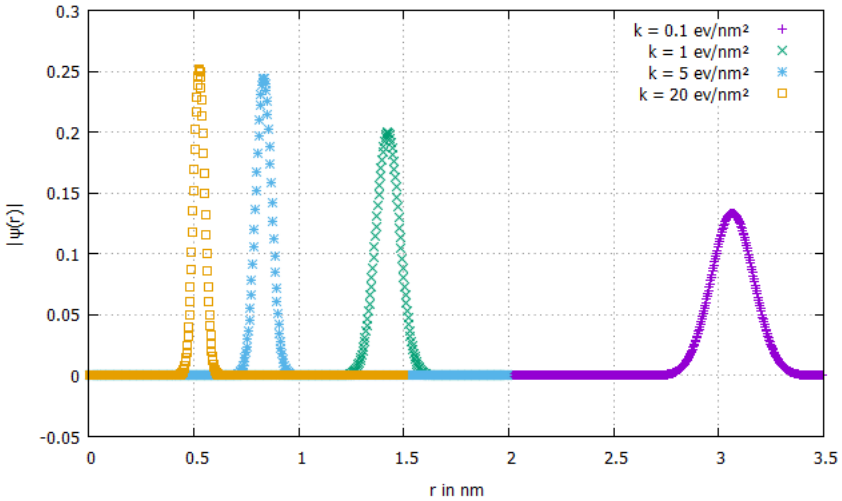

Figure 6: Two electrons in a harmonic oscillator for different values of k

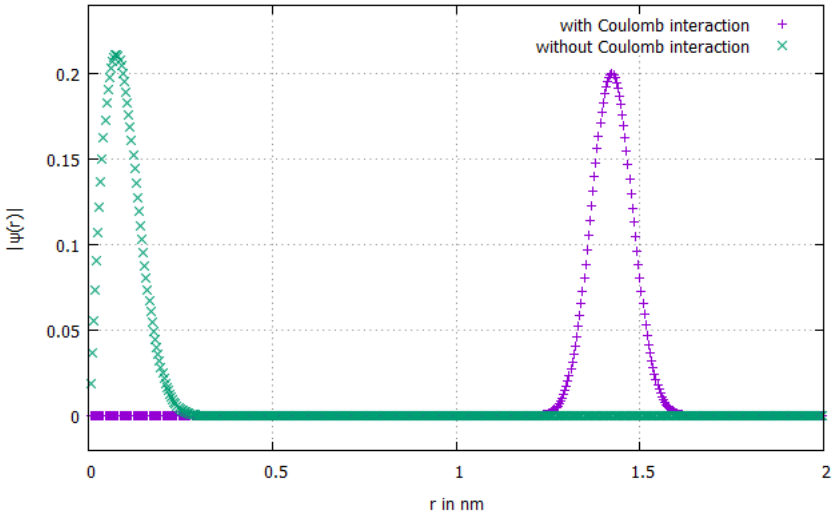

Figure 7: Two electrons in a harmonic oscillator with and without Coulomb interaction

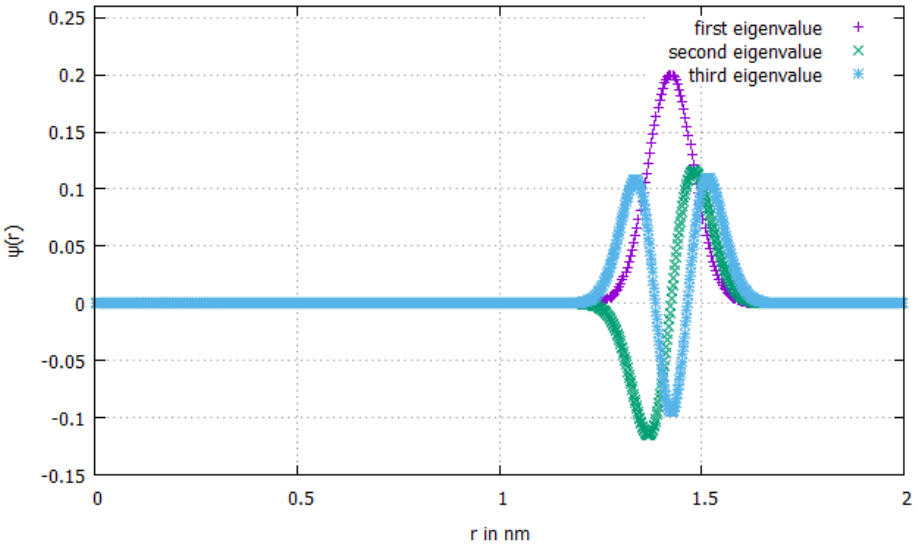

Figure 8: Functions to the first three eigenvalues for two electrons

4 Comparison and discussion of the results

All in all, we can say that our algorithm computes reasonable and fairly exact eigenvalues for the case of one electron in a harmonic oscillator that match the analytical values very well. Furthermore, the eigenfunctions seem to be consistent both for the one particle as well as for the two particle case. Nevertheless, the computation time needed by Jacobi's method increases very fast (quadratically) with the number of grid points. Therefore, this algorithm is not the method of choice for this particular problem. We would rather recommend using an algorithm specially designed for tri-diagonal matrices as this is the case for our matrix. For example, the algorithm that is used in the library provided at the Github repository of the course is much faster, as shown above.

The eigenfunctions that we obtained for the two particle case show the effects of the coulomb interaction between the electrons and how changing the oscillator potential affects the distance between the electrons: The lower the Coulomb repulsion or the higher the oscillator potential is, the smaller is the most probable value for the distance between the two particles. This is exactly what we expected qualitatively, although we did not compare the exact results for the two particle case as it is hard to find literature with values for this particular problem.

To sum up, we observed that, although Jacobi's method provided precise results, programs using Householder's algorithm combined with tri-diagonal solvers seem to have a high potential in solving eigenvalue problems in general in a fast and efficient way.

5 Source code

All the source code used in this project can be found at our Github repository: https://github.com/vincentn1/Project-2