Using Jacobi's rotation algorithm to solve the two electron Schrödinger equation.

Ole Gunnar Johansen

October 5, 2015

Abstract

In this project, I have implemented the Jacobi rotation algorithm to look at the eigenstates of one and two electrons in the harmonic oscillator potential. I have found that the eigenvalues for the three lowest states to a very high precision. In the one electron case, they were found to be $\lambda_1 = 2.999$, $\lambda_2 = 6.999$ and $\lambda_3 = 10.997$ compared to the analytical 3, 7 and 11 respectively.

There is also made a discussion of the speed of the algorithm compared to other methods, and in relation, pros and cons connected with the Jacobi method compared to one other method. The main idea here is that the Jacobi method is intrinsically slow, but easy to parallelize.

The source code for this project can be found at my github[1].

1 Introduction

Eigenvalue problems are of great importance in the modern sciences. One such example is the Schrödinger equation in quantum mechanics. The problem we are facing is, however, that such problems have analytical solutions only to a very small number of cases. In this project, I will solve the radial part of the Schrödinger equation for one and two electrons in a harmonic oscillator potential with and without repulsive Coulomb interaction using the Jacobi's rotation method for finding eigenvalues.

The report will be naturally structured into the one electron case and the two electron case, and additional results and discussions specific to the Jacobi method as separate sub sections.

2 Method

2.1 Jacobi's rotation algorithm

The aim of this method is to transform a general, symmetric, $n \times n$ matrix $\mathbf{A}$ into a diagonal matrix $\mathbf{D}$ through a series of similarity transforms. A similarity matrix $\mathbf{S}$ has the property that, when used in a similarity transform on $\mathbf{A}$, the similar matrix $\mathbf{B}$ has the same eigenvalues as $\mathbf{A}$. A similarity transform is defined as

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

---

[1] https://github.com/olegjo/FYS3150-project-2

where we require $\mathbf{S}^T\mathbf{S} = \mathbf{S}^{-1}\mathbf{S} = \mathbf{I}$. It is clear then, that this has the same eigenvalues as $\mathbf{A}$:

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{x}$$
$$(\mathbf{S}^T\mathbf{A}\mathbf{S})(\mathbf{S}^T\mathbf{x}) = \lambda S^T\mathbf{x}$$
$$\mathbf{B}\mathbf{S}^T\mathbf{x} = \lambda\mathbf{S}^T\mathbf{x}$$

In Jacobi's rotation method, the similarity matrix $\mathbf{S}$ is defined as

$$\mathbf{S} = \begin{pmatrix} \mathbf{I} & & & & \mathbf{0} \\ & \cos\theta & \cdots & \sin\theta & \\ & \vdots & \ddots & \vdots & \\ & -\sin\theta & \cdots & \cos\theta & \\ \mathbf{0} & & & & \mathbf{I} \end{pmatrix}$$

i.e. the $n \times n$ identity matrix with the elements s_{kk}, s_{ll}, s_{kl} and s_{lk} changed so that

$$s_{kk} = s_{ll} = \cos\theta, \; s_{kl} = -\sin\theta \text{ and } s_{lk} = \sin\theta$$

where the angle of rotation θ can be chosen arbitrarily. The similarity transform $\mathbf{B} = \mathbf{S}^T\mathbf{A}\mathbf{S}$ then results in

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

The algorithm is then, for each iteration: find the largest off-diagonal value in $\mathbf{A}$, a_{kl}. Choose the angle θ so that the non-diagonal element in the similarity matrix $b_{kl} = 0$. Defining $t = \tan\theta = s/c$, $\tau = (a_{ll} - a_{kk})/2a_{kl}$, $c = \cos\theta$ and $s = \sin\theta$, the last expression above becomes

$$(a_{kk} - a_{ll})cs + a_{kl}(c^2 - s^2) = 0$$
$$(a_{kk} - a_{ll})t + a_{kl}(1 - t^2) = 0$$
$$t^2 + 2\tau t - 1 = 0$$
$$\Rightarrow t = -\tau \pm \sqrt{1 + \tau^2}$$

where, in turn, the values of $\cos\theta$ and $\sin\theta$ are obtained through

$$c = \frac{1}{\sqrt{1 + t^2}} \quad \text{and} \quad s = tc$$

Note that we have two options in the choice of two values for t. To see which one is the best, we need to look back at the element definitions in the matrix $\mathbf{B}$ above. There are three expressions which describe the change in the non-diagonal matrix elements. These are for b_{ik}, b_{il} and b_{kl}, where the last is set to zero in the algorithm. After a number of similarity transforms, the elements b_{ik}, b_{il} should approach zero. If these are sufficiently close to zero, we wish not to change them, however, we do not have a choice. We can, however, make the change as small as possible by forcing $cos\theta \to 1$ and $\sin\theta \to 0$ simultaneously - a result obtained by letting the angle θ be as small as possible. Therefore, by choosing the smaller of the roots for t, we get the least difference in these elements and convergence of the method is ensured to happen faster.

When all the non-diagonal matrix elements are sufficiently close to zero, the eigenvalues of matrix $\mathbf{A}$ is listed along the diagonal.

2.2 Eigenvectors

The eigenvalues for the matrix is found using Jacobi's rotation algorithm as discussed above. The eigenvectors, however, is not returned by this method.

Recall the definition of similarity transform on the system:

$$\mathbf{B}(\mathbf{S}^T\mathbf{u}_1) = \lambda_1(\mathbf{S}^T\mathbf{u}_1)$$

so, after a similarity transform, the similar matrix $\mathbf{B}$, whose eigenvalues are the same as $\mathbf{A}$, have the eigenvector $\mathbf{S}^T\mathbf{u}_1$ where $\mathbf{u}_1$ is the eigenvector of $\mathbf{A}$ associated with the eigenvalue λ_1. If $\mathbf{B}$ was a diagonal matrix, it's eigenvector would be the vectors $\mathbf{e_i}$. In this case, we would simply compute $\mathbf{u}_i = \mathbf{S}\mathbf{e}_i$. If we start with $\mathbf{R} = \mathbf{I}$, the unit matrix, the algorithm for iteratively approaching the eigenvectors is

$$r_{ik} = r_{ik}\cos\theta - r_{il}\sin\theta$$
$$r_{il} = r_{il}\cos\theta + r_{ik}\sin\theta$$

2.3 Single electron Schrödinger equation

The radial part of the Schrödinger equation for one electron is

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

we are interested in the solution in a harmonic oscillator potential, so $V(r) = (1/2)kr^2$ where $k = m\omega^2$ and E is the energy of the harmonic oscillator in three dimensions. ω is the oscillator frequency and the energies can be found analytically:

$$E_{nl} = \hbar\omega\left(2n + l + \frac{3}{2}\right)$$

with $n = 0, 1, 2, ...$ and $l = 0, 1, 2, ..., n - 1$.

We will substitute $R(r) = (1/r)u(r)$ and rewrite eq. (1) as follows

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

Next, introduce a dimensionless variable $\rho = (1/\alpha)r$ where α is a constant with dimension length and $\rho \in [0, \infty)$. We will only be looking at solutions where $l = 0$. Inserting this and $V(\rho) = (1/2)k\alpha^2\rho^2$ we get

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

Further simplification by multiplying through by $2m\alpha^2/\hbar^2$ and fixing α so that

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

results finally in

$$-\frac{d^2}{d\rho^2}u(\rho) + \rho^2u(\rho) = \lambda u(\rho) \tag{3}$$

3

where

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

which in three dimensions can take the values $\lambda_0 = 3$, $\lambda_1 = 7$, $\lambda_2 = 11$, ...

Numerically, the second derivative in eq. (3) can be approximated using

$$\frac{d^2 u(\rho)}{d\rho^2} = \frac{u(\rho + h - 2u(\rho) + u(\rho - h)}{h^2} + O(h^2) \tag{4}$$

The variable ρ can be interpreted as a distance from the center and $u(\rho)^2$ as the probability of finding the electron at the distance ρ. As ρ increases, this probability decreases rapidly, and so, although the boundary conditions require $u(0) = 0$ and $u(\infty) = 0$, we can safely truncate ρ to a finite number, as long as $u(\rho)$ is sufficiently small. The step length can then be defined as

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

where n is the number of steps we wish to take and $\rho_i = \rho_{\text{min}} + ih$ for $i = 0, 1, 2, ..., n$.

Rewriting eq. 4 in a more compact way and inserting into eq. 3, we obtain

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

where $\rho_i^2 = V_i$ is the harmonic oscillator potential.

We can rewrite this as the matrix equation

$$\begin{pmatrix} d_1 & e_1 & 0 & 0 & \ldots & 0 & 0 \\ e_1 & d_2 & e_2 & 0 & \ldots & 0 & 0 \\ 0 & e_2 & d_3 & e_3 & 0 & & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & \ldots & \ldots & \ldots & \ldots & d_{n_{\text{step}}-2} & e_{n_{\text{step}}-1} \\ 0 & \ldots & \ldots & \ldots & \ldots & e_{n_{\text{step}}-1} & d_{n_{\text{step}}-1} \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ \ldots \\ \ldots \\ \ldots \\ u_{n_{\text{step}}-1} \end{pmatrix} = \lambda \begin{pmatrix} u_1 \\ u_2 \\ \ldots \\ \ldots \\ \ldots \\ u_{n_{\text{step}}-1} \end{pmatrix} \tag{5}$$

where the diagonal matrix elements are defines as

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

and the non-diagonal elements are

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

2.4 Two electron Schrödinger equation

Quite analogous to the one electron solution, the radial part of the Schrödinger equation for two electrons is just the sum of the single electron solution

$$\left(-\frac{\hbar^2}{2m} \frac{d^2}{dr_1^2} - \frac{\hbar^2}{2m} \frac{d^2}{dr_2^2} + \frac{1}{2} kr_1^2 + +\frac{1}{2} kr_2^2 \right) u(r_1, r_2) = E^{(2)} u_{(}r_1, r_2)$$

where $E^{(2)}$ denotes the energy for two electrons. Introducing the relative coorsinate $\mathbf{r} = \mathbf{r}_1 - \mathbf{r}_2$ and the center of mass coordinate $\mathbf{R} = (1/2)(\mathbf{r}_1 + \mathbf{r}_2)$ we can rewrite the last expression as

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

Furthermore, we can perform a separation of variables on the wavefunction letting $u(r, R) = \psi(r)\phi(R)$ and the energy is given by

$$E^{(2)} = E_r + E_R$$

The repulsive Coulomb interaction between two electrons is

$$V(r_1, r_2) = \frac{\beta e^2}{|\mathbf{r}_1 - \mathbf{r}_2|} = \frac{\beta e^2}{r}$$

where $\beta e^2 = 1.44$ eVnm. We can then simplify further, obtaining

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

which is very similar to that in eq. (2). Following a much similar approach as above, we obtain finally

$$-\frac{d^2}{d\rho^2}\psi(\rho) + \omega_r^2\rho^2\psi(\rho) + \frac{1}{\rho} = \lambda\psi(\rho) \tag{6}$$

where now $\rho = r/\alpha$, α is fixed so that

$$\frac{m\alpha\beta e^2}{\hbar^2} = 1$$

ω_r can be interpreted as a frequency of the harmonic oscillator:

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

and finally

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

Comparing eq. (3) and eq. (6), we see that they are equal, but with the potential changing from $V_i = \rho_i^2$ to $V_i = \omega_r^2\rho_i^2 + 1/\rho_i$.

2.5 Implementation

I have implemented the above methods in the C++ programming language. Note that the values for the potential in both cases are known at $i = 0$ and $i = n$, namely $V_0 = 0$ and $V_n = 0$. To create the matrix in eq. (5), we therefore only need values for ρ_i for $i = 1, 2, ..., n - 1$, which are plugged into the diagonal elements d_i.

Also, since only some elements in the matrix will change for each iteration, we need not create a new matrix, but rather change the existing matrix in place. The implemented Jacobi is

Algorithm 1: Jacobi algorithm

```
int jacobi_method(double **A, double **R, int n)
{
    int iterations = 0;
    int k, l;
    double eps = 1.0e-8;
    double max_off = max_off_diag(A, &k, &l, n);
    double max_number_iterations = (double) n * (double) n * (double) n;
    while (max_off*max_off > eps && (double) iterations < max_number_iterations) {
        rotate(A, R, k, l, n);
        max_off = max_off_diag(A, &k, &l, n);
        iterations++;
    }
    return iterations;
}
```
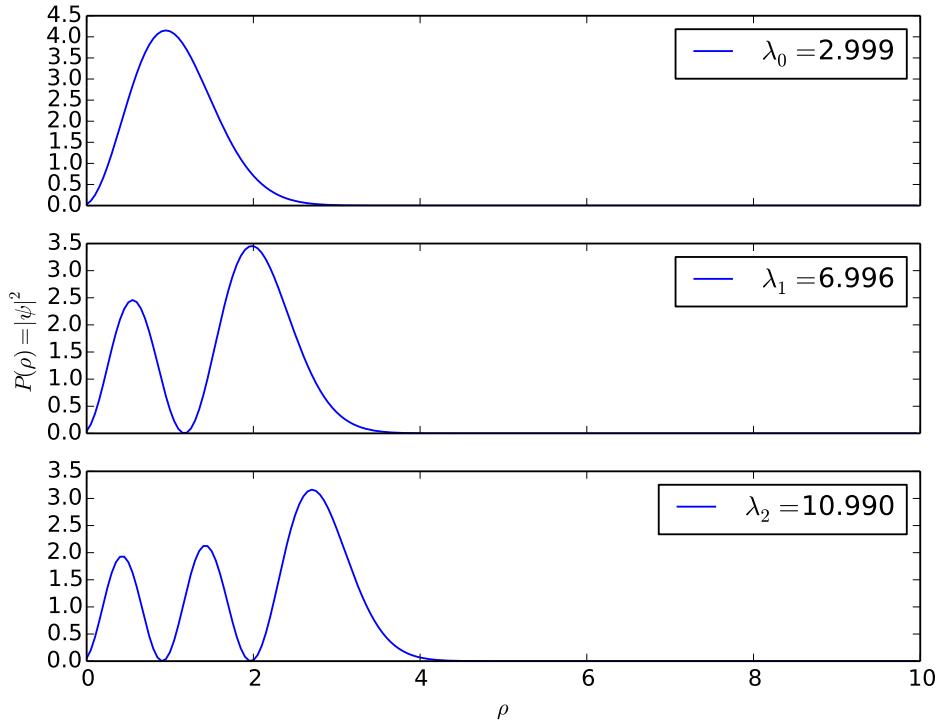
Fig. 1: Probability distribution of the eigenstates for one electron in the harmonic oscillator potential. As we can see, the eigenvalues are not completely equal the theoretical results. In this example, $n = 200$ grip points and $\rho_{\max} = 10$ were used.

where the function `rotate` follows the rotation algorithm presented in section 2.1. The function `max_off_diag` finds and returns the largest element in the current matrix $\mathbf{A}$, also changing the values of k and l so that these refer to the largest element in $\mathbf{A}$.

3 Results

3.1 One electron

Fig. 1 shows the eigenstates of the electron plotted as a probability distribution as a function of ρ. The respective eigenvalues are indicated, and resemble those found theoretically, building confidence in the reliability of the program. As previously discussed, the value of $\rho_{\max}$ should, ideally, be ∞, but as fig. 1 shows, the probability of finding the electron outside of $\rho_{\max} \approx 5$ is almost, or equal to, zero for the three lowest eigenstates. One can therefore increase the accuracy by setting $\rho_{\max}$ lower, without needing to increase the number of grid points. Table 1 shows such a relation. Note that decreasing $\rho_{\max}$ into an area where there is a larger than zero probability of finding the electron ($\rho_{\max} < \sim 4$, has the effect of decreasing the accuracy.

By the method of trial and error, it was found that $n = 220$ is enough to get the three first eigenvalues stable to four leading digits.

It is also of interest to find the number of transformations needed as a function of the number of grid points. Fig. 2 shows such a relation, and it is evident that the number of iterations needed goes like $\sim O(n^2)$, at least for $n < \sim 400$.

Table 1: The accuracy of λ increases as the density of grid points increases, here represented by decreasing values of $\rho_{\max}$. $n = 200$.

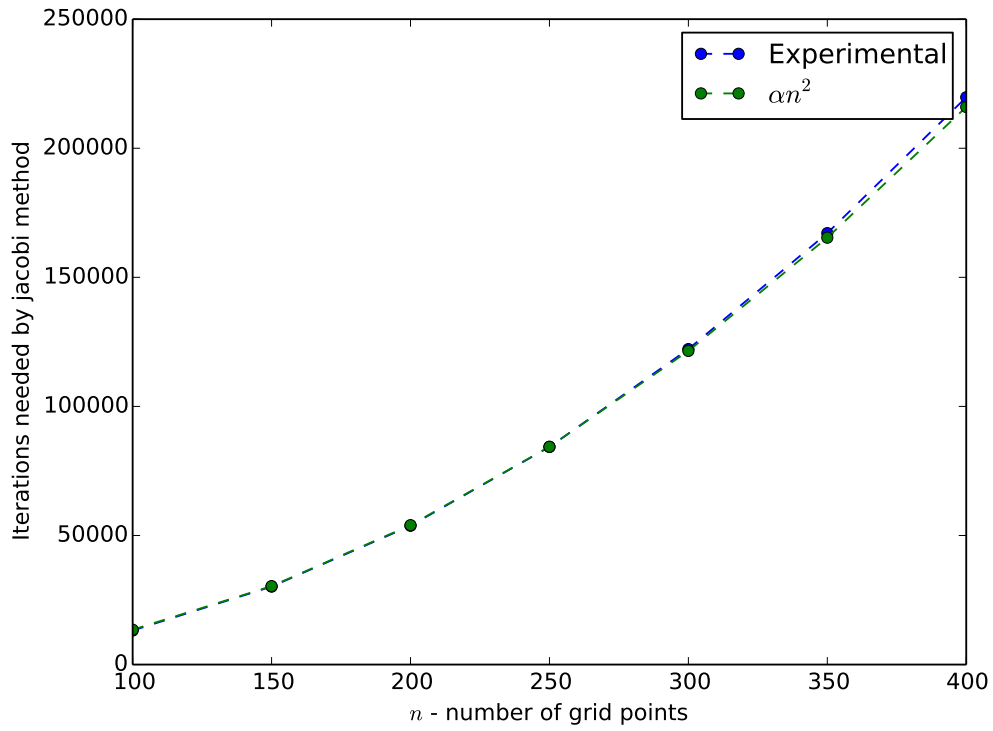| $\rho_{\max}$	λ_1	λ_2	λ_3
10	2.999	6.996	10.990
8	2.999	6.997	10.993
6	2.999	6.998	10.996
5	2.999	6.999	10.997
4	2.999	7.002	11.070

Fig. 2: The number iterations needed to reach diagonality of the matrix $\mathbf{A}$ as a function of number of grid points n.
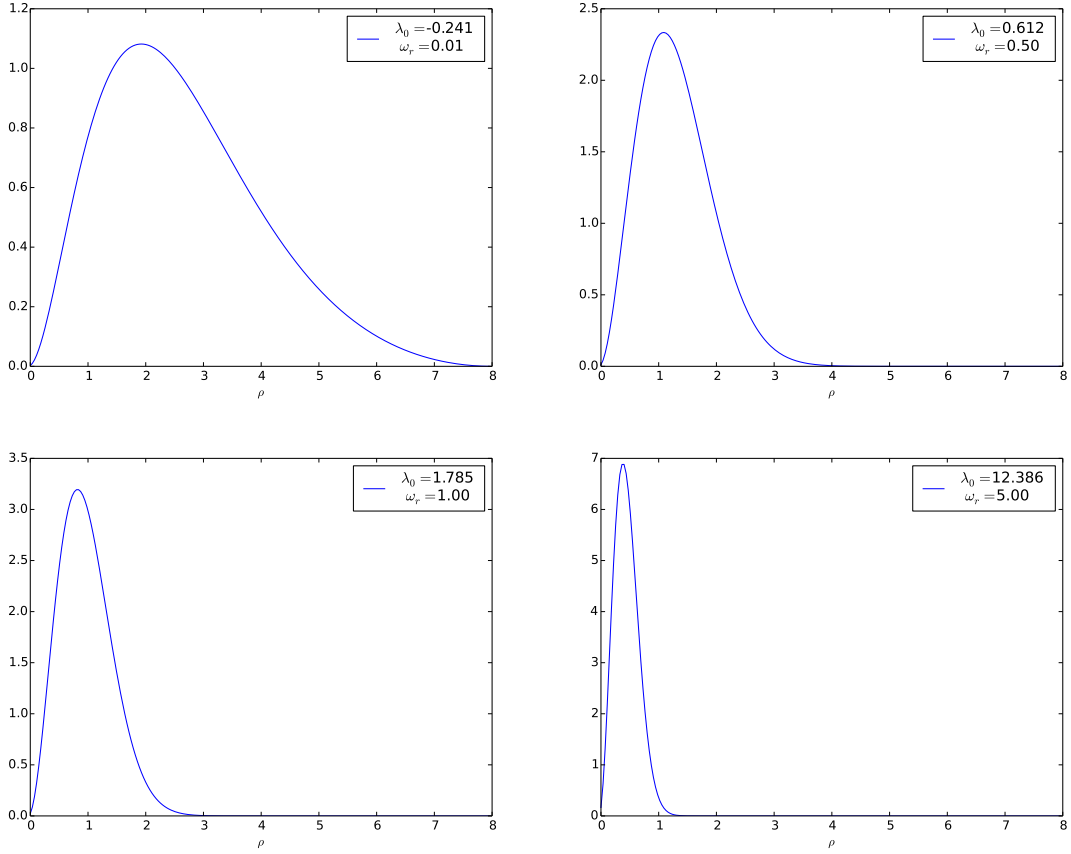
Fig. 3: Plot of the probability distributions for two electrons with four different oscillator "frequencies" ω_r.

3.2 Two electrons

For the case of two electrons, we are interested in the wavefunction for the ground state (lowest eigenvalue) for four different values of the oscillator "frequency" ω_r. The results are shown in fig. 3.

As the plot show, the oscillator frequency greatly influences the probability distribution of the position of the center of mass for the electron pair, higher frequencies leading to a narrower distribution, much closer to the origin. One other thing to note is that higher frequency leads to larger eigenvalue. This makes direct sense, as the eigenvalues $\lambda \propto E$, and higher frequencies give higher energies. Note also that for $\omega_r = 0.01$, we have $\lambda_0 < 0 \Rightarrow E_0 < 0$. What this means, I do not have the competence to answer with certainty, however, I propose that this leads to an improper eigenstates (that it is a solution to the Schrödinger equation, however not the normalization condition) and that the ground state in this case is the first to have positive energy.

3.3 Different methods

The Jacobi method for finding eigenvalues and eigenvectors is intrinsically slow. Table 2 shows the time needed to find the eigenvalues and eigenvectors of the implemented Jacobi method vs. the function `numpy.linalg.eig` in the python `numpy` library. Ideally this comparison should be done with a C++ library, e.g. armadillo, but due to a premature system update, this library is

Table 2: Computation times for the Jacobi method and an eigenvalue method from the numpy library in python averaged over 10 runs for each method for each n.

n	Jacobi [s]	Python [s]
100	0.009	0.172
150	0.025	0.903
200	0.059	2.793
250	0.132	6.564
300	0.233	13.607
350	0.367	24.354
400	0.540	40.842

not working on my computer. It is to say that the `numpy.linalg`-library relies heavily on the LAPACK and BLAS libraries, which is also a dependency on armadillo.

As table 2 shows, the method in the `numpy` library is much faster than the implemented Jacobi algorithm. The values for eigenvectors are not, however significantly different for a particular choice of n and ρ_{max}, meaning for a relatively small system as the one looked at in this project, the jacobi-method is a "good enough" method when it comes to time consummation.

4   Conclusion

The Jacobi method is a very slow method of finding eigenvalues compared to many other with computation times for a 400×400 matrix being around 40 s, compared to 0.5 s for the method embedded in the python `numpy` llibrary. On average, it is needed around $O(n^2)$ Jacobi rotations to find the eigenvalues of an $n \times n$ matrix, each rotation using around 30-40n FLOPS, leading to $\sim O(n^3)$ FLOPS.

We have seen that both methods reaches convergence for the three lowest eigenvalues relatively quickly at $n \approx 200$. A good reason to use the Jacobi method that is easily parallelizable, so for systems where a relatively small number of grid points n is sufficient, the Jacobi method may be the preferred method.