FYS3150/4150 Project 2

Kristian Gjestad Vangsnes, Tore Klungland and Laura Beghini

Abstract

We examine the Jacobi's method to solve different eigenvalues equations. We study three different problems: the buckling beam, one electron and two electrons systems. Each one can be represented by an eigenvalue equation. Although it leads to correct results, we find that the Jacobi's method is really slow compared with other method. For each problem we get the lower eigenvalues and the corresponding eigenvectors with good precision; in the buckling beam problem the analytical solutions were reproduced to machine precision, while the lowest energy eigenvalues for the quantum mechanical systems were reproduced to 3 or 4 leading digits after the decimal point. The somewhat larger error here arises since the wavefunctions in these problems have boundary conditions at infinity, which must be approximated by some large number.

All programs used to produce and plot these results, as well as output files containing the results referred to in this text, can be seen at `https://github.com/Krissvang/Computational-Physics-group`.

UiO : University of Oslo

University of Oslo
Norway
September 30, 2019

Contents

1 Introduction

Many of the most powerful equations used in physics today can be represented as eigenvalues equations. This is true both for the classical world and the quantum one. Thus, it is important to know how to solve them, not only accurately, but also in a timely manner.

This project's aim is to solve different kinds of eigenvalues equations, from the buckling beam problem to the Schrödinger's equation for two electrons in a three-dimensional harmonic oscillator well. In order to use the same code to solve different physical systems is useful to scale the equations [1].

All the problems that are solved in this project are depicted by second order differential equations that could be written in the form

$$-\frac{d^2}{d\rho^2}u(\rho)+\rho^2 u(\rho)=\lambda u(\rho)\,, \tag{1}$$

or introducing small changes.

In this text we will first discuss how the Jacobi method can be implemented to solve the buckling beam problem as well as quantum mechanical problems of one or two electrons. We will then compare the obtained results to analytical ones, and discuss how the precision changes as different parameters are varied. We will also examine the efficiency of the method.

2 Methods

In this section the mathematical theory behind the computational algorithms used is outlined. We have used the Jacobi's method to solve the eigenvalues equations. This method is described here. We will also explain how it is possible to scale a differential equation.

2.1 The Jacobi's method

Jacobis's method is an algorithm to diagonalize a matrix, i.e. to find the eigenpairs. In order to put all the non-diagonal elements to zero we use a series of unitary transformations (or similarity transformations), which give a new matrix with the same eigenvalues as the original but different eigenvectors [2]. These transformations keep the inner product in our space invariant. In this project we want to solve second order differential equations of the kind (1). These equations are represented by symmetrical Toeplitz matrices. Due to this fact, using a unitary transformation we can put to zero two non diagonal elements with one iteration.

As stated unitary transformations keep the inner product of our space invariant. The space we use is the ordinary Euclidean space with the canonical inner product over the real or complex numbers. Over the complex numbers this inner product is defined by $\langle x,y\rangle = x^{\dagger}y$ where $x^{\dagger}$ is the transpose complex conjugate of the vector x. Over the real numbers we have that $x^{\dagger} = x^T$ since $x^* = x$. Since unitary transformations keep the inner product invariant, $\langle Ux,Uy\rangle = \langle x,y\rangle \Rightarrow (Ux)^{\dagger}Uy = x^{\dagger}U^{\dagger}Uy = x^{\dagger}y$. We see that they must obey the following relation:

$$UU^{\dagger} = U^{\dagger}U = \mathbb{1} \quad \Rightarrow \quad U^{\dagger} = U^{-1}\,. \tag{2}$$

Over the real numbers we get that $U^{\dagger} = U^T$ which mean that unitary transformations correspond to orthogonal transformations.

$$UU^T = U^TU = \mathbb{1} \quad \Rightarrow \quad U^{-1} = U^T\,. \tag{3}$$

An interesting property of unitary transformation is that it preserves the orthogonality of the obtained eigenvectors. To see this we consider a basis of vectors $\{\vec{v}_i\}$

$$\vec{v}_i = \begin{bmatrix} v_{i1} \\ v_{i2} \\ \cdots \\ \cdots \\ \cdots \\ v_{in} \end{bmatrix} \quad \text{and} \quad \vec{v}_j^T\vec{v}_i = \delta_{ij}\,. \tag{4}$$

And we define

$$\vec{w}_i = U\vec{v}_i\,. \tag{5}$$

Thus by using equation (3) we get

$$\vec{w}_j^T\vec{w}_i = \vec{v}_j^T U^T U\vec{v}_i = \vec{v}_j^T\vec{v}_i = \delta_{ij}\,, \tag{6}$$

as we wanted to show.

For this project we will consider only unitary transformations that performs a plane rotation around an angle θ in the Euclidean space. This are matrices are on the form

$$U = \begin{bmatrix} 1 & 0 & \dots & 0 & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & \cos\theta & 0 & \dots & 0 & \sin\theta \\ 0 & 0 & \dots & 0 & 1 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 0 & 0 & \dots & 1 & 0 \\ 0 & 0 & \dots & -\sin\theta & 0 & \dots & 0 & \cos\theta \end{bmatrix}, \tag{7}$$

that have elements that differs from zero only in the positions

$$u_{kk} = u_{ll} = \cos\theta \quad u_{kl} = -u_{lk} = -\sin\theta \quad u_{ii} = 1 \quad i \neq k \quad i \neq l\,. \tag{8}$$

The algorithm steps to diagonalize a matrix A are this:

- Choose a tolerance ε
- Find the bigger non diagonal element and check that its module is bigger than ε

  $$|a_{kl}| = max(|a_{ij}|) > \varepsilon\,, \tag{9}$$

  where a_{ij} are the A matrix elements. If this verify is negative the algorithm stops here.
- Find the unitary transformation that puts the element a_{kl} and a_{lk} to zero. So we have to find the right rotation angle θ in order to build desired the matrix U. If we define $\tan\theta = t = s/c$ with $s = \sin\theta$ and $c = \cos\theta$ we get

  $$\cot 2\theta = \frac{a_{ll} - a_{kk}}{2a_{kl}}\,. \tag{10}$$

  From this we obtain

  $$c = \frac{1}{\sqrt{1+t^2}} \quad \text{and} \quad s = tc\,. \tag{11}$$

  for the detailed calculation see [2].
- Compute the transformation for this set of values (k,l), obtaining the new matrix

  $$A' = U(k,l,\theta)^T A U(k,l,\theta)\,. \tag{12}$$

- Continue till

$$max(|a_{ij}|) \leq \varepsilon\,. \tag{13}$$

This method makes it possible to find both the eigenvalues and eigenvectors of the matrix; if the matrix is diagonalized by N transformations of the type described above, it will be transformed to a diagonal matrix D on the form

$$D = U_N^T U_{N-1}^T \cdots U_2^T U_1^T A U_1 U_2 \cdots U_{N-1} U_N\,, \tag{14}$$

where the elements of D are its eigenvalues, and, as mentioned previously, also the eigenvalues of A. The corresponding eigenvectors can be found by using the fact that a symmetric matrix can always be written as $A = PDP^T$, where D is a diagonal matrix with the eigenvalues of A as its elements, and P has the eigenvectors of A as columns (the vector in the i'th column corresponds to the i'th diagonal element of D) [3]. Alternatively, this can be written as

$$D = P^T A P\,. \tag{15}$$

Comparing this with (14), one can see that $P = U_1 U_2 \cdots U_N$. Thus the eigenvectors are found by initially setting $P = I_{n\times n}$, and then multiplying this matrix from the right with $U(k,l,\theta)$ for each iteration.

2.2 Unit tests

Unit testing is a way to test that certain parts of the program works correctly. We use unit tests mainly to test the mathematical properties of our algorithms. They are very useful during the development phase of the algorithm because they make searching for mistakes much easier. For this project we have tested:

- the eigenvectors orthogonality;
- the eigenvalues magnitude compare with the expected magnitude
- the section of the code that finds the maximum element of the matrix A.

2.3 The buckling beam problem

The first problem, which will partially serve as a test that the algorithm is working correctly in later problems, is finding the vertical displacement u of a beam of length L, where both ends are fastened at the same height (defined as zero), as a function of $x \in [0,L]$. This displacement is given by

$$\gamma \frac{d^2u(x)}{dx^2} = -Fu(x)\,, \tag{16}$$

(with boundary conditions $u(0) = u(L) = 0$) where γ and F are parameters given by the properties of the beam and the magnitude of the force working on in, respectively [4]. Here L and F are assumed to be known, while γ is to be determined when solving the equation. This equation can be brought to a dimensionless form by defining $\rho \equiv x/L \in [0,1]$, which gives

$$-\frac{d^2u(\rho)}{d\rho^2} = \lambda u(\rho)\,, \tag{17}$$

where $\lambda \equiv FL^2/\gamma$ [4]. In order for this equation to be solvable numerically it must be discretized; this is done by dividing the interval $\rho \in [0,1]$ into discrete points x_i ($i = 0,1,\ldots,n+1$), where $x_0 = 0$ and $x_{n+1} = 1$. With a step size $h = x_{i+1} - x_i = 1/(n+1)$, the second derivative can be approximated by [2]

$$u_i'' \approx \frac{u_{i-1} - 2u_i + u_{i+1}}{h^2}, \tag{18}$$

(where $u_i = u(x_i)$ etc.) with an error of order h^2. Inserting this into (17) gives

$$au_{i-1} + du_i + au_{i+1} = \lambda u_i, \tag{19}$$

for $i = 1,2,\ldots,n$. Here $a \equiv -1/h^2$ and $d \equiv 2/h^2$. This can be written on matrix form as the eigenvalue equation

$$A_0\vec{u} = \lambda\vec{u}, \tag{20}$$

where A_0 is an $n \times n$ tridiagonal matrix with d as its diagonal elements and a as its non-diagonal elements, and $\vec{u}^T \equiv [u_1, u_2, \ldots, u_n]$. Thus we have reformulated the beam problem as an eigenvalue problem, which can be solved as described above using Jacobi's method. The eigenvalues of this matrix are given analytically by [4]

$$\lambda_j = d + 2a\cos\left(\frac{j\pi}{n+1}\right). \tag{21}$$

We compared the eigenvalues from the Jacobi method to these analytical ones, as well as those obtained by using the `eigsym` function in Armadillo [5][6].

2.4 One electron problem

For a particle in a spherically symmetric potential $V(r)$, the radial component $R(r)$ of the wavefunction is given by the Shrödinger equation [7]

$$-\frac{\hbar^2}{2m}\left[\frac{1}{r^2}\frac{d}{dr}\left(r^2\frac{dR}{dr}\right) - \frac{l(l+1)}{r^2}\right]R + V(r)R = ER, \tag{22}$$

where E is the particle's energy and $\hbar^2 l(l+1)$ is its orbital angular momentum squared. Here the only boundary condition is $R(\infty) = 0$ (in order for the wavefunction to be normalizable). This is simplified by making the substitution to $u(r) \equiv rR(r)$; the equation then becomes [7]

$$-\frac{\hbar^2}{2m}\frac{d^2u}{dr^2} + \left[V(r) + \frac{\hbar^2}{2m}\frac{l(l+1)}{r^2}\right]u = Eu. \tag{23}$$

The boundary conditions on u are $u(0) = u(\infty) = 0$. In the present case we want to solve this equation for an electron in a harmonic oscillator potential $V(r) = \frac{1}{2}m\omega^2r^2$, with $l = 0$. The analytical energy eigenvalues are given by [4]

$$E_n = \hbar\omega\left(2n + \frac{3}{2}\right). \tag{24}$$

Scaling the equation in the same way as previously, by introducing a dimensionless variable $\rho \equiv r/\alpha$ (here α is a, for the moment, undetermined length parameter), which gives [4]

$$-\frac{d^2u(\rho)}{d\rho^2} + \frac{m^2\omega^2}{\hbar^2}\alpha^4\rho^2u(\rho) = \frac{2m\alpha^2}{\hbar^2}Eu(\rho). \tag{25}$$

Now we define $\alpha \equiv \sqrt{\frac{\hbar}{m\omega}}$ and $\lambda \equiv \frac{2m\alpha^2}{\hbar^2}E = \frac{E}{\hbar\omega/2}$, so that the equation becomes

$$-\frac{d^2u(\rho)}{d\rho^2} + \rho^2 u(\rho) = \lambda u(\rho). \tag{26}$$

Discretizing this equation is done similarly as for the previous problem, with one important difference; here we would ideally want $\rho \in [0,\infty]$, as the endpoints of this interval are where u has boundary conditions. However this is not possible numerically, so one must instead "approximate infinity" by setting $u(\rho_{\max}) = 0$ instead of $u(\infty) = 0$. u is then defined on the interval $[0,\rho_{\max}]$, which is divided into $n+2$ points as before, where now $\rho_{n+1} = \rho_{\max}$ and the step size $h = \rho_{\max}/(n+1)$. The discretized equation is then (using the same approximation for the second derivative as before)

$$au_{i-1} + (d+\rho_i)u_i + au_{i+1} = \lambda u_i\,, \tag{27}$$

where a an d are as defined previously. On matrix form this is

$$A_1\vec{u} = \lambda\vec{u}\,, \tag{28}$$

where $A_1 \equiv A_0 + \mathrm{diag}\left(\rho_1^2,\rho_2^2,\ldots,\rho_n^2\right)$. This problem can thus be solved exactly like the buckling beam problem, if ρ_i^2 is added to the i'th diagonal element. From the analytical energies in (24) one can see that the analytical eigenvalues are $\lambda = 3,7,11,\ldots$ (since $\lambda = \frac{E}{\hbar\omega/2}$).

2.5 Two electrons problem

The Shrödinger equation for two electrons with no repulsive Coulomb interaction is

$$\left(-\frac{\hbar^2}{2m}\frac{d^2}{dr_1^2} - \frac{\hbar^2}{2m}\frac{d^2}{dr_2^2} + \frac{1}{2}kr_1^2 + \frac{1}{2}kr_2^2\right)u(r_1,r_2) = E^{(2)}u(r_1,r_2). \tag{29}$$

Where we now have a two-electron wave function and energy. The wave function is separable if we neglect the Coulomb potential which depends on the distance between the two electrons. In order to get the equation on the form we need we introduce the relative coordinate $\mathbf{r} = \mathbf{r}_1 - \mathbf{r}_2$ and the center-of-mass coordinate $\mathbf{R} = 1/2(\mathbf{r}_1 + \mathbf{r}_2)$. The radial Schröedinger equation then reads

$$\left(-\frac{\hbar^2}{m}\frac{d^2}{dr^2} - \frac{\hbar^2}{4m}\frac{d^2}{dR^2} + \frac{1}{4}kr^2 + kR^2\right)u(r,R) = E^{(2)}u(r,R). \tag{30}$$

Assuming that u is seperable, we set $u(r,R) = \psi(r)\phi(R)$ and the energy is given by the sum of the relative energy E_r and the center-of-mass energy E_R, that is

$$E^{(2)} = E_r + E_R. \tag{31}$$

We add the repulsive Coulomb interaction

$$V(r_1,r_2) = \frac{\beta e^2}{|\mathbf{r}_1 - \mathbf{r}_2|} = \frac{\beta e^2}{r}, \tag{32}$$

with $\beta e^2 = 1.44$ eVnm.

Adding this term, the r-dependent Schrdinger equation becomes

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

Repeating the same steps as with the one electron problem, we arrive at

$$-\frac{d^2}{d\rho^2}\psi(\rho) + \frac{1}{4}\frac{mk}{\hbar^2}\alpha^4\rho^2\psi(\rho) + \frac{m\alpha\beta e^2}{\rho\hbar^2}\psi(\rho) = \frac{m\alpha^2}{\hbar^2}E_r\psi(\rho). \tag{33}$$

We define a new 'frequency'

$$\omega_r^2 = \frac{1}{4}\frac{mk}{\hbar^2}\alpha^4, \tag{34}$$

and fix the constant α by requiring

$$\frac{m\alpha\beta e^2}{\hbar^2} = 1 \tag{35}$$

Defining

$$\lambda = \frac{m\alpha^2}{\hbar^2}E. \tag{36}$$

We then get the equation

$$-\frac{d^2}{d\rho^2}\psi(\rho) + \omega_r^2\rho^2\psi(\rho) + \frac{1}{\rho} = \lambda\psi(\rho). \tag{37}$$

We treat ω_r as a parameter which reflects the strength of the oscillator potential and will show later why it makes sense to call it a "frequency". [4]

This equation is discretized like the one-electron one; this gives the matrix equation $A_2\vec{u} = \lambda\vec{u}$ where $A_2 = A_0 + \text{diag}\left(\omega_r^2\rho_1^2 + 1/\rho_1, \omega_r^2\rho_2^2 + 1/\rho_2, \ldots, \omega_r^2\rho_n^2 + 1/\rho_n\right)$. This equation is solved like the previous ones using the Jacobi method, adding the extra terms, $\frac{1}{\rho}$, to the diagonal.

3 Implementation

The programs used in this project is programmed using c++ and are called "beam.cpp", "qho_no_int.cpp" and "qho_int.cpp", corresponding to the buckling beam problem, the one electron problem and the two

electron problem. We also have a file called ”One_electron.cpp”, which is used to find the eigenvalues to the one electron problem with a precision of four leading digits, and a file called ”count.cpp”, which gives the number of transformations and the time it used for different resolutions. All the programs uses the Jacobi algorithm to solve the eigenvalue problems, such procedures are written in a separate file called ”jacobi.cpp”, and is called in the different programs through its header file.

We have two python files which creates plots. The file ”plot.py” uses the configurations in the configuration file ”config.json”, runs the specified algorithm and plots the solutions. For the buckling beam problem it plots the three first eigenvectors (solutions) as a function of $\rho = x/L$, for the quantum harmonic oscillator problems it plots the ground state. If for example one would want to plot the buckling beam solution with a resolution of $n = 300$, one would go into the ”config.json”, write ”beam” as the algorithm (”alg”), 300 as the resolution (”res”), and write ”yes” for ”plot_solution”, which determines if the program will plot. Then run ”python3 plot.py” in the command line. The program will then compile and run the ”beam.cpp” program and create the plot. The program which plots the differences between the computed and the theoretical eigenvalues is called ”One_electron.py”.

For unit testing we have used the CATCH library for c++. The file ”test-functions.cpp” contains the unit tests. We have three unit tests, one which tests the max value of the matrix in the one electron problem with resolution $n = 3$. The second test checks if the eigenvalues for the matrix is correct and the third test checks if the orthogonality of the eigenvectors is left invariant under the unitary transformations as mentioned in the section on the Jacobi method. We can also run the unit tests from the python file by putting the configuration variable ”run_unit_tests” to ”yes”.

4 Results and discussion

The results obtained for are discussed in this section. First of all we will show the buckling beam problem results. After that we will display the one electron and two electron problems results.

4.1 The buckling beam problem

We solved the beam problem as described previously for a number of different matrix dimensions n, with a tolerance of $\varepsilon = 10^{-16}$. The first three eigenvectors (i.e. the first three solutions to the differential equation) are plotted, for $n = 500$, in figure 1. The shape of these solutions are precisely as one would expect from (17); they are sinusoidal functions, which is natural as (17) is a harmonic oscillator equation, which satisfy the boundary conditions $u(0) = u(1) = 0$, with an increasing number of nodes as λ increases.

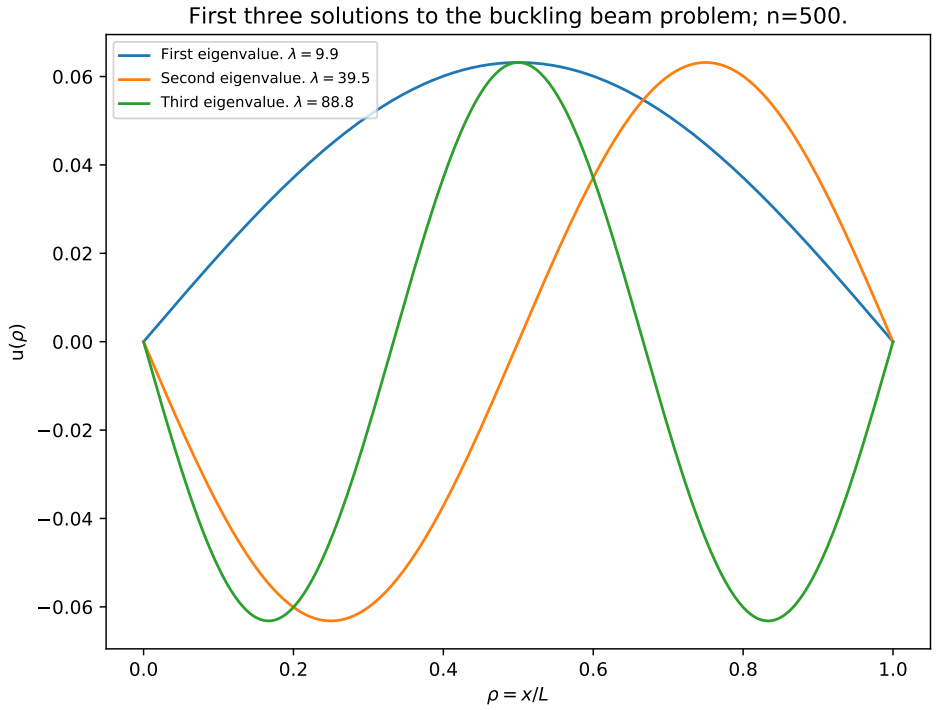

Fig. 1. The first three solutions to the buckling beam problem, solved using a 500×500 matrix.

The lowest computed eigenvalues from both Armadillo and the Jacobi method are compared, for different values of n, to the analytical eigenvalue, in table 1. The CPU time used for the two methods are also compared. These results show that both methods are very precise, as they are able to reproduce the analytical results essentially to machine precision (at least to eight leading digits after the decimal point, which was the precision we used for all the output). This demonstrates that the Jacobi method as implemented here is able to find the eigenvalues of a tridiagonal matrix to high precision.

Table 1. Comparison of the CPU time used in obtaining the eigenvalues with the Jacobi algorithm and Armadillo's `eigsym` function. The lowest computed eigenvalues for both methods are also shown, and compared to the analytical result obtained from (21).

	Jacobi		Armadillo		Analytic
n	t [s]	λ_1	t [s]	λ_1	λ_1
20	8.18e-4	9.85121127	1.95e-3	9.85121127	9.85121127
50	0.0319	9.86648391	5.20e-3	9.86648391	9.86648391
100	0.381	9.86880868	3.94e-3	9.86880868	9.86880868
200	6.05	9.86940348	5.24e-3	9.86940348	9.86940348
300	28.32	9.86951481	0.0148	9.86951481	9.86951481
500	249.18	9.86957206	0.0508	9.86957206	9.86957206

However, one can also see that it is highly inefficient; for low n the CPU time is similar for the Jacobi and Armadillo methods, but as n increases the Jacobi method becomes significantly slower. To examine this point further, we plotted the logarithms of both the CPU time and the number of iterations for the Jacobi method against the logarithm of n, for n between 2 and 200. This is shown in figure 2, and from the slopes of the plots one can see that the number of iterations is roughly proportional to n^2, and the CPU time is proportional to somewhere between n^3 and n^4. The high CPU times for the Jacobi method is thus mostly due to the fact that it takes a very large number of transformations to diagonalize the matrix, and since about $\mathcal{O}(n)$ floating-point operations are needed for each transformation (since the n-dependence of the CPU time is about one order higher than that of the number of iterations), the method is rather slow.

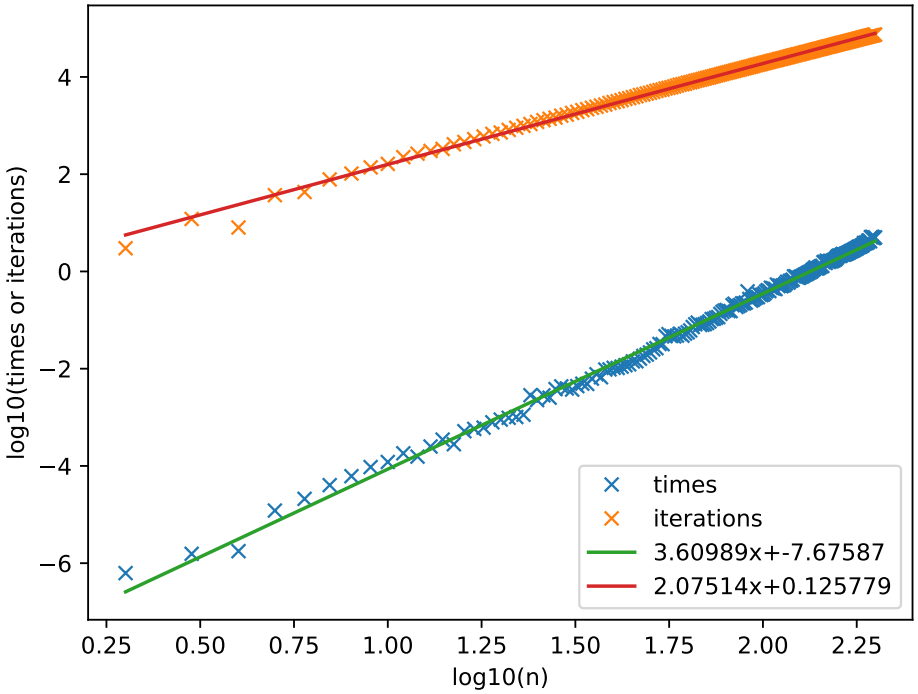

Fig. 2. Logarithms of both the CPU time and the number of iterations for the Jacobi method against the logarithm of n, for the buckling beam problem. Straight lines are fitted to both data sets.

The high number of transformations needed to diagonalize the matrix is not surprising if one considers how the Jacobi method works; for each iteration the transformation matrix U is chosen so that the largest non-diagonal matrix element in A is set to zero. However, this element does not stay equal to zero; in the next iteration the transformation matrix is chosen in order to set another non-diagonal element to zero, but the element which was set to zero in the previous iteration will then be changed. Thus the transformation must be performed several times for each non-diagonal matrix element before all of them are sufficiently close to zero.

4.2 One electron problem

The aim of this part of the project is to find the first three eigenvectors and eigenvalues of the equation (22). Firstly we need to understand dependence of the eigenvectors and eigenvalues on the parameters ρ_{max} and n. This is useful in order to find the best values of n and ρ_{max} to solve the problem. Thus the first result we need is a plot of the difference between the computed eigenvalues and the expected ones, computed for different values of ρ_{max} and fixed n. This is shown in Figure 3.

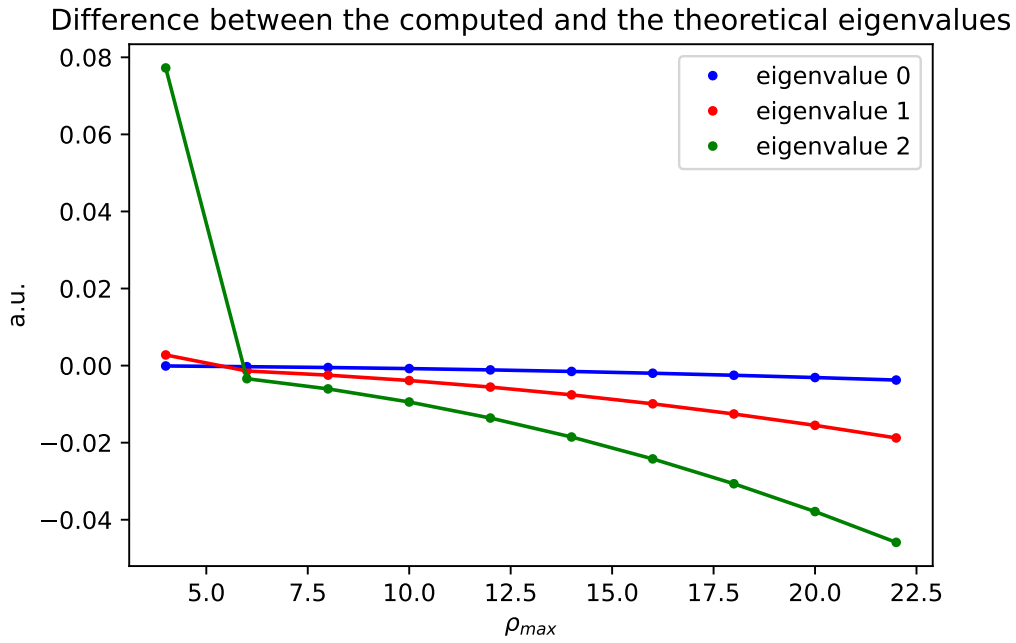

Fig. 3. Plot of the difference between the computed eigenvalues and the expected ones (expressed in arbitrary units), for different values of ρ_{max} and fixed $n = 200$. We note that for $\rho_{max} < 6$ some problems might occur, and the difference increases (especially for the third eigenvalue). This behaviour is due to the fact that the corresponding wave function is the most extended, and therefore it is the most affected by the upper limit of the boundary conditions.

In Figure 3 we note that, when ρ is around 6, we get the best evaluation of the eigenvalues. However, this value of ρ_{max} changes when n becomes bigger. We can see this in Figure 4, where we show the same plot as in Figure 3, but using $n = 300$ instead of $n = 200$.

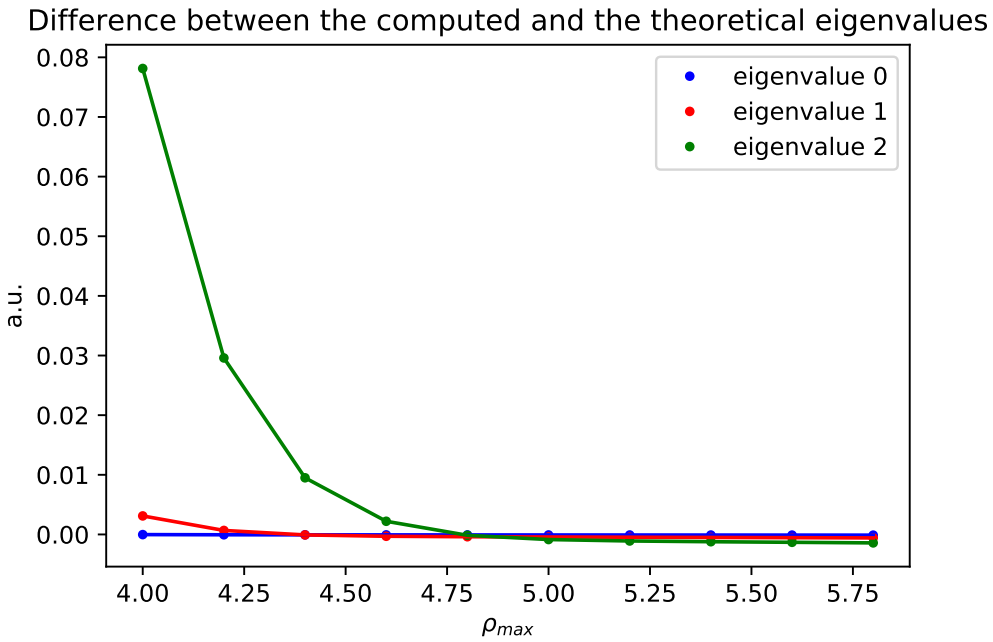

Fig. 4. Plot of the difference between the computed eigenvalues and the expected ones (expressed in arbitrary units), for different values of ρ_{max} and fixed $n = 300$. We note that for $\rho_{max} < 4.8$ some problems might occur, and the difference increases (especially for the third eigenvalue). This behaviour is due to the fact that the corresponding wave function in the most extended and therefore the most affected by the upper limit of the boundary conditions.

From the results above we know that the best value of ρ_{max} suitable for finding the first three eigenvalues is $\rho_{max} \in [4.8, 5.1]$. In fact, these are the smaller values of ρ_{max} which avoids numerical problems.

The next step is to find the best value of n for our calculation. It is useful to investigate the n dependence of the difference between the computed and the expected eigenvalues. In Figure 5 we show this difference.

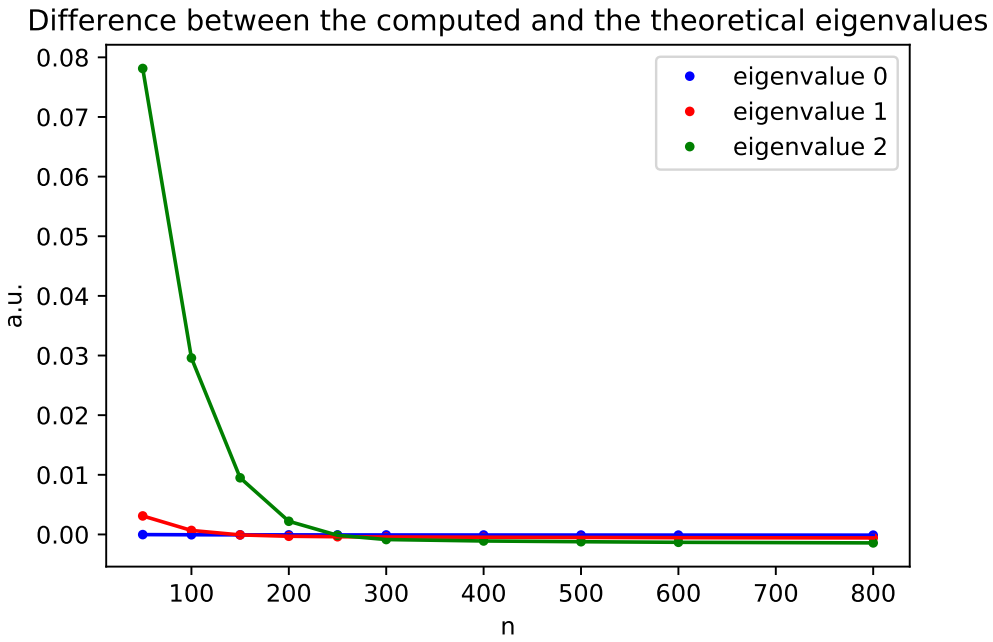

Fig. 5. Plot of the difference between the computed eigenvalues and the expected ones (expressed in arbitrary units), for different values of n. In this plot ρ_{max} is fixed as the mean value in the interval previously found, i.e. $\rho_{max} = 4.95$. We note that for $n < 250$ some problems occurs, and the difference increases (especially for the third eigenvalue). This is due to the fact that the corresponding wave function in the most extended, and therefore the most affected by the upper limit of the boundary conditions.

The data for $n \in [250, 400]$ are not precise enough to give satisfactory results. However as n increases the difference decreases. The correct values of n and ρ_{max}, in order to get the first three eigenvalues exact up to four leading digits after the decimal point, are $n = 910$ and $\rho_{max} = 5.03$. The eigenvalues results are shown in Table 2.

Table 2. Results for the first three eigenvalues of equation (22), using $n = 910$ and $\rho_{max} = 5.03$.

	Computed result [a.u.]	**Rounded computed value [a.u.]**	**Expected value [a.u.]**
Eigenvalue 0	2.999990	3.0000	3
Eigenvalue 1	6.999954	7.0000	7
Eigenvalue 2	11.00004	11.0000	11

The corresponding eigenvectors are shown in Figure 6.

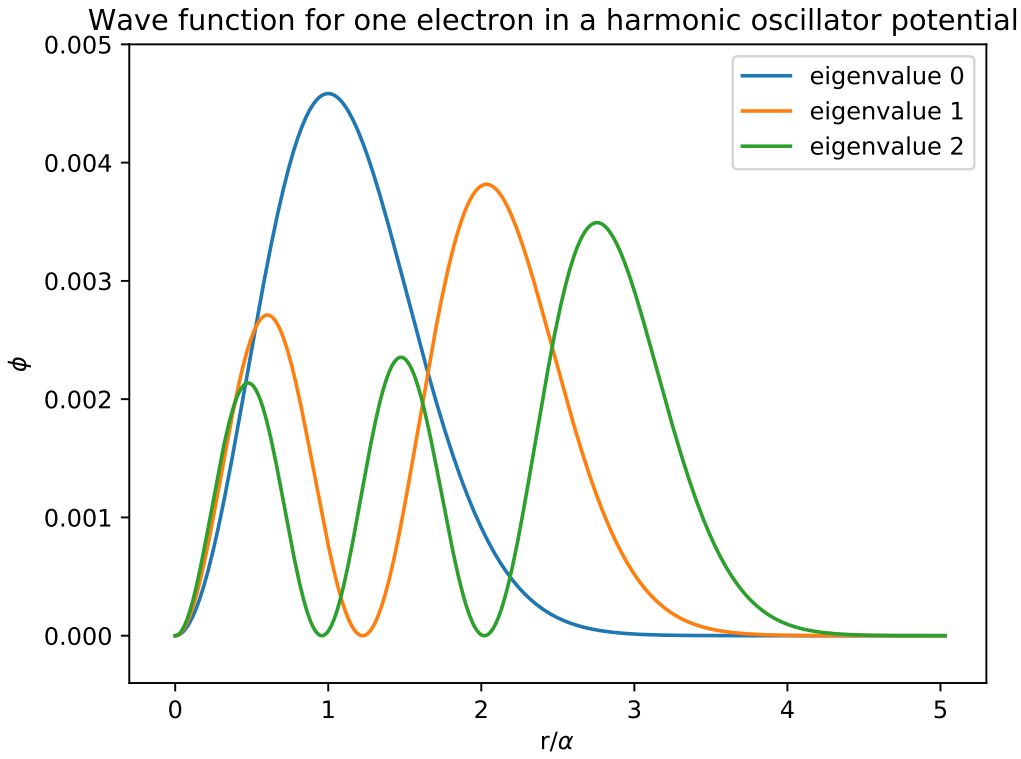

Fig. 6. Plot of the wave functions corresponding to the first three eigenvalues of equation (2) as functions of the normalized radius r/α. This wave function are computed with $n = 910$ and $\rho_{max} = 5.03$. We see that the ground state has no node, while the first and second excited states have respectively one and two nodes.

4.3 Two electrons problem

We used the code used for the one electron problem and added the Coulomb potential with a tolerance $\varepsilon = 10^{-16}$. Since the two electron problem has analytical solutions, we will compare these to our numerical solutions. We find the analytical solution in M.Taut's article "Two electrons in an external oscillator potential: Particular analytic solutions of a Coulomb correlation problem" [8]. When we ran our algorithm with resolution $n = 300$, max radius $\rho_{max} = 10$ and a frequency $\omega_r = \frac{1}{4}$ we got the eigenvalue (energy) $\lambda_1 = 1.24997869$, when we compare it to the analytical eigenvalue which is $\lambda_{analytical} = 1,25$ we see that it is correct up to four leading digits. When we ran the algorithm with the same resolution, max radius $\rho_{max} = 25$ and $\omega_r = \frac{1}{20}$ we got than $\lambda_1 = 3.499955$, the analytical solution is $\lambda_{analytical} = 3.5$, we see the numerical solution is correct up to four leading digits.

When we plot the wave functions (eigenvectors) to our eigenvalues as a function of $\rho = r/\alpha$ we see that the wavelength is inversely proportional to the frequency ω_r, which is why we can think of ω_r as a frequency, see figure 7 and 8. The reason the wavelength is bigger is because for $\omega_r << 1$ we have that the Coulomb potential is dominating, which has infinite range. As ω_r becomes bigger, the harmonic oscillator potential increases which confines the wave function in a smaller area.

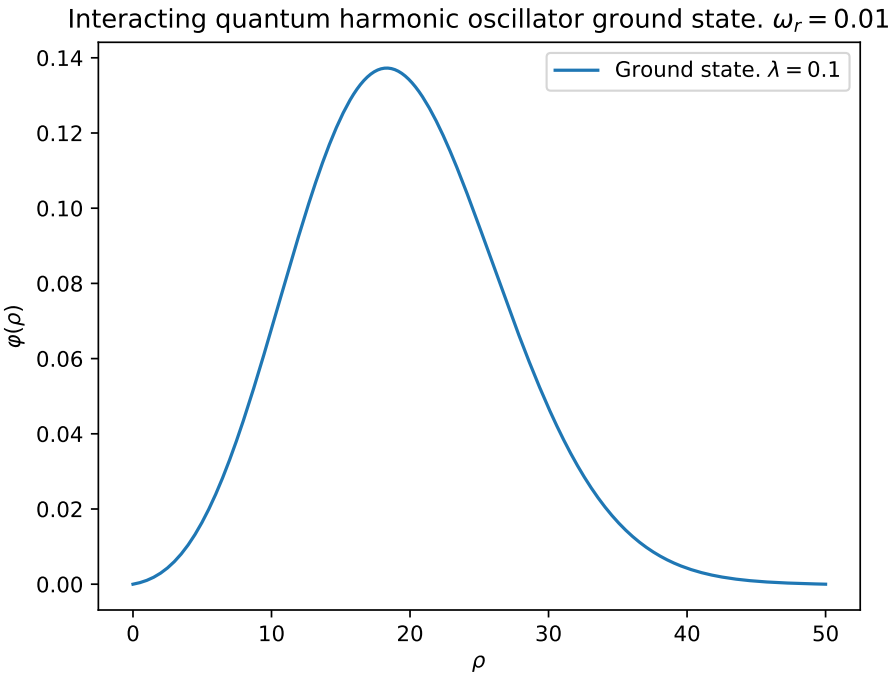

Fig. 7. Ground state solution for two electrons in a harmonic oscillator potential with frequency $\omega_r = 0.01$

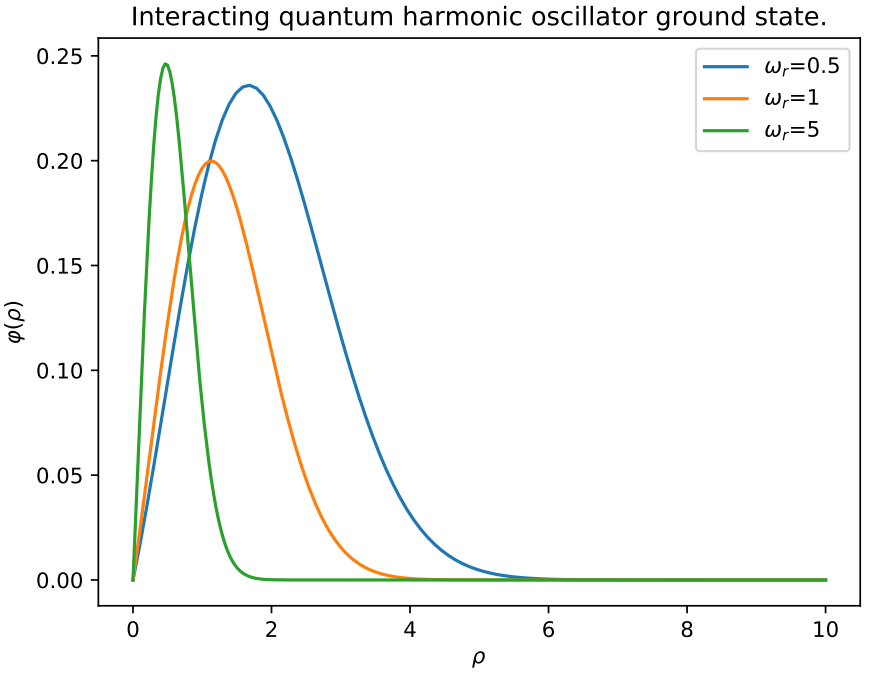

Fig. 8. Ground state solution for two electrons in a harmonic oscillator potential with frequency $\omega_r = 0.5, 1$ and 5.

5 Conclusions

As is seen in the results for the buckling beam problem, the Jacobi method is able to find the analytical eigenvalues of a tridiagonal matrix essentially to machine precision. For the quantum mechanical problems (both with one and two electrons), however, the method was only able to reproduce the analytical results up to 3 or 4 leading digits after the decimal sign. As discussed previously this is due to an extra difficulty in these problems that was not present in the buckling beam problem, namely that the wavefunction has boundary conditions at infinity. As this is impossible to represent on a computer we had to approximate infinity with a large number instead, which is an extra source of errors, especially for high-energy states which are spread out over a large interval.

We have also discussed the efficiency of the Jacobi method compared to the `eigsym` function from the Armadillo library. Here the Jacobi method was shown to be significantly slower, due to the high number of transformations required to bring all non-diagonal elements sufficiently close to zero. Hence, for computations which require high precision (i.e. small step sizes h, which means large matrices) the Jacobi method is not very practical to use. This is especially true for matrices as simple as the ones we have dealt with here, for which there are specialized methods which are much more efficient, this would be a good topic for future studies and improvements. For more general dense matrices, however, the Jacobi method, which is also applicable for more complicated matrices, might be more useful.

References

[1] Geir K. Pedersen. *Scaling of differential equations*. Springer International Publish, 2016.

[2] Morten Hjorth-Jensen. *Computational Physics Lecture Notes Fall 2015*.

[3] David C. Lay, Steven R. Lay, and Judi J. McDonald. *Linear Algebra and its Applications, 5th edition*. Pearson, 2016.

[4] *Assignment text for project 2 in FYS3150/FYS4150*. Department of Physics, University of Oslo, Norway.

[5] Conrad Sanderson and Ryan Curtin. "Armadillo: a template-based C++ library for linear algebra". In: *Journal of Open Source Software* 1 (2016), p. 26.

[6] Conrad Sanderson and Ryan Curtin. "Practical Sparse Matrices in C++ with Hybrid Storage and Template-Based Expression Optimisation". In: *Mathematical and Computational Applications* 24.3 (2019).

[7] David J. Griffiths. *Introduction to Quantum Mechanics, 2nd edition*. Cambridge University Press, 2017.

[8] M. Taut. "Two electrons in an external oscillator potential: Particular analytic solutions of a Coulomb correlation problem". In: *Physical Review A* 48.5 (Jan. 1993), pp. 3561–3566. DOI: 10.1103/physreva.48.3561.