FYS4150 - Project 2

Kristoffer Langstad
krilangs@uio.no

Abstract

In this project we want to solve eigenvalue problems of a tridiagonal Töplitz matrix using Jacobi's method. We will use unit tests to check our results with other solvers like Python's numpy and analytical results. We will solve different physical problems by discretization and scaling to make them into similar differential equations that we solve numerically. The first problem is a classical wave function problem in one dimension as a two-point boundary value problem of a buckling beam. This eigenvalue problem has analytical solutions which we will use as a unit test against our numerical algorithm. We also verify that the eigenvectors retain their orthogonality after the transformations. Both these tests passed and gave similar eigenvalues within a small tolerance of 1e-8. Next we look at a quantum mechanical problem for an electron in a 3-dimensional harmonic oscillator (quantum dots). This case also have analytical eigenvalues which we test against. For our numerical simulation we get eigenvalues $\lambda = [2.9998, 6.9990, 10.9976]$ for the first three eigenvalues, with 206'466 transformations for a 400×400 matrix and a $\rho_{max} = 10$. For the last case we expand for two electrons which interact via a repulsive Coulomb interaction. Here we test for different frequency strengths $\omega_r = [0.01, 0.1, 1, 5]$ for the ground state. We find our numerical eigenvalues as $\lambda = [0.3116, 2.2231, 4.0577, 17.4436]$ for a 400×400 matrix and $\rho_{max} = 10$, for their respective ω_r. So here we see that another electron increases the energy levels, and that it increases with increasing potential strength ω_r. Also when increasing the potential strength we get that the potential well width decrease with its peak moving closer to the center of the well.

1 Introduction

A hot topic in todays physics is what we call quantum dots where we look at electrons in small areas in semiconductors. To be able to solve these problems we have to look at differential equations. These are often difficult to solve analytically, and have to be solved numerically. Even then we have to do approximations to be able to make the equations easier to solve.

In our case we will use methods from linear algebra and look at eigenvalue problems. These are widely used to solve physical problems, so to understand how they work and how they are solved is important. Another powerful tool is to scale the equations. This means that we can solve an equation for one case, and then scale the equation to be able to solve it for another. This is one of the steps that we do in this project where we start with the wave equation for a buckling beam in one dimension, and then expand for quantum mechanics cases with the Schrödinger equation for one and two electrons (quantum dots) in three dimensions.

In the methods section we look at the theory we use and the implementation of the algorithms. We start with the buckling beam which is fastened in both ends, such that we know the endpoints with Dirichlet boundary conditions. The differential equations is simplified (scaled) with the introduction of a dimensionless variable ρ, such that we can discretize the new equation. This is now an eigenvalue problem that we will solve by solving a tridiagonal Töplitz matrix with Jacobi's method numerically. We then derive the algorithms to be used, and implement them and unit tests into a our program. Then we expand to quantum mechanics with the radial Schrödinger equation for one and two electrons in a three-dimensional harmonic oscillator potential. These equations are then manipulated and scaled to the form as the discretized buckling beam problem, but with other potential terms. For these cases we have analytical results to compare with. So the results of the eigenvalues are then compared to the analytical ones. We also look at the number of similarity transformations needed as a result of the matrix dimensionality and the maximum value of ρ for the Jacobi method algorithm. Lastly we come up with a conclusion to the project.

2 Methods

2.1 Unitary transformation

Consider a basis of vectors

$$\mathbf{v}_i = \begin{bmatrix} v_{i,1} \\ \vdots \\ v_{i,n} \end{bmatrix}$$

which is orthogonal. This means that $\mathbf{v}_j^T \mathbf{v}_i = \delta_{ij}$. Then we define a unitary matrix $\mathbf{U}$ such that $\mathbf{U}\mathbf{U}^T = \mathcal{I}$ (identity matrix). A unitary transformation of the basis vector $\mathbf{v}_i$ is then:

$$\mathbf{w}_i = \mathbf{U}\mathbf{v}_i$$

Then we take:

$$\mathbf{w}_j^T \mathbf{w}_i = (\mathbf{U}\mathbf{v}_j)^T \mathbf{U}\mathbf{v}_i = \mathbf{v}_j^T \mathbf{v}_i \mathbf{U}^T \mathbf{U} = \mathbf{v}_j^T \mathbf{v}_i = \delta_{ij}$$

This is clearly orthogonal, which means that a unitary transformation preserves both the orthogonality and the dot product of the basis vector.

We can also check that the eigenvalues are preserved under a unitary transformation as seen in Hjorth-Jensen (1)[1]:
Start with the eigenvalue problem and a similarity transformed matrix $\mathbf{B}$.

$$\mathbf{Ax} = \lambda\mathbf{x} \quad \text{and} \quad \mathbf{B} = \mathbf{U}^T\mathbf{AU} \tag{1}$$

Multiply with $\mathbf{U}^T$ and use that $\mathbf{U}^T\mathbf{U}$=$\mathcal{I}$:

$$\begin{aligned} \rightarrow (\mathbf{U}^T\mathbf{AU})(\mathbf{U}^T\mathbf{x}) &= \lambda\mathbf{U}^T\mathbf{x} \\ \Rightarrow \mathbf{B}(\mathbf{U}^T\mathbf{x}) &= \lambda(\mathbf{U}^T\mathbf{x}) \end{aligned}$$

Here we see that the eigenvalues (λ) are the same, but the eigenvectors have changed.

2.2 Buckling Beam

First we have the classical wave equation in one dimension for the buckling beam problem with Dirichlet boundary conditions:

$$\gamma\frac{d^2u(x)}{dx^2} = -Fu(x), \quad x \in [0, L], \quad u(0) = u(L) = 0 \tag{2}$$

γ is a constant depending on properties of the beam with length L, $u(x)$ is the vertical displacement of the beam in y-direction and F is the force acting on the beam. Then we introduce a dimensionless variable

$$\rho = \frac{x}{L}, \quad \rho \in [0, 1],$$

into the differential equation 2. The new differential equation then looks like this:

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

This we discretize to an eigenvalue problem by using the second derivative of the function u as

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

where h is the step length defined, from the minimum value $\rho_{min} = \rho_0$ and maximum value $\rho_{max} = \rho_N$ for a given number of mesh points N, as

$$h = \frac{\rho_N - \rho_0}{N}.$$

[1]Computational Physics (2015) - Chapter 7.3 Similarity transformations

For a value of ρ at a point i we have $\rho_i = \rho_0 + ih$ with $i = 1, 2, \cdots, N$. The differential equation 3 can now be rewritten with the second derivative u'', ρ_i and $u(\rho_i) = u_i$ on compact form as

$$-\frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} = \lambda u_i \tag{4}$$

As an eigenvalue problem we get

$$\begin{bmatrix} d & a & 0 & \cdots & 0 \\ a & d & a & \vdots & \vdots \\ 0 & a & d & \ddots & 0 \\ \vdots & \cdots & \ddots & \ddots & a \\ 0 & \cdots & 0 & a & d \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_{N-2} \\ u_{N-1} \end{bmatrix} = \lambda \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_{N-2} \\ u_{N-1} \end{bmatrix}, \tag{5}$$

without considering the endpoints u_0 and u_N, and with $d = \frac{2}{h^2}$ and $a = -\frac{1}{h^2}$. This eigenvalue problem has analytical eigenpairs with eigenvalues as

$$\lambda_j = d + 2a\cos(\frac{j\pi}{N+1}), \quad j = 1, 2, \cdots, N. \tag{6}$$

2.3 Jacobi's method

Now we implement the Töplitz matrix from the eigenvalue problem in equation 5 into our program. This is dependent on the dimension of the matrix and the value for ρ_{max} which we choose. Then we make a check of the matrix implementation where we compare the eigenvalues we get from the analytical eigenvalues in equation 6 for a small Töplitz matrix of dimension 8 and with $\rho_{max} = 1$ with Python's numpy.linalg.eig() for finding eigenpairs of the Töplitz matrix.

We are now going to solve the Töplitz matrix with Jacobi's method. The main idea with Jacobi's method is to do several similarity (unitary) transformations on the eigenvalue matrix in equation 5, till it is transformed to a diagonal matrix containing only the eigenvalues on the diagonal with zeros as the rest of the elements. The Jacobi rotation matrix

$$\begin{bmatrix} 1 & 0 & \cdots & 0 & \cdots & 0 \\ 0 & \ddots & \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & \cos\theta & \vdots & \sin\theta & \vdots \\ \vdots & \vdots & \cdots & 1 & 0 & \vdots \\ \vdots & \cdots & -\sin\theta & 0 & \cos\theta & 0 \\ 0 & \cdots & 0 & \cdots & 0 & 1 \end{bmatrix} \tag{7}$$

is an orthogonal $(n \times n)$ transformation matrix that performs a plane rotation around the angle θ in Euclidean n-dimensional space (Hjorth-Jensen (1)[2]). The angle θ is arbitrary and chosen such that all off-diagonal elements in **B** in equation 1 is zero. To do this we calculate the Frobenius norm of the matrix **A**. Since the Frobenius norm is conserved under unitary transformations:

$$||\mathbf{A}||_F^2 = ||\mathbf{B}||_F^2 = \sum_{ij} |a_{ij}|^2 = \sum_{ij} |b_{ij}|^2 \tag{8}$$

After taking the Frobenius norm we find the off-diagonal elements in **B** as:

$$off(\mathbf{B})^2 = ||\mathbf{B}||_F^2 - \sum_{i=1}^{n} b_{ii}^2 = off(\mathbf{A})^2 - 2a_{kl}^2 \tag{9}$$

For each similarity transformation we can see that the matrix moves closer to diagonal form for each step. To do this numerically we implement a function for finding the maximum non-diagonal element value for which we start to use the Jacobi rotation matrix on, and this returns the position of the element to be used in the Jacobi rotation matrix. After one transformation the function finds the next maximum element. This continues until we are left with the diagonal matrix we want containing only the eigenvalues (or at least has non-diagonal elements smaller than a given tolerance $1e-8$). This function is implemented as:

```python
for i in range(n):
    for j in range(i+1, n):
        if abs(matrix[i,j]) > max_elem:
            max_elem = abs(matrix[i,j])
            max_k = i
            max_l = j
```

For simplification we define:

$$s = \sin\theta, \quad c = \cos\theta, \quad t = \tan\theta = \frac{s}{c} \quad \tau = \cot 2\theta = \frac{a_{ll} - a_{kk}}{2a_{kl}}$$

Here we define θ such that $a_{kl} \neq 0$ for the non-diagonal elements of the transformed matrix. This yields the quadratic equation

$$t^2 + 2\tau t - 1 = 0.$$

Where we easily find

$$t = -\tau \pm \sqrt{1+\tau^2}, \quad c = \frac{1}{\sqrt{1+t^2}}, \quad s = tc.$$

[2]Computational Physics (2015) - Chapter 7.4 Jacobi's method

If we get a large value for τ we could get wrong results when calculating t. For large τ we could get a situation with $t \approx -\tau \pm \sqrt{\tau^2}$, which the computer would have to solve. This can give round-off errors. So we have to make a change that takes this case into consideration. We rewrite t by its conjugate and implement it under an if-test:

```
if tau >= 0:
    t = 1./(tau + np.sqrt(1. + tau**2))
else:
    t = -1./(-tau + np.sqrt(1. + tau**2))
```

For a similarity transformation like the one in equation 1, where **U** is the rotation matrix in equation 7 with the non-zero elements

$$\begin{aligned} u_{kk} &= u_{ll} = c \\ u_{kl} &= -u_{lk} = -s \\ u_{ii} &= -uii = 1 \quad i \neq k \quad i \neq l, \end{aligned}$$

we can rewrite it as:

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

Here we have to choose the angle θ such that all $b_{kl} = 0$, which is what we do with the function for the maximum non-diagonal elements and the Jacobi rotation matrix. We also compute the eigenvectors for the matrix. Since Jacobi's method is a slow algorithm, we choose to use numba's function **jit** to speed up the calculations for the rotation function and the function for finding the maximum non-diagonal element. For the solver for finding the eigenvalues we also take the time and number of transformations of the Jacobi rotation/maximum off-diagonal algorithm to compare for different matrix dimensions.

2.4 Testing algorithms

The main algorithms are found in the program *main.py* in a GitHub-folder as found in Appendix A by following the link there. The programs for the testing of the algorithms we are about to explain are found in the same folder with name *tests.py*. We test for small dimensions of matrices. For small matrices we can more easily control the expected results.

The first test we implement is a test of our matrix and the maximum non-diagonal element function. For this test we manually make a 5×5 matrix with random numbers. This is to easily check that the function can find the maximum non-diagonal element in the matrix. We also use a numpy function called **numpy.lib.stride_tricks.as_strided** to numerically find the maximum element in the matrix, and which is treated as analytical results and only used to cross-check our function. Then we implement an error-call for if our function finds a diagonal element, or if it finds another element which is not the same as for the numpy function.

Next we want to test that the eigenvalues we get from out constructed matrix in equation 5 is correct. Here we choose to test for matrix dimensionality 8×8. We calculate the analytical eigenvalues of the eigenvalue problem which is given in equation 6, and compare it with Python's **numpy.linalg.eig** of our constructed matrix. If they do not match within a tolerance of $1e-10$, we assert an error.

The last test we want to do is a test of Jacobi's method on our constructed Töplitz matrix. Here we want to check that we get the correct eigenpairs after the similarity transformations. First we check that Jacobi's method gives us the same eigenvalues as for the numpy function we used earlier, for any given square matrix of dimensionality we choose. Here we test a 5×5 matrix. If they do not match within a tolerance of $1e-8$, we assert an error. We can also take the time of the algorithm and see number of transformations if we choose so. Then we want to check that the orthogonality of the eigenvectors after transformations are preserved. To check this we take the inner product of the eigenvectors after the transformations. We set the test of orthogonality as successful if the inner products are smaller than a tolerance of $1e-8$. We check for the case if the element $i \neq j$ within the tolerance, and if the element $i = j$. If the tests fails, then we assert an error.

Another thing we do is to make some plots to see the dependency the matrix dimension has on the number of transformations needed and the time they take. All these plots can be seen in the Figures-folder in GitHub (A).

2.5 Quantum dots, one electron

Next, we want to expand our numerics to a quantum mechanical problem for an electron moving in a three-dimensional harmonic oscillator potential. We will assume spherical symmetry, and will look at the radial part of Schrödinger's equation in spherical coordinates with $r \in [0, \infty)$:

$$-\frac{\hbar^2}{2m}\left(\frac{1}{r^2}\frac{d}{dr}r^2\frac{d}{dr} - \frac{l(l+1)}{r^2}\right)R(r) + V(r)R(r) = ER(r) \tag{10}$$

The potential for this case is the harmonic oscillator potential, $V(r) = \frac{1}{2}kr^2 = \frac{1}{2}m\omega^2 r^2$. The energy is then the corresponding harmonic oscillator energy with oscillator frequency ω as:

$$E_{nl} = \hbar\omega\left(2n + l + \frac{3}{2}\right) \tag{11}$$

$n = 0, 1, \cdots$ stands for which state we are looking at, while $l = 0, 1, \cdots$ is the orbital momentum of the electron.

Then we do a substitution with $R(r) = \frac{1}{r}u(r)$ with boundary conditions $u(0) = u(\infty) = 0$ to get:

$$-\frac{\hbar^2}{2m}\frac{d}{dr^2}u(r) + \left(V(r) + \frac{l(l+1)}{r^2}\frac{\hbar^2}{2m}\right)u(r) = Eu(r) \tag{12}$$

Now we want to scale the equation with a dimensionless variable $\rho = \frac{r}{\alpha}$, where α is a constant with dimension length. The harmonic oscillator potential is then $V(\rho) = \frac{1}{2}k\alpha^2\rho^2$. In this project we will use $l = 0$. So now with the new potential and scaling, Schrödinger's equation is now:

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

Then we multiply the equation with $\frac{2m\alpha^2}{\hbar^2}$:

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

Since α is a constant, we can fix it however we want. So to make the equation easier we fix it to be:

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

Then we can define one the right hand side, a variable for the eigenvalue of the problem as

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

The final Schrödinger equation that we have to solve has now become

$$-\frac{d^2}{d\rho^2}u(\rho) + \rho^2u(\rho) = \lambda u(\rho). \tag{13}$$

This now has the same type of form like the buckling beam problem in equation 3 just with another term. So we will do the exact same as we did earlier by discretization and a Taylor approximation to rewrite with the same $\rho_i = \rho_0 + ih$. We have to use $\rho_{max} = \rho_N$ and not equal to ∞, since infinity cannot be represented numerically. So we can now rewrite the Schrödinger equation (13) as

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

In a more compact way we write:

$$-\frac{u_{i+1}-2u_i+u_{i-1}}{h^2}+\rho_i^2 u_i = -\frac{u_{i+1}-2u_i+u_{i-1}}{h^2}+V_i u_i = \lambda u_i \tag{14}$$

In the buckling beam problem we had no potential working on it. In this problem we have the harmonic oscillator potential. This means that we have to add a potential term to the diagonal elements $d_i = \frac{2}{h^2} + V_i$, while the non-diagonal elements stays the same and are constants $e_i = -\frac{1}{h^2}$. Schrödinger's equation can then again be rewritten as

$$d_i u_i + e_i u_{i-1} + e_i u_{i+1} = \lambda u_i. \tag{15}$$

This equation can be written as a symmetric matrix eigenvalue problem just like equation 5, where we have left out the endpoints u_0 and u_N.

As we can see, our eigenvalue problem for the electron has the same form and is solved in the same way as for the buckling beam problem numerically. The only thing we do is to add the harmonic oscillator potential to the diagonal elements. So now we can experiment with the dimensionality of the matrix and the value of ρ_{max}. We want to find optimal values for the dimensionality and ρ_{max} such that the eigenvalues we get are close enough to the analytical eigenvalues $\lambda = 3, 7, 11, 15, ...$ with 3-4 leading digits after the decimal. We also look at the number of similarity transformations needed to produce these results. The eigenvalues for the three lowest-lying states are then plotted for the optimal values.

2.6 Quantum dots, two electrons

Once again we will expand our problem. Now we add another electron to the harmonic oscillator well. These two electrons interact with each other by a repulsive Coulomb interaction. We start with the following radial Schrödinger's equation for one electron:

$$-\frac{\hbar^2}{2m}\frac{d^2}{dr^2}u(r)+\frac{1}{2}kr^2u(r) = E^{(1)}u(r) \tag{16}$$

Then for two electrons with no interaction and the two-electron energy $E^{(2)}$, we get

$$\left(-\frac{\hbar^2}{2m}\frac{d^2}{dr_1^2}-\frac{\hbar^2}{2m}\frac{d^2}{dr_2^2}+\frac{1}{2}k(r_1^2+r_2^2)\right)u(r_1,r_2) = E^{(2)}u(r_1,r_2). \tag{17}$$

Now $u(r_1, r_2)$ is a two-electron wave function. Since we do not have an interaction between the electrons here, this can be taken as the product between two single-electron wave functions.

To simplify the equation we introduce the center-of-mass coordinate $\mathbf{R} = \frac{1}{2}(\mathbf{r}_1 + \mathbf{r}_2)$ and the relative coordinate $\mathbf{r} = \mathbf{r}_1 - \mathbf{r}_2$. This makes it possible to solve the equation as a single body problem, instead of a two-body problem which is more difficult. Schrödinger's equation with these coordinates becomes

$$\left(-\frac{\hbar^2}{m}\frac{d^2}{dr^2} - \frac{\hbar^2}{4m}\frac{d^2}{dR^2} + \frac{1}{4}kr^2 + kR^2\right) u(r,R) = E^{(2)}u(r,R). \tag{18}$$

Next we separate the wave function as $u(r,R) = \psi(r)\phi(R)$ and with the two-electron energy as $E^{(2)} = E_r + E_R$, where E_r is the radial energy and E_R is the center-of-mass energy. Then by adding the Coulomb interaction as

$$V(r_1,r_2) = \frac{\beta e^2}{r}, \quad \beta e^2 = 1.44 \text{ eVnm},$$

to Schrödinger's equation (18) we obtain the r-dependent term as

$$\left(-\frac{\hbar^2}{m}\frac{d^2}{dr^2} + \frac{1}{4}kr^2 + \frac{\beta e^2}{r}\right)\psi(r) = E_r\psi(r). \tag{19}$$

This is similar to equation 12, so we introduce again the dimensionless variable ρ and do the same calculations as before. We end up with

$$-\frac{d^2}{d\rho^2}\psi(\rho) + \frac{mk}{4\hbar^2}\alpha^4\rho^2\psi(\rho) + \frac{m\alpha\beta e^2}{\rho\hbar^2}\psi(\rho) = \frac{m\alpha^2}{\hbar^2}E_r\psi(\rho). \tag{20}$$

Now we define a new frequency term

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

and fix α to be

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

Then we define the eigenvalues for the problem as

$$\lambda = \frac{m\alpha^2}{\hbar^2}E. \tag{21}$$

The final Schrödinger equation to be solved is now

$$-\frac{d^2}{d\rho^2}\psi(\rho) + \omega_r^2\rho^2\psi(\rho) + \frac{1}{\rho} = \lambda\psi(\rho). \tag{22}$$

We want to study the problem for several values of the oscillator frequency $\omega_r = [0.01, 0.5, 1, 5]$ for the ground state with $l = 0$, which we now consider as the strength of the oscillator potential. In this case we also have to find the optimal/stable values for the dimensionality and ρ_{max} for the ω_r values. Since the Schrödinger equation we have obtained is on the form as the two previous cases, we reuse our code but now with a new potential $V(\rho) = \omega_r^2\rho^2 + \frac{1}{\rho}$. Then we look at our eigenvalues for the ground states, and plot the ground state for the different ω_r's.

3 Results

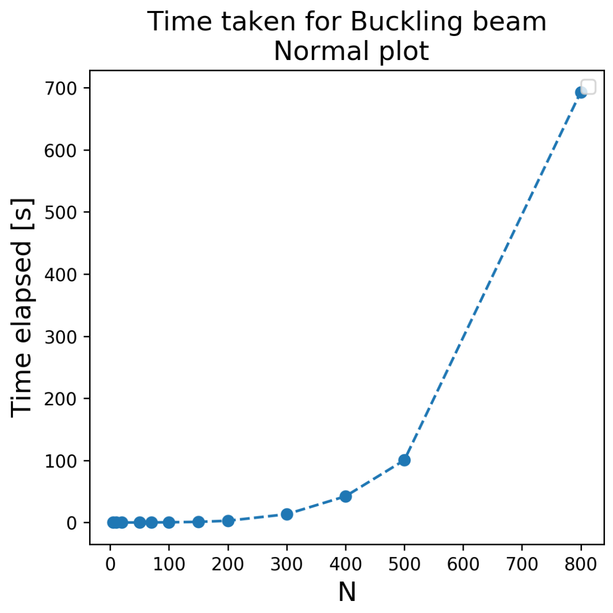
For our testing of the buckling beam we chose to run for smaller matrices. This is because smaller matrices need fewer similarity transformations and does not take a lot of time. With our code, all the tests in **tests.py** were successful. Thus, nothing was printed when we ran the program, other than printouts to verify that all the tests actually started. The output can be seen at the bottom of the code.

Then we wanted to see how the run-time and number of transformations needed to reduce the Töplitz matrix to a diagonal eigenvalue matrix, were affected by the dimension of the matrix. In the folder "Figures", which you find by going to the GitHub-link in Appendix A, lies plots for the time taken for the similarity transformations and the number of transformations as functions of the matrix dimensionality. We have plotted three different types for each of the two cases;

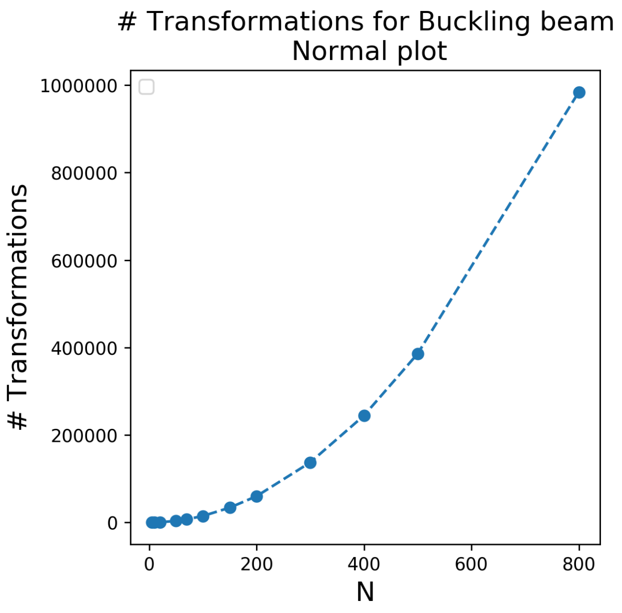
- log-log plot, where we get a linear growth in both cases with increasing dimensionality.
- semi-log plot, where we see there is a high increase in time and transformations for the first couple of hundred dimensions before it starts to flatten more out as the dimensionality increases.
- a "normal" plot as seen in Figure 1, where we see that the curve has almost an exponential growth with few transformations and short times for the first couple of hundred dimensions.

In Table 1 we see results of the increasing matrix dimensionality and how it affects the CPU-time of the algorithm and the number of transformations needed to reduce the Töplitz matrix to a diagonal matrix. We look at a mean of the CPU-times since they will change for each time we run the program. So we have run the program 8 times and taken the mean. The difference in the CPU-time comes from that the CPU may be doing other stuff when we run the program. The CPU-time differ from run to run, but the number of transformations is always the same for the same dimensionality. This table is just to see the result numbers, while Figure 1 gives us a more overview of the results as plots. From the table and the plots we can conclude:

- Up till the 200 $\times$ 200 matrix, the CPU-time is under two seconds with a very small increase between the dimensions. Up to that dimension we also see that the number of transformations are well under 100 000 transformations, and this also does not increase much. For dimensions larger than 200 we see that the increase is much higher for both. So when we reach dimension of 500 the time has gone up to about 80 seconds while the number of transformations is up to about 386 000. And for dimension of 800 we almost reach 1 million transformations after 8.5 minutes. So for big matrices the Jacobi method is very slow, and we have also used **jit** in our code to better the CPU-time. In this case the matrix we have used is a tridiagonal symmetric matrix, which we would expect to be a lot faster than say a dense matrix. We have also tried to run a 1000 $\times$ 1000 matrix, but this took incredibly long time, and is not necessary since we have made our point with the lower dimensions already.



(a) Plot of the time of the transformations as function of dimensionality



(b) Plot of the number of transformations as function of dimensionality

Figure 1: To the left we see a plot of the mean time for 8 runs of the Jacobi method for reducing the Töplitz matrix to an eigenvalue diagonal matrix as a function of the matrix dimensionality. To the right we see a plot of the number of transformations as a function of matrix dimensionality needed to reduce the Töplitz matrix by use of the Jacobi method. Both has a near exponential growth, but the right plot has a higher increase between each point than the left. So the mean time does not change much until we go to very high dimensionality:

Dimensionality n	Transformations	Mean CPU-time [s]
5	26	0.3142630
10	123	0.0002465
20	540	0.0012635
50	3′643	0.0191172
70	7′269	0.0401410
100	14′809	0.135744
150	33′897	0.6312082
200	60′129	1.9526031
300	137′374	9.3065250
400	244′553	31.965839
500	385′969	80.254807
800	984′354	511.37819

Table 1: In this table we see the results of the affect the dimensionality of the matrix has on the (mean) CPU-time of the algorithm and the number of similarity transformations. This is visualized in Figure 1.

For the one electron in a harmonic oscillator potential well we compare our numerically calculated eigenvalues with the analytical eigenvalues which should be 3, 7, 11 and 15. In Figure 2 we have plotted the probability distribution of the dimensionless wave function for the three first eigenvalues as function of the dimensionless variable ρ. In Table 2 we see the first four eigenvalues we got when finding the optimal values for the dimensionality and ρ_{max} compared to the analytical ones. For the numerical results in Table 2 we got the best eigenvalues for a 400×400 matrix with $\rho_{max} = 10$. The mean CPU-time in this case is about 30 seconds for 206 466 similarity transformations. We see that the first eigenvalues has the best accuracy to the analytical, and the accuracy decreases with increasing eigenvalue. In Figure 2 we see that the first eigenvalue has the highest peak. The second eigenvalue has the second highest and so on, and for each eigenvalue there is added another peak which is smaller than the last. From this we can conclude that the higher excited states have a larger spatial distribution than the lower energetic states.

For the two electron in a harmonic oscillator potential well we analyze the ground state eigenvalue for different oscillator potential strengths $\omega_r = [0.01, 0.5, 1, 5]$. In Figure 3 we have plotted the probability distribution for the dimensionless wave function for the ground state eigenvalue for the different potential strengths. There we see that the distribution widens for smaller potential strengths. So with increasing potential strength the peak is higher and closer to the origin. This means that the probability of finding the two elec-

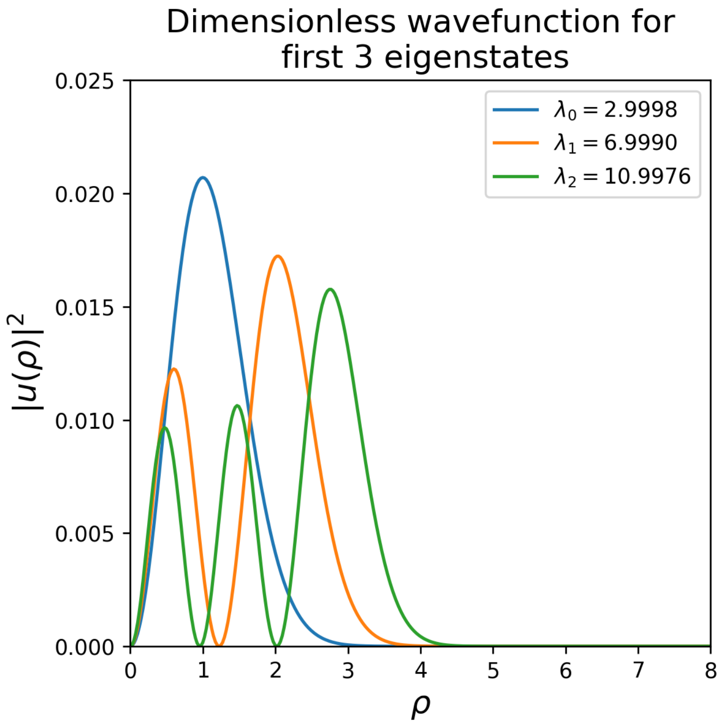


Figure 2: One electron in a harmonic oscillator potential well: Plot of the probability distribution of the dimensionless wave function as a function of the dimensionless variable ρ for the three first eigenvalues with matrix dimension 400×400 and $\rho_{max} = 10$. These eigenvalues took about 30 seconds in average and needed 206 466 similarity transformations.

Eigenvalues λ	Numerical	Analytical
λ_0	2.9998	3
λ_1	6.9990	7
λ_2	10.9976	11
λ_3	14.9956	15

Table 2: In this table we see the eigenvalues for the four first eigenstates for the analytical and the numerical we got with Jacobi's method for a tridiagonal matrix of dimension 400×400 and with $\rho_{max} = 10$. We see that the numerical eigenvalues are very close to the analytical.

trons closer together, and closer to the center of the potential well, is higher for bigger ω_r, since increasing ω_r decreases the width of the potential well. We can then interpret ρ as the distance between the two electron in the potential well. In Table 3 we see the eigenvalues for the ground state for the different potential strengths and the number of transformations of Jacobi's method. We can see there that the eigenvalues increase with increasing strength with fits which Figure 3 where the highest peak is the highest strength, which means it has the highest energy. We can also see that as the strength increases, the number of transformations decrease.

Potential strengths ω_r	Ground state eigenvalue λ_0	Transformations
0.01	0.31164	209'556
0.5	2.23007	209'392
1	4.05767	207'506
5	17.44360	195.523

Table 3: In this table we see the ground state eigenvalues and number of similarity transformations for the varying potential strengths. For increasing strengths we see that the eigenvalue increases which means that the energy for this state increases, and that there is a decrease in the number of transformations.

4 Conclusion

One of the main things in this project was to solve three eigenvalue problems for a tridiagonal symmetric Töplitz matrix with the use of Jacobi's method numerically. We have discretized and scaled differential equations with the ends results being on the same form with different potential terms.

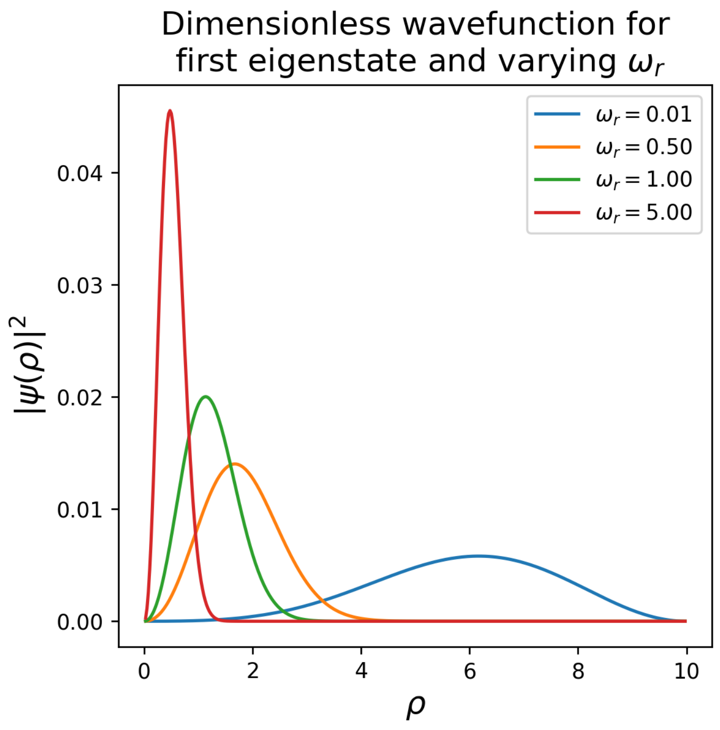


Figure 3: Two electrons in a harmonic oscillator potential well: Plot of the probability distribution of the dimensionless wave function as a function of the dimensionless variable ρ for matrix with dimension 400×400 and $\rho_{max} = 10$. Here we see that with increasing potential strength, we get increasing distribution and smaller peak. This corresponds to that the width of the potential well decreases with increasing potential strength, such that the electrons have a higher possibility of being closer to each other around the center of th well for higher strengths.

Then we have studied the affect the dimensionality of the matrix have on the CPU-time and number of unitary transformations needed to make the Töplitz matrix into a diagonal eigenvalue matrix. There we came to the conclusion that for big matrices the CPU-time quickly becomes very high, as do the number of transformations. So we can conclude from our CPU-time results that Jacobi's method is not a very fast algorithm for big matrices, even though we have used **jit** compilation to speed up the code. However, Jacobi's method is a versatile algorithm as we have seen by scaling and reducing different differential equations to 1D problems and then solving them numerically. After getting the problem on this form, then the algorithm can be used to solve several physical problems with few changes in the code. Jacobi's method is also a very reliable algorithm for solving eigenvalue problems even though it is not the most efficient as we have seen.

Then we looked at different potential strengths for the two electron problem. There we concluded with that higher potential strengths give higher eigenvalues for the ground state which then gives higher energies. This also made the potential well more narrower allowing the two electrons to be within a shorter range of each other around the center of the well.

Future work could be to find more efficient algorithms to solve our eigenvalue problems like Lanczos' and Householder's algorithms. We could also add the repulsion between the two electrons, and analyze the effect this will have on the system.

A Appendix

Link to GitHub repository:
`https://github.com/krilangs/FYS4150/tree/master/Project2`

References

[1] M. Hjorth-Jensen. **Computational Physics - Lecture notes Fall 2015**. 2015.