A Study of Numerical Eigenvalue Solvers

Caspar William Bruenech

October 1, 2019

Abstract

Various methods for numerically computing the eigenvalue- and vectors for matrices were implemented and tested against each other. Results show that the Jacobi method has low scalability with high computation time needed for relatively small matrices. The energies of an electron in a quantum harmonic oscillator was calculated as the eigenvalues of a symmetrical, tridiagonal matrix. This produced eigenvalues with a high error, which was shown to be due to the boundary condition of the problem requiring an approximation to ∞. The Lanczos algorithm combined with eigenvalue solvers in the Julia programming language was implemented and shown to produce accurate results with a significant speedup compared to the Jacobi method.

1 Introduction

Computing the eigenvalues of matrices is a vital part of solving certain physical problems. However, these computations are often of a high complexity, which can lead to requiring a significant amount of CPU hours to perform the calculations, especially for larger matrices. This experiment will look at how different methods can be used to compute the eigenvalues- and vectors of a given matrix, and how they vary in complexity and accuracy. We will limit ourselves to symmetrical matrices, and introduce a tridiagonal Toeplitz matrix, which will be solved as if it was a dense or sparse matrix. We will solve these eigenvalues using two physical examples with different boundary conditions; one of a buckling beam fastened at two ends, and one of an electron in a quantum oscillator potential well. The Jacobi method will be implemented to compute the eigenvalues. Finally we will look at how the Lanczos algorithm can be used to derive the same results while requiring a significantly smaller amount of CPU time.

2 Method

2.1 A Buckling Beam

To start, we will look at the example of a buckling beam which is fastened at both ends. The resulting model is one for a spring with two boundary conditions. A differential equation for the vertical displacement $u(x)$ of the beam (spring) is given as:

$$\gamma \frac{d^2u(x)}{dx^2} = -Fu(x) \tag{1}$$

Where γ is a constant given by the physical properties of the beam. We will denote the length of the beam as L, which gives us $x \in [0, L]$. The force F is then applied at $(L, 0)$ in direction towards the origin. The Dirichlet boundary conditions are applied. By defining the variable

$$\rho = x/L$$

we can reorder the equation as:

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

Where we have defined $\lambda = \frac{FL^2}{R}$. By discretizing and approximating the second derivative of $u(\rho)$ as

$$u''(\rho) \approx \frac{u(\rho + h) - 2u(\rho) + u(\rho - h)}{h^2}$$

Where $h = \frac{\rho_N - \rho_0}{N}$ is the step size with N integration points between $\rho_{min} = \rho_0$ and $\rho_{max} = \rho_N$ Which gives us the final expression for the approximated, discretized differential equation as:

$$-\frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} = \lambda u_i \tag{3}$$

We observe that this can be expressed as a matrix-vector multiplication $A\vec{u} = \lambda\vec{u}$, with $\vec{u}^T = [u_1, u_2, \dots u_{N-1}]$ and

$$A = \begin{bmatrix} d & a & 0 & 0 & \dots & 0 & 0 \\ a & d & a & 0 & \dots & 0 & 0 \\ 0 & a & d & a & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & \dots & \dots & a & d & a \\ 0 & \dots & \dots & \dots & \dots & a & d \end{bmatrix}$$

where $d = 2/h^2$, and $a = -1/h^2$.

Which means that we now have an eigenvalue problem to solve.

2.2 A quantum particle in three dimensions

In addition to the buckling beam scenario as described above, the eigenvalues, corresponding to the energy levels of an electron in a spherically symmetrical harmonic oscillator potential were computed. Starting with the radial part of the Schroedinger's equation for one electron;

$$-\frac{\hbar^2}{2m}\left(\frac{1}{r^2}\frac{d}{dr^2}r^2\frac{d}{dr} - \frac{l(l+1)}{r^2}\right)R(r) + V(r)R(r) = ER(r) \tag{4}$$

Where $V(r)$ is the harmonic oscillator potential, and E is the energy of the electron. By rewriting and scaling the equation we are eventually left with

$$-\frac{d^2}{d\rho^2}u(\rho) + \rho^2 u(\rho) = \lambda u(\rho). \tag{5}$$

Where $\lambda = \frac{2m\alpha^2}{\hbar^2}E$. We recognize that this equation is similar to 1. However, in this case the boundary conditions are different. They are now

$$u(0) = 0 \qquad u(\infty) = 0.$$

Discretizing the equation then produces

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

where

$$h = \frac{\rho_{max} - \rho_{min}}{N}$$

Is the step length, ρ_{max} is the numerically approximated ∞,ρ_{min} is the starting point, set to be 0 in this case, and N is the number of integration points. By defining $V_i = \rho_i^2$, we get the diagonal and non-diagonal elements of the matrix as, respectively:

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

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

Which produces the equation

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

and the matrix-vector multiplication

$$\begin{bmatrix} d_1 & e & 0 & 0 & \dots & 0 & 0 \\ e & d_2 & e & 0 & \dots & 0 & 0 \\ 0 & e & d_3 & e & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & \dots & \dots & e & d_{N-2} & e \\ 0 & \dots & \dots & \dots & \dots & e & d_{N-1} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \dots \\ \dots \\ \dots \\ u_{N-1} \end{bmatrix} = \lambda \begin{bmatrix} u_1 \\ u_2 \dots \\ \dots \\ \dots \\ u_{N-1} \end{bmatrix}$$

Which we again recognize as an eigenvalue problem, where the eigenvalues λ represent the different energy levels of the electron.

2.3 The Jacobi Eigenvalue algorithm

A known method for solving eigenvalue problems of real symmetric matrices is the Jacobi transformation method of matrix diagonalization. Given a real, symmetric matrix $A \in \mathbb{R}^{nxn}$ and an ortohonal matrix $S \in \mathbb{R}^{nxn}$ which performs a plane rotation around an angle θ, the similarity transformation

$$B = S^T A S$$

will change non-zero matrix elements of A by a factor related to the value of θ. By choosing θ such that the off-diagonal elements of A iteratively become zero, the remaining diagonal elements will be approximations to the real eigenvalues of A. More specifically, the elements of the resulting matrix B are given by:

$$b_{ii} = a_{ii}, \quad i \neq k, i \neq l$$

$$b_{ik} = a_{ik} \cos\theta - a_{il} \sin\theta, \quad i \neq k, i \neq l$$

$$b_{il} = a_{il} \cos\theta + a_{ik} \sin\theta, \quad i \neq k, i \neq l$$

$$b_{kk} = a_{kk} \cos^2\theta - 2a_{kl} \cos\theta \sin\theta + a_{kk} \sin^2\theta$$

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

By defining the quantities $\tan\theta = t = s/c$, where $s = \sin\theta$ and $c = \cos\theta$, and

$$\tau = \frac{a_{ll} - a_{kk}}{2a_{kl}} = \cot 2\theta,$$

using that $\cot 2\theta = \frac{1}{2}(\cot\theta - \tan\theta)$ and our requirement for having $\theta = 0$, we produce the quadratic equation

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

Which gives us:

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

and

$$c = 1/\sqrt{1 + t^2}$$

$$s = tc$$

Where the smallest of the roots are chosen for best precision.

One of the problems which may arise here is in the calculation of τ. Since the method aims at reducing the off-diagonal elements to zero, the value of a_{kl} may at some point become very large as the procedure approaches convergence. This can result in the value of τ becomes extremely large. Due to the limitations of storing numbers on a computer, the value $\sqrt{1+\tau^2}$ when $\tau >> 1$, will result in τ, which will give the wrong values of t, which again will give the wrong values of $\cos\theta$ and $\sin\theta$.

The indices k and l are the indices of the largest off-diagonal element in A. Since the goal of the transformation is to drive the off-diagonal elements of A to zero, we want that

$$off(B)^2 = \sum_{i,j_{i\neq j}} b_{ij}^2 < off(A)^2 = \sum_{i,j_{i\neq j}} a_{ij}^2.$$

We also know that for orthogonal/unitary transformation, the Frobenius norm is conserved. I.e.

$$||A||_F^2 = ||B||_F^2 = \sum_{i,j} a_{ij}^2 = \sum_{i,j} b_{ij}^2$$

Which means that for a given k and l, we have:

$$a_{kk}^2 + a_{ll}^2 + 2a_{lk}^2 = b_{kk}^2 + b_{ll}^2 + 2b_{kl}^2$$

Where b_{kl} is the element which we want to drive to 0. Using this, we can now write the off-diagonal square sum of B as:

$$off(B)^2 = ||B||_F^2 - \sum_{i,j} b_{ij}^2$$

$$= ||A||_F^2 - \sum_{i,j} a_{ij}^+ a_{kk}^2 + a_{ll}^2 - b_{ll}^2 - b_{kk}^2$$

We recognize the rightmost term as $-2a_{kl}^2$, giving us:

$$off(B)^2 = off(A)^2 - 2a_{kl}^2$$

Showing that choosing the largest off-diagonal element a_{kl} will shrink the off-diagonal elements of the resulting matrix B the most, speeding up the convergence time of the algorithm.

2.4 Lanczos' algorithm

The Jacobi algorithm as described above is one way of computing the eigenvalues for dense matrices. However, it is notoriously slow in its computation. Another way of computing the eigenvalues of a matrix is by using Lanczos' algorithm [1]. The algorithm is an iterative procedure which when supplied with a Hermitian matrix of dimension $n \times n$ produces a tridiagonal matrix $T \in \mathbb{R}^{m\times m}$ with m being the number of matrices, which, when T is $n \times n$ has the same eigenvalues as the input matrix A. For a input matrix of large dimension, the eigenvalues of T will correspond to the m most extreme eigenvalues of A. It is therefore a useful algorithm for large matrices when only certain eigenvalues are of interest. The existence of many specialized algorithms for computing the eigenvalues of tridiagonal matrices makes this algorithm a good choice for dealing with matrices of high dimension. See 3.4 for implementation and results.

	4	10	50	100	200	250	300	350	400
N_T	6	153	4316	17663	70858	110785	160449	219643	284881
$\log_{10}$ max error	-15.13	-14.75	-14.01	-12.92	-13.31	-11.89	-13.26	-11.67	-11.66
Jacobi time [ms]	7.5 e-4	0.72	24.75	401.5	1.79 e4	4.07e4	8.09 e4	1.25 e5	2.23 e5
Julia time [ms]	45.15	0.871	8.967	20.05	59.55	65.89	72.67	87.31	112.8
Jacobi/Julia time	1.62 e-5	0.83	2.75	20.019	300	617	1112	1431	2031

Table 1: A table showing the results of performing the Jacobi method on the matrix as defined in 2.1 for varying orders. The value N_T denotes the number of transformations performed.

3 Implementation

All the code used in this experiment can be found in the following Jupyter notebook:

`https://github.com/casparwb/FYS3150/blob/master/2_project/project2_jupyter.ipynb`

3.1 Validating the Analytical Eigenvalues

As seen in [2]), a tridiagonal matrix which is also Toeplitz, such as the one used in this experiment, has analytical eigenvalues given by:

$$\lambda_k = a + 2\sqrt{bc}\cos\left(\frac{k\pi}{n+1}\right)$$

where a is the value of the diagonal elements of the matrix, and c and b are, respectively, the value of the sub- and superdiagonal elements. In our case the matrix is also symmetrical, meaning $b = c$. Using this property, we can validate the constructed matrix by comparing the analytical eigenvalues to the ones computed by the Linear Algebra functionality of the Julia Programming Language. Setting a tolerance to 10^{-10}, the computed eigenvalues were all found to be close to the analytical ones within the tolerance.

3.2 The Jacobi Method

The Jacobi algorithm as described in 2.3 was directly implemented. As the algorithm requires the value and location of the largest off-diagonal element in A for each iteration, a function for locating this was also implemented. To validity its accuracy, the function was successfully tested on a smaller matrix where the largest off-diagonal element and its position was known. Table 3.2 shows the results of the computations.

3.3 The Quantum Case

To extend our machinery to the specialized quantum mechanical case, a new matrix was constructed. This matrix was identical to the matrix used in the buckling-beam case, apart from different values of a and d, as shown in 2.2. In addition to this, the boundary conditions forces us to numerically approximate ∞, meaning we now have two parameters that will influence

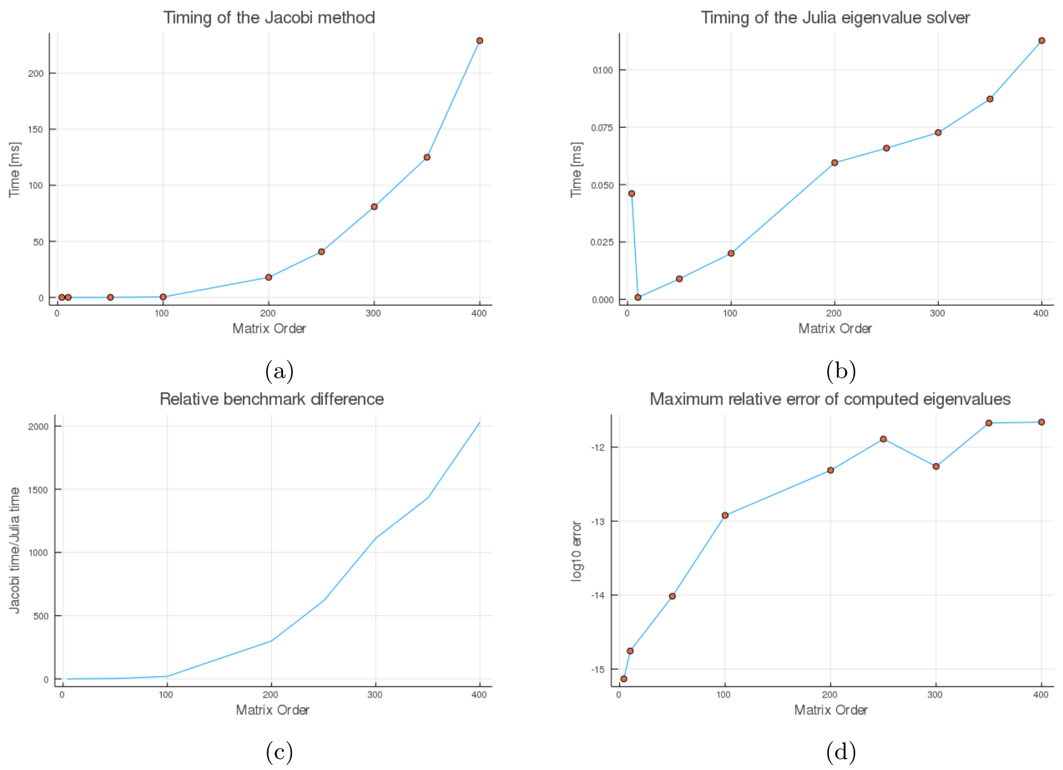


Figure 1: Figures visualising the data in table 3.2. The upper figures shows the time taken by respectively the Jacobi method and the eigenvalue solver in the Julia programming language as a function of the order of the matrix. The lower left figure shows the time taken by the Jacobi method divided by the time taken by the Julia solver, while the lower right figure shows the error of the computed eigenvalues as a function of matrix order.

the accuracy of our model; the number of integration points N and the approximation to ∞, $\rho_{max} = \Lambda$. This was implemented varying the value of ρ_{max} for each order of the matrix. Due to limitations in the available hardware combined with the high duration of a Jacobi computation, a smaller interval of matrix dimensions was used. The results are shown in figure 2.

3.4 A better algorithm

The Lanczos algorithm was implemented and timed over the same interval of matrix dimensions as the ones used when benchmarking the Jacobi method, plus two higher orders, using the same matrix. The version of the algorithm which was implemented is the on described in [3], which is, according to said article, the most numerically stable. The results are shown in table 3.4.

Plots of the times and errors are shown in figure 3.

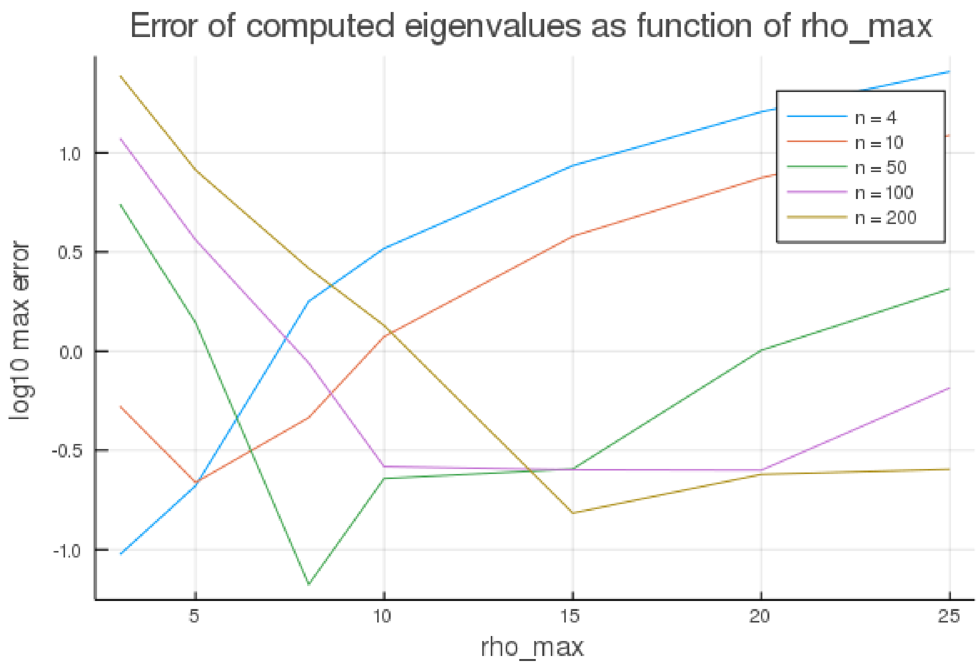
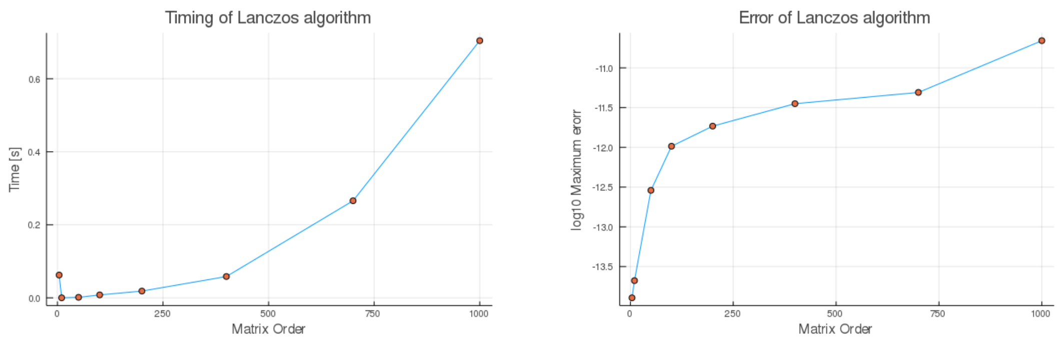


Figure 2: A visualisation of the error of the computed eigenvalues as a function of the numerical approximation to ∞, denoted by ρ_{max}. Each line is for a different matrix order.

	n							
	4	10	50	100	200	400	700	1000
Time [ms]	0.021	0.029	1.128	9.039	17.67	59.9	302.6	751.2
log10 max error	-13.89	-13.67	-12.54	-11.98	-11.73	-11.45	-11.30	-10.65

Table 2: An overview of the time taken by performing the Lanczos' algorithm plus the eigenvalue decomposition of the resulting tridiagonal matrix, and the error of the resulting eigenvalues.



(a) (b)

Figure 3: Two plots showing the data presented in table 3.4. The left plot shows how the time taken to compute the eigenvalues using the Lanczos algorithm increases with the dimensionality of the matrix, while the right plot shows how the error of the results vary with the same parameters.

3.5 Tests

Two unit tests were added to the experiment. One for testing the function for locating the maximum off-diagonal element in a matrix and its indices, and one for testing that the orthogonality of the computed eigenvectors in the implemented Jacobi method was maintained. In order to avoid too many additional computations (since testing orthogonality requires multiple vector-multiplications), the orthogonality was calculated only for every 100 iteration. The former test the was produced by constructing a smaller matrix with a known maximum off-diagonal value and its position in the matrix, while the latter test simply tested that the dot product between all the eigenvectors were below a set tolerance close to zero. Both tests were successful on all runs.

4 Analysis

While the Jacobi algorithm produced accurate results, it is no doubt that it is a performance bottleneck. An 4-factor increase in dimensionality increased the CPU time by a factor of 1600. It is no surprise, then, that the method proved to be significantly slower than the eigendecomposition solver in Julia. However, for dimensions $n = 4$, and $n = 10$, the Jacobi method is actually faster than the Julia solver. The reason for this is most likely, based on the extremely high CPU time of the Julia solver for $n = 4$, latency in the function calls. To understand why the Jacobi method is so slow, we need to look at number of operations. According to [4], the method will converge after approximately $3n^2 - 5n^2$ transformations, with each transformation requiring $4n$ flops, totaling in at $12n^3 - 20n^3$ operations, which is high when compared to, for example, the QR algorithm via Householder transformations which requires $\frac{4}{3}n^3 + O(n^2) + O(n)$ operations. ([5]) One of the reasons the Jacobi algorithm is so slow, is because elements that have been set to zero in one iteration, can be changed to a non-zero value in another iteration, increasing the number of transformations necessary in order to reach convergence, as it may be necessary to set the same value to zero multiple times.

As seen in figure 2, for $n \leq 10$, the maximum error of the computed eigenvalues actually increase as the approximation of ρ_{max} increases, while for higher order of the matrix, the error decreases initially, before increasing again. For lower values of n, this turning point occurs at smaller values of ρ_{max} than for higher n. This implies that the approximation to ∞ should be higher for higher matrix dimensions. Since the accuracy of the numerical integration depends on the step size h, which we defined as

$$h = \frac{\rho_{max}}{N},$$

increasing ρ_{max} for a certain set matrix order N should decrease the accuracy of the results, as h is increased. However, increasing ρ_{max} increases the approximation to ∞, which means that the results are more realistic. In the buckling beam-example, the accuracy only depended on the matrix order N), but in this case, it also relies on the approximation to infinity. This approximation will increase the step size, thus decreasing the accuracy of the numerical computation, but a high value of ρ_{max} also implies a more realistic approximation to the set boundary conditions. This leads to a trade-off between the accuracy of the computed eigenvalues, the approximation to infinity, ρ_{max}, and the order of the matrix N. To find the ideal values of ρ_{max} and N, multiple combinations of the two parameters should be evaluated, and the fraction $h = \frac{\rho_{max}}{N}$ which results in the lowest error should be used as a reference for changing either value. This, however, means that to maintain the lowest error, both parameters have to be altered. Based on the results in 2, the best choice of these parameters are the ones which results in $h \in [0.075, 0.5]$.

The results of computing the eigenvalue factorization of a matrix using Lanczos' algorithm (table 3.4) shows that the algorithm, as expected, significantly outperforms the Jacobi method in terms of speed when computing the eigenvalues of a matrix. Already at matrix order of $n = 200$, using the algorithm followed by an eigenvalue solver computed the eigenvalues 1000 times faster than the Jacobi method, while the error was more or less the same for both methods. This algorithm is especially efficient when implemented with the Julia programming language, as the Linear Algebra package features specialized eigenvalue solvers for symmetrical, tridiagonal matrices. When calling the eigenvalue solver on a matrix which is of type SymTridiagonal, the solver was observed to perform $35 - 20$ times faster than for the same matrix of standard type Array. While the algorithm is, without a doubt, significantly faster than the Jacobi method, it is notorious for being numerically unstable ([3]). This was not observed in this experiment, but it should be noted that extra measures may have to be taken in order to produce sufficiently accurate results. The algorithm is also especially efficient of sparse matrices, as it takes advantage of this property, something the Householder transformations do not. In addition to the observed speed advantage, the error in the results were very similar to the error when using the Jacobi method.

5 Conclusion

To sum up, the Jacobi method produced accurate results, but it displayed an extreme performance bottleneck when compared to other eigenvalue solvers. However, the accuracy of the computed eigenvalues when applying it to the buckling beam-scenario were respectable, with a maximum observed relative error of $\approx 10^{-12}$ for all the matrix orders analysed in this experiment. When applying the Jacobi method to a specialized case from quantum physics, the accuracy was substantially lower, with a minimum relative error of only $\approx 10^{-1.5}$, showing that having to approximate in addition to the unavoidable approximations when performing numerical calculations is a severe problem in terms of the accuracy of the results. This may possibly be negated by further analysis, by analysis more combinations of ρ_{max} and N, and finding the ones which gives the lowest error. Finally, we observed how the Lanczos' algorithm, when combined with the specialized eigenvalue solvers for symmetrical, tridiagonal matrices in the Julia programming language, displayed a significant speedup compared to the Jacobi method, with observed speedups up to 2000 times faster while maintain the same low error of the results. While the Lanczos algorithm is reportedly useful for large ($n > 10^6$), sparse matrices, this was not possible to verify in this experiment due to memory limitations on the available hardware. In conclusion, the Jacobi method, while producing accurate results, is not favourable due to its low scalability. The Lanczos algorithm provides an efficient way of calculating the most important (extreme) eigenvalues of large, sparse matrices.

6 References

References

[1] C. Lanczos, *An iteration method for the solution fo eigen-value problem of linear differential an integral operators*, 1950.

[2] G. H. Golub and C. F. Van Loan, “Matrix computations. 1996”, Johns Hopkins University, Press, Baltimore, MD, USA (1996).

[3] J. K. Cullum and R. A. Willoughby, “Lanczos algorithms for large symmetric eigenvalue computations. Vol. I: Theory”, Classics in Applied Mathematics (2002).

[4] M. Hjorth-Jensen, *Lecture Notes 2015 FYS3150* (2015).

[5] G. Ermentrout, “Numerical recipes in C”, Mathematical Biosciences (1989).