Project 2: Writing an Eigenvalue Solver which can be applied to Scaled Differential Equations

Noah Oldfield
Department of physics, University of Oslo
(Dated: 5. oktober 2018)

In this report, we write a numerical eigenvalue solver using Jacobi's eigenvalue algorithm. We see how this solver can be applied to widely different physical systems when proper scaling of differential equations are made. There are three parts: in part 1 we study a buckling beam system, where we compare the computed eigenvalues of the solver with known analytical values. We then test the computed values towards the numpy.linalg.eig solver and find that the values hold up to a specific set precision. The runtimes of the two however greatly favour the written solver. In part 2 we study a quantum system in a spherical symmetric potential, we find that we can reuse our solver by simply adding the potential values to the diagonal of our matrix. In addition we find that $\rho_N = 10$ is a decent approximation to infinity for the quantum system, resulting in an error (between computed and analytical energy values) which tends to zero for a $(N \times N)$ matrix. In part 3 we study two electrons with a repulsion force and also in spherical symmetric potential, so called quantum dots. We study the ground state energy eigenvalues, which are proportional to the computed eigenvalues, and find that an increase in the potential strength leads to an increase in the seperation of the electrons.

Github repository: `https://github.com/nhofield/fys3150Projects.git`

I. INTRODUCTION

In this report we will use Jacobi's eigenvalue algorithm to write a common solver for three seemingly different physical systems. There are three parts: part 1-3 centers on a buckling beam system, a particle in spherically symmetrical potential and two electrons with a repulsion force respectively. Each will be shown solvable by the same solver by only adding elements to the diagonal of our matrix A custom to each system.

The solver will use Jacobi's eigenvalue algorithm to solve for the eigenvalues of a symmetric matrix A, and unit tests will be implemented in order to test mathematical properties of the algorithm and other functions in the program. There are great advantages when it comes to implementing unit tests during the development phase in light of debugging. Especially when the programs become relatively large and complex such as in this case.

II. THEORY

A. Unitary Transformations

Let $v_i^T = (v_{i1}, v_{i2}, \cdots, v_{in})$ where n=1,2,3,.. be an orthonormal vector from some orthonormal basis of dimension n; and let U be a unitary or orthogonal matrix.

A unitary or orthogonal transformation $\hat{w}_i = U\hat{v}_i$ is a transformation such that the orthonormality of a specific basis is preserved.

1. *Proof*

We'll consider another basisvector from the same basis of $\hat{v}_i$, $\hat{v}_j$ and both are considered to have real inputs. The inner product between $\hat{v}_i$ and $\hat{v}_j$ is assumed equal to the matrix product of v_i^T with $\hat{v}_j$ denoted

$$\langle \hat{v}_i, \hat{v}_j \rangle = v_i^T \hat{v}_j \tag{1}$$

thus the inner product of the unitary or orthogonal transformations is

$$\langle \hat{w}_i, \hat{w}_j \rangle = w_i^T \hat{w}_j = (U\hat{v}_i)^T U\hat{v}_j = v_i^T U^T U \hat{v}_j \tag{2}$$

since U is real orthogonal and using the associative property, then $(U^T U) = I$, where I denotes the identity matrix. Thus

$$\langle \hat{w}_i, \hat{w}_j \rangle = v_i^T \hat{v}_j = \langle \hat{v}_i, \hat{v}_j \rangle \tag{3}$$

Furthermore, since both $\hat{v}_i$ and $\hat{v}_j$ are elements of the same orthonormal basis, the inner product is equal to the Kronecker-delta

$$\langle \hat{w}_i, \hat{w}_j \rangle = \langle \hat{v}_i, \hat{v}_j \rangle = \delta_{ij} \tag{4}$$

which equates to zero for $i \neq j$ and 1 for $i = j$. Thus preserving the orthonormality.

In order to prove the same for a unitary transformation one must make a similar assumption with the inner product between two complex vectors $\langle \hat{v}_i, \hat{v}_j \rangle = \hat{v}_i^\dagger \hat{v}_j$ and use the unitary condition $U^\dagger U = I$ instead. Thus proving orthonormality in this case also.

Thus it would seem that unitary transformations preserve orthonormality for both real and complex bases, but the orthogonal transformations do not necessarily preserve orthonormality for complex bases. QED

B. Scaled differential Equations

1. Part 1: Buckling-Beam problem

The differential equation

$$\gamma \frac{d^2 u(x)}{dx^2} = -F u(x) \tag{5}$$

describes a vertically oscillating buckling beam of length L due to its rigidity γ and force F which acts to perpetually return the beam to equilibrium. The equation will be solved using Dirichlet boundary conditions over the discretized values of x$\in$ $[0, L]$ meaning the first and last values of x yields $u(L) = u(0) = 0$.

By introducing a dimensionless parameter $\rho = \frac{x}{L}$ the differential equation reduces to the following dimensionless equation

$$\frac{d^2 u(\rho)}{d\rho^2} = -\frac{FL^2}{R} u(\rho) = -\lambda u(\rho) \tag{6}$$

with the $\rho \in [0, 1]$ and $u(0) = u(1) = 0$ due to the definition of ρ.

2. Part 2: Schrodinger-Equation

Now introducing the Schrodinger equation for a particle in a spherical symmetric potential $V(r)$

$$-\frac{\hbar^2}{2m}\left(\frac{1}{r^2}\frac{d}{dr}r^2\frac{d}{dr} - \frac{l(l+1)}{r^2}\right)R(r) + V(r)R(r) = ER(r) \tag{7}$$

We'll take the potential to be the harmonic oscillator potential $V(r) = \frac{1}{2}kr^2$ where $k = m\omega^2$. The same procedure can be followed to scale the equation, though slightly more sofisticated. We begin by introducing two dimensionless variables $R(r) = \frac{u(r)}{r}$ and $\rho(r) = \frac{r}{\alpha}$ which turns out to simplify certain terms in the radial Schrodinger equation which come from the laplacian operator ∇^2 in spherical coordinates. The first step is to compute

$$\left(\frac{1}{r^2}\frac{d}{dr}r^2\frac{d}{dr}\right)\frac{u(r)}{r} \tag{8}$$

and following the algebra from this and inserting the potential $V(r) = \frac{1}{2}kr^2$ one can arrive at the equation

$$-\frac{d^2}{d\rho^2}u(\rho) + \frac{mk}{\hbar^2}\alpha^4\rho^2 u(\rho) = \frac{2m\alpha^2}{\hbar^2}Eu(\rho) \tag{9}$$

Now we can fix either of the constants on the left side in front of either the double derivative term or the other to 1. The purpose of the scaling is to reduce the problem to a general form of differential equation which can then be solved using a particular numerical solver. By choosing to fix the second term on the left side to 1 the equation reduces to the dimensionless differential equation

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

3. Part 3: Quantum-dots in 3 dimensions

We shall also be making computations for a system of two electrons in a harmonic oscillator potential which also interact through a repulsive Coulomb interaction. We shall not go into detail for the scaling of this equation, but simply post the differential equation and define all terms.

$$-\frac{d^2}{d\rho^2}\psi(\rho) + \omega_r^2\rho^2\psi(\rho) + \frac{1}{\rho} = \lambda\psi(\rho) \tag{11}$$

The equation is scaled such that $\rho = \frac{r}{\alpha}$ where α is a length reference parameter. The parameter ω_r represents the strength of the harmonic oscillator potential and shall take on the values [0.01, 0.5, 5]. The coordinate r and thereby ρ in this case represents the relative distance between the electrons.

C. Analytical solutions and other definitions

1. Analytical Eigenvalues for the Buckling Beam Toeplitz Matrix

The eigenvalues for the coming $(N \times N)$ matrix A, given by the finite difference method for the buckling beam problem, are given by

$$\lambda_j = d + 2a\cos\left(\frac{j\pi}{N+1}\right) \qquad for\ j = 1, 2, \cdots, N-1 \tag{12}$$

where d are the diagonal elements of A and a are the just above and just below diagonal elements of A.

2. Analytical Energy values for particle in spherical symmetrical potential with no orbital angular momentum

$$E_n = \hbar\omega\left(2n + \frac{3}{2}\right) \qquad n = 0, 1, 2, \cdots \tag{13}$$

where $\hbar$ is the reduced Planck constant and ω is the angular frequency.

3. Frobenius Norm

The frobenius norm of a $(m \times n)$ matrix A is denoted $||A||_F$

$$||A||_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|a_{ij}|^2} \tag{14}$$

where a_{ij} are the matrix elements of A.

III. NUMERICAL METHOD

A. Jacobi's Eigenvalue Algorithm

The Jacobi Eigenvalue Algorithm is a means to compute the eigenvalues of a symmetric matrix A by applying a sequence of orthogonal rotation transformations $J^T AJ$ until, in theory, the off diagonal elements of the resulting transformation are all zero. Thus resulting in a diagonalization with the resulting matrix being a diagonal matrix with the eigenvalues of A as it's diagonal elements.

In short, first the largest off diagonal element is chosen (which is due to being the most efficient way of zeroing out the off diagonals), the indices of the chosen element then defines a specific Givens rotation matrix, represented by J. Which the elements of are then to be calculated by

$$\tau = \frac{a_{ll} - a_{kk}}{2a_{kl}} \tag{15}$$

where the indices l, k refer to the absolute value of the largest non diagonal element in A as explained above. This helps us define and solve the quadratic equation

$$\tan^2\theta + 2\tau\tan\theta - 1 = 0 \tag{16}$$

where the solutions are given by

$$t = -\tau \pm \sqrt{1+\tau^2} \tag{17}$$

and now $t = \tan\theta$ is a common shorthand notation. With this, the angle for the rotation matrix is known and the orthogonal transformation can be computed.

This process is repeated iteratively until the Frobenius norm of the off diagonal elements is smaller than a fixed tolerance.

So how many rotations are typically needed?

Since the algorithm is based on zeroing out non diagonal elements in order to diagonalize the matrix, it should be safe to presume that at least $N^2 - N$ rotations for a matrix of size $(N \times N)$ are needed. But generally the orthogonal transformations often make previously zeroed out off diagonal elements non zero again at various iterations. So more than $N^2 - N$ rotations are required. In lack of more information, this additional term is assumed to be $(N^2 - N)/2$ or in other words at least half of the already zeroed out off diagonal elements have to be zeroed out again. This gives us a net expected number of rotations $N_{rot}(N)$ required for a $(N \times N)$ matrix

$$N_{rot}(N) = \frac{3}{2}N(N-1) \tag{18}$$

so for example a (10×10) matrix would require $N_{rot}(10) = 165$ rotations.

B. The finite difference method

The second order derivative of a function can be approximated by

$$u'' = -\frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} \tag{19}$$

where h is the chosen step length.

C. Discretization

By discretizing our differential equations and using the finite difference method, we can solve both equations by matrix equations of the form

$$A\hat{u} = \lambda\hat{u} \tag{20}$$

with the product $A\hat{u}$ defines the set of linear equations generated by the finite difference method. By making A a general Toeplitz matrix we can write A as $(N \times N)$ with the elements d_i, b_i, c_i

$$A = \begin{pmatrix} d_1 & b_1 & 0 & 0 & \cdots & 0 & 0 \\ c_1 & d_2 & b_2 & 0 & \cdots & 0 & 0 \\ 0 & c_2 & d_3 & b_3 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & c_{N-2} & d_{N-1} & b_{N-1} \\ 0 & 0 & 0 & \cdots & 0 & c_{N-1} & d_N \end{pmatrix} \tag{21}$$

In simplicity, (20) is an eigenvalue problem. Where $\hat{u}$ is a specific eigenvector (a solution to the equation) for a specific eigenvalue (set of constants) for our problem. Physically these represent different energy states of either the buckling beam system or the quantum system.

Discretizing the dimensionless parameter $\rho \to \rho_i$ giving $u(\rho) \to u(\rho_i) = u_i$ and for the buckling beam $\rho_i \in [0, \frac{L}{L}] = [0, 1]$ due to how we defined ρ.

The quantum system is a little bit more tricky, due to the fact that for a particle in spherical symmetric potential, r runs from zero to inifinity. Since we can not define infinity, we'll have to approximate it. We'll use 1-10 as a starting points due to the small scales of quantum systems. We can choose ρ to be in units of Ångstrøms [Å], where 1Å=0.1nm.

The matrix A becomes tridiagonal due to the finite difference method and we'll define the diagonal elements as d_i (i=1,2,...,N) and the diagonals above and below the main diagonal as b_i and c_i (i=1,2,...N-1) respectively.

The buckling beam problem solutions are given by

$$\lambda u_i = -\frac{u_{i-1} - 2u_i + u_i i + 1}{h^2} \tag{22}$$

while the quantum systems has an extra term due to the potential $\rho^2 u(\rho)$ which discretizes to

$$\lambda u_i = -\frac{u_{i-1} - 2u_i + u_i i + 1}{h^2} + \rho_i^2 u_i \tag{23}$$

considering that the indices i refer to the main diagonal of the matrix A we see that the eigenvalue problem of the quantum system is the same as for the buckling beam except that we need to add the values of the potential to the main diagonal of A. The same goes for the terms in the system of two electrons (11), except the extra term becomes

$$\omega_r^2 \rho_i^2 u_i + \frac{1}{\rho_i} \tag{24}$$

where ω_r is taken to represent the strength of the harmonic oscillator potential which the electrons are trapped within.

In order to make a general solver for similar equations such as this. A function which creates a Toeplitz matrix given an array of diagonal, above and below diagonal seems to be suitable. By using dynamic array allocations, the arrays for these diagonals are only needed in runtime very briefly.

D. Program Structure and Computations

This part describes firstly the program structure and secondly the computations which will be performed in this report.

The program for this computation will be written in terms of a hierarchy of functions, where a main function $JacobiEigenValueAlgorithm$ will be depend on several other functions (which in turn represent steps in the Jacobi eigenvalue algorithm). This hierarchy of functions is compactly collected in the $JacobiEigenvalSolver.h$ file.

The function hierarchy is divided into four function:

- 1 - Calculate value of maximum off diagonal element in matrix A $\rightarrow$ function $Max\_Off\_Diag\_Element$
- 2 - Find indices (k,l) of maximum element in the matrix A $\rightarrow$ function $Max\_Off\_Diag\_kl$
- 3 - Perform Jacobi rotation $\rightarrow$ function $JacobiRotation$
- 4 - Perform orthogonal matrix transformation $\rightarrow$ function $Update\_A$

all implemented in function $JacobiEigenValueAlgorithm$:

E. Main Algorithm

```
while $max|a_{ij}| > tol$ do
    if Orthogonality-Test = False then
        Break and print message
    else
        Perform Jacobi rotation
        Perform orthogonal transformation
        Update A
    end if
end while
```

F. Unit tests

Since our program consists of a large number of statements, functions and variables. It is a good idea to implement test functions so as to firstly confirm that the function runs as expected when initially written. And secondly to make sure it keeps executing as expected in the program.

Test functions are written for functions 1-3 and the main algorithm function $JacobiEigenValueAlgorithm$. The latter includes a test case for the matrix

$$\begin{pmatrix} 1 & 1 & -2 \\ 1 & 2 & 1 \\ -2 & 1 & -1 \end{pmatrix} \tag{25}$$

with the analytical eigenvalues $(2, \pm\sqrt{7})$.

G. Other test functions

One particularly useful function written in the program is the *CompareEigenvalueComputations* function, which takes two arrays as inputs where each contains the analytical eigenvalues (calculated by (12)) and the eigenvalues computed by the Jacobi algorithm.

Since the order of the eigenvalues in the arrays can be somewhat random, the function compares all values and increments a counter by +1 if it finds a match. Lastly the program checks whether the counter is equal to the length of either of the received arrays. If the check is successful within a certain tolerance, the eigenvalues are considered equal.

H. About repository and programs

There are three c++ programs

- Part 1 *JacobiEigenvalSolverBuck.cpp*
- Part 2 *JacobiEigenvalSolverSchr.cpp*
- Part 3 *JacobiEigenvalSolverQdots.cpp*

which are all linked to the header *JacobiEigenvalSolver.h*. In addition there are corresponding python programs

- Part 1 *JacobiEigenvalSolverNumpy.py*
- Part 2 *SchrPython.py*
- Part 3 *QdotsPython.py*

which produce the various results used in the report. There are also corresponding dat for storing the results of the c++ programs.

IV. RESULTS

A. Part 1: Buckling-Beam problem, comparison of eigenvalues computed by Jacobi algorithm and numpy.linalg.eig

The programs *JacobiEigenvalSolverBuck.cpp* and *JacobiEigenvalSolverNumpy.py* are run. Each write their own dat file *eigenvalues.dat* and *eigenvaluespython.dat* respectively. The values are collected from one execution and computation times logged for a series of 5 executions.

Tabell I: Table of computed eigenvalues with both Jacobi algorithm and numpy.linalg.eig function for a ($N = 8 \times N = 8$) matrix A

λ_i	Jacobi Algorithm	numpy.linalg.eig
1	192	192.000000
2	64	64.000000
3	248.28066	248.280655
4	7.7193445	7.719345
5	150.22697	150.226967
6	105.77303	105.773033
7	226.05369	226.053689
8	29.946311	29.946311

the number of decimals are left unequal intentionally to highlight the rounding differences.

If the decimal precision is set to match, then the eigenvalues are equal to that precision. The number of rotations reported by the program is 106, while the expected number of rotations, given by (18), is 84. But this was equation is expected to be an approximation.

Additionally as computed by the *CompareEigenvalueComputations* function, the eigenvalues from the Jacobi algorithm are equal to the analytical eigenvalues from (12) to a precision of 10^{-8}. The tolerance for the Jacobi algorithm was set to 10^{-16}. So essentially requiring the off diagonal elements to be zero for a 64-bit floating point machine precision.

Tabell II: Results of runtimes T_{Jac},T_{eig} in milliseconds [ms] for 5 runs of eigenvalue computations by Jacobi algorithm and by numpy.linalg.eig for ($N = 8 \times N = 8$) matrix A

	T_{Jac} [ms]	T_{eig} [ms]
	0.21	4.00
	0.20	4.00
	0.21	3.00
	0.20	4.00
	0.23	3.00
Mean + error in mean	0.21± 0.01	3.6± 0.24

B. Part 2: Schrodinger equation, energy level comparison

By experimenting with tolerances, there seems to be a requirement to reduce the tolerance of the orthogonal transformation function ϵ_V when the number of eigenvalues N is increased.

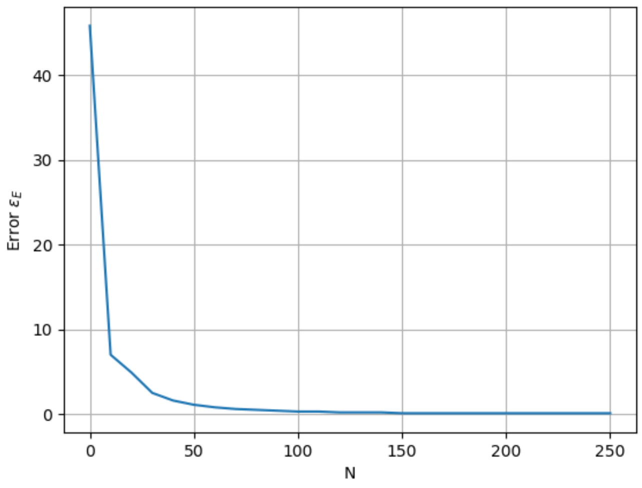
The error in the numerical energy values ϵ_E is calculated as the difference between the analytical eigenvalues calculated by (referer her) and the numerical are calculated by taking the norm of the difference vector between them.

The tolerance for the orthogonal transformation test is set to $\epsilon_V = 10^{-11}$ and the tolerance for when non diagonal elements in the Jacobi algorithm is considered zero $\epsilon_J = 10^{-16}$. Lastly $\rho_N = 10$.

Tabell III: Table with error ϵ_E values and number of eigenvalues calculated by the Jacobi rotation algorithm N. Only the ten lowest energy values are extracted from the error vector.

ϵ_N	N
10	45.8
30	4.9
50	1.6
70	0.8
90	0.5
110	0.3
130	0.2
150	0.2
200	0.1
300	0.0

Adding a plot of the results in the table.



Figur 1: Plot of error for ten lowest energy values.

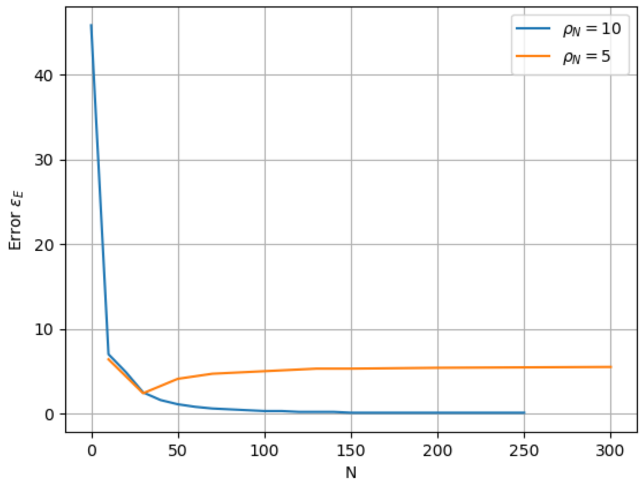
Now adding a table of the three lowest energy values in a table for three chosen N values, E_J are computed energy values by Jacobi algorithm, while E refers to the analytical values and are given in another table just below.

N	10	30	110
E_J	$\begin{pmatrix}1.343\\3.057\\5.529\end{pmatrix}$	$\begin{pmatrix}1.482\\3.411\\5.279\end{pmatrix}$	$\begin{pmatrix}1.499\\3.494\\5.484\end{pmatrix}$

Tabell IV: Table of analytical energy values E

E	1.5	3.5	5.5

Now lowering the approximation of infinity by changing $\rho_N = 5$. Adding plot of both results of error with $\rho_N = 10$ as well



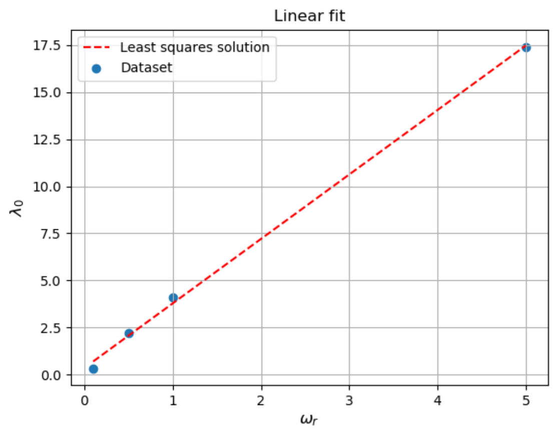
Figur 2: Plot of error ϵ_N as function of number of computed eigenvalues N for $\rho_N \in 5, 10$.

C. Part 3: Schrodinger equation, Quantum dots

Since we're only focusing on the ground state, we present a table of the computation results for the eigenvalues of varying potential strength ω_r with corresponding eigenvalue for the ground state λ_0. Keeping $\rho_N = 10$.

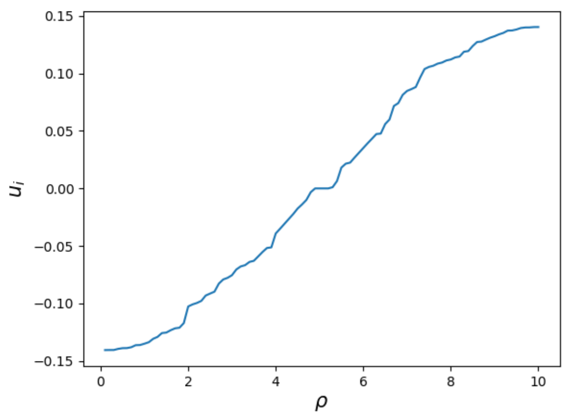
Tabell V: Table showing eigenvalues for ground state λ_0 with corresponding potential strength ω_0. For $N = 100$ eigenvalue calculation in the Jacobi algorithm.

ω_r	λ_0
0.1	0.3
0.5	2.2
1.0	4.1
5.0	17.4

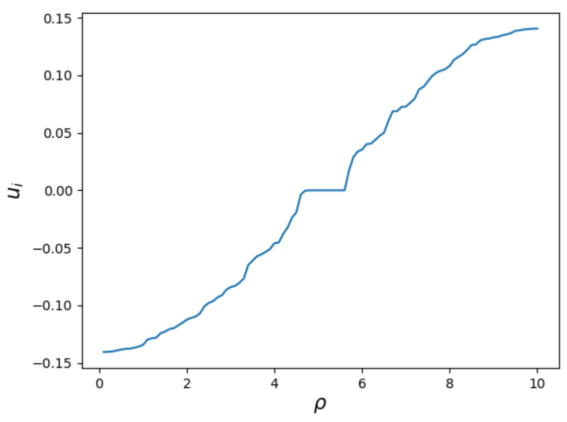


Figur 3: Least squares fit of λ_0 as function of ω_r.

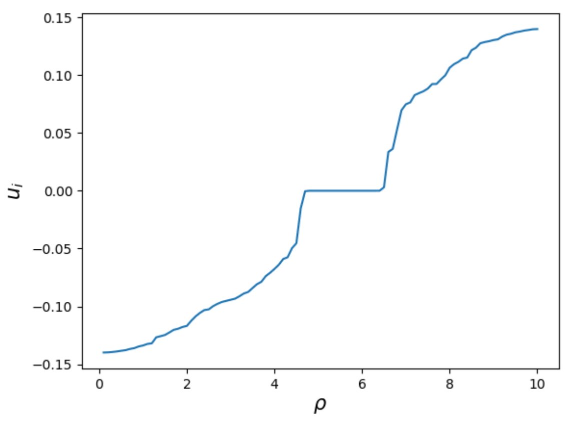
Now extracting eigenvector containing u_i corresponding to eigenvalue and plotting as a function of ρ_i



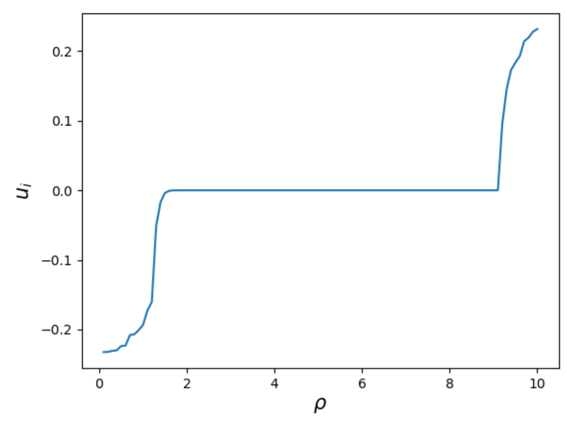
Figur 4: Plot of eigenvector corresponding to ground state $\lambda_0 = 0.3$ with oscillator potential $\omega_r = 0.1$.



Figur 5: Plot of eigenvector corresponding to ground state $\lambda_0 = 2.2$ with oscillator potential $\omega_r = 0.5$.



Figur 6: Plot of eigenvector corresponding to ground state $\lambda_0 = 4.1$ with oscillator potential $\omega_r = 1.0$.



Figur 7: Plot of eigenvector corresponding to ground state $\lambda_0 = 17.4$ with oscillator potential $\omega_r = 5.0$.

V. DISCUSSION

From table (I) we see that the eigenvalues correspond to the fixed tolerance, although the computation times as seen in table (II) seem to be in favour of the Jacobi algorithm. Dividing the mean values we can show that according to these computations, the written Jacobi algorithm runs about 17 times faster than numpy's linalg.eig function.

What was also observed when running the Jacobi algorithm several times was that our number of rotations function R_{rot} (18) was much better approximating the number of required rotations if changed to

$$N_{rot}(N) = 2N(N-1) \tag{26}$$

namely that every non diagonal element must be zeroed out twice.

In part 2 many test runs with variations of the tolerances for the orthogonal conservation ϵ_V and the zero tolerance for the off diagonal elements of A ϵ_J had to be testet in order for the energy values to be returned as sensible values. Thus having to decrease ϵ_V for increasing N seems to be a natural consequence of floating point operations. Since the Jacobi algorithm keeps zeroing out non diagonal elements and rewriting earlier zeroed out non diagonals with new number which contain errors. Accumulating more and more errors for each rotation.

Moving on, we can see that $\rho_N = 10$ was a presumably good approximation for infinity in the quantum case. In plot (1) we see the error in the first ten energy eigenvalues decrease rapidly with increasing N. It might not be a bad suggestion that the error goes as $\frac{1}{N}$ or even $\frac{1}{\sqrt{N}}$. A comment to table (III) when it comes to the error at $N = 300$. Due to the choice of significant digits this became rounded to zero, but it is (perhaps obviously) not exactly zero. So we shall interpret this as meaning practically zero. In other words, the accuracy of the eigenvalue computation increases with increasing N. If we consider the other much lower approximation to infinity we made, $\rho_N = 5$, we can consider (2), here the error starts out lower than for $\rho_N = 10$, reaches a minima at around $N = 30$, then increases very slowly (logarithmic perhaps) with increasing N. This might be explained by the fact that when ρ_N increases, the step size decreases. Which should give a more accurate solution, while a lower ρ_N tends to the opposite.

Finally in part 3, an analytical solution for the energy values were not available. So the computed eigenvalues have to be assumed correct or at least decent approximations. Table (V) shows an increasing eigenvalue for an increasing ω_r (harmonic oscillator potential strength). Since the computed eigenvalues are proportional to the energy eigenvalues of the system, it means that there is also an increase in the hamiltonian (total) energy of the system with an increased potential strength. This makes sense since the electrons are, i.e given more energy when the potential strength is increased. In the same way you're applying more potential energy (thereby increasing total energy) to yourself when you move up a hill. The least squares fit in (3) suggests a proportionality constant of 3.5 between the computed eigenvalues for the ground state λ_0 and ω_r. The plots (4) - (7) of the eigenstates show how the electrons have practically zero chance of existing in the center of the potential. The center in a physical sense would contain the cause of the potential, for instance protons. In this way we can interpret the increase in zero probability area which becomes very visible in plot (7) as an increase in the central charge, forcing the electrons to be more seperated.

VI. CONCLUDING REMARKS

By scaling differential equations, a common solver can be applied to several physical systems. For the particular systems we have found: writing a solver for the Jacobi algorithm in c++ may reduce runtime significantly when compared with other solvers. We've confirmed that this holds true for the numpy.linalg.eig eigenvalue solver. In part 2 we've found that a decent approximation to infinity for a quantum system is $\rho_N = 10$, resulting in a difference in numerical and analytical energy values which approaches zero for relatively large values of N. Although other untested choices of ρ_N might lead to better results.

In part 3 we've found that increasing the potential strength in a 2 particle quantum systems leads to a proportional increase of energy with a proportionality constant of 3.5.

VII. GITHUB-REPOSITORY

`https://github.com/nhofield/fys3150Projects.git`

VIII. REFERENCES

- `http://compphysics.github.io/ComputationalPhysics/doc/Projects/2018/Project2/pdf/Project2.pdf`
- `http://math.ecnu.edu.cn/~jypan/Teaching/books/2013%20Matrix%20Computations%204th.pdf`