FYS4150: Project 2

Jonathan Brakstad Waters

September 30, 2019

Abstract

In this article we show how to implement Jacobi's algorithm for solving eigenvalue problems numerically. We also apply this solver on the quantum mechanical case of two electrons in a three dimensional harmonic oscillator potential. We suggest that the repulsive Coulomb potential yields a sharper wave function than the one for two non-interacting electrons.

Introduction

The goal of this report is to present a numerical solver for eigenvalue problems, and use this solver to study the quantum mechanical case of electrons in a three dimensional harmonic oscillator.

I will start with a brief explanation of Jacobi's method for finding the eigenvalue of a symmetric, dense matrix, and my implentation of this method. I will comment on how this algorithm performs, and compare this to other eigenvalue solvers.

Then I will show how this solver can be used to investigate the properties of electrons in a 3D harmonic oscillator potential.

General Jacobi Solver

Description of the problem

Our initial motivation for making an eigenvalue solver, is equations on the form

$$\gamma \frac{d^2u(x)}{dx^2} = -Fu(x) \qquad\qquad u(0) = u(L) = 0$$

Such equations could model a number of different scenarios, for example a buckling beam. In that case, u represents the vertical displacement of the beam in the y-direction, L represents the length of the beam, and F represents a force applied at $(L, 0)$ in the direction towards the origin. γ is a constant parameter defined by various properties of the beam.

Now we can scale the equation, by introducing a variable $\rho = \frac{x}{L}$. Since x takes values in $[0, L]$, $\rho \in [0, 1]$, with $\rho(0) = \rho(1) = 0$. We can then reorder the equation and get:

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

If we then discretize ρ with step size h, we can approximate the second derivative as

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

which means we can rewrite our equation as $\mathbf{Tu} = \lambda\mathbf{u}$, with T being

$$\frac{1}{h^2}\begin{bmatrix} 2 & -1 & 0 & \cdots & \cdots & 0 \\ -1 & 2 & -1 & 0 & \cdots & 0 \\ 0 & -1 & 2 & -1 & \cdots & 0 \\ 0 & 0 & -1 & 2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & \cdots & 0 & -1 & 2 \end{bmatrix}$$

Note that the endpoints ($\rho = 0$ and $\rho = 1$) are not included in the matrix equation, since u is known in these points.

Now we are presented with an eigenvalue problem, and I will go on to explain how to solve this numerically, using Jacobi's algorithm.

Implementation of Jacobi's Method

Details on the theory and algebra behind Jacobi's algorithm can be found in the referenced lecture notes by Hjorth-Jensen (2015). The basic idea is to perform a number of orthogonal transformations of the form

$$\mathbf{T} \rightarrow \mathbf{T}' = \mathbf{QTQ}^T$$

and end up with a diagonal matrix. Since eigenvalues are conserved under orthogonal transformations (Hjort-Jensen, 2015), the eigenvalues can then be easily read off as the diagonal elements of our transformed matrix.

In our case we will use a $\mathbf{Q}$ that represents a rotation around an axis, with sine and cosine functions at the matrix elements Q_{kk}, Q_{ll}, Q_{kl} and Q_{lk}. We can choose the rotation angle so that the element $\mathbf{T}'_{kl}$ (and $\mathbf{T}'_{lk}$) become equal to zero. It can be shown that for every such transformation we perform, the frobenious norm ($\sum_{ij} |a_{ij}|^2$) of the offdiagonal elements reduces, and the frobenious norm of the diagonal elements increases, while the total frobenious norm remains unchanged - in a sense we are "moving stuff from the offdiagonal to the diagonal".

In total, this means that after a certain number of transformations, the offdiagonal elements will all be essentially zero, and the diagonal elements will be the eigenvalues of the matrix. To perform

one such transformation, we define

$$s = \sin\theta$$

$$c = \cos\theta$$

and find these values by solving:

$$\tau = \frac{T_{ll} - T_{kk}}{2T_{kl}}$$

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

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

$$s = tc$$

Our new matrix elements are then given by:

$$\text{If } i \neq k, i \neq l\text{:}$$

$$T'_{ii} = T_{ii}$$

$$T'_{ik} = T'_{ki} = T_{ik}c - T_{il}s$$

$$T'_{il} = T'_{li} = T_{il}c + T_{ik}s$$

$$\text{Else:}$$

$$T'_{kk} = T_{kk}c^2 + T_{ll}s^2 - 2T_{kl}cs$$

$$T'_{ll} = T_{ll}c^2 + T_{kk}s^2 + 2T_{kl}cs$$

If we perform these calculations a number of times, we will get the eigenvalues of the initial matrix T. To find the eigenvectors, we start with a orthogonal basis

$$\mathbf{R} = \mathbf{I} = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix}$$

and let our transformations work on this basis:

$$R'_{ik} = cR_{ik} - sR_{il}$$

$$R'_{il} = cR_{il} + sR_{ik}$$

It is easy to show that the orthogonality and dot product of the basis is preserved under this orthogonal transformation:

$$\begin{aligned}\mathbf{R}'_i &= \mathbf{Q}\mathbf{R}_i \\ \mathbf{R}'^T_j \mathbf{R}'_i &= (\mathbf{Q}\mathbf{R}_j)^T \mathbf{Q}\mathbf{R}_i = \mathbf{R}^T_j \mathbf{Q}^T \mathbf{Q}\mathbf{R}_i \\ &= \mathbf{R}^T_j \mathbf{I}\mathbf{R}_i = \mathbf{R}^T_j \mathbf{R}_i = \delta_{ij}\end{aligned}$$

. ■

I have implemented this algorithm in the function `eigenvalues(T,eps)`. The function takes a matrix and a tolerance as input, and returns a matrix with the eigenvectors $\mathbf{D}_i$, corresponding eigenvalues, and also the number of transformations performed, `N_trans`:

$$\begin{bmatrix} \mathbf{D}_1 & \mathbf{D}_2 & \cdots & \mathbf{D}_N & \mathbf{0} \\ \lambda_1 & \lambda_2 & \cdots & \lambda_N & \texttt{N\_trans} \end{bmatrix}$$

Note that the eigenvalues are not returned in a sorted form, and must be sorted outside the function. The code for this (and all other functions) can be found in the `p2_functions`, at the github-location given at the end of the report.

Results and discussion

For a tridiagonal Toeplitz Matrix (like the one we have from our eigenvalue problem), the eigenvalues are given by the analytical expression:

$$\lambda_j = d + 2a\cos\left(\frac{j\pi}{N+1}\right)\, j = 1, 2, \ldots N$$

This provides an excellent possibility for testing our implentation of the Jacobi algorithm. I'm running my solver for `N = 5`, and comparing with the exact solution I find that my results are accurate to a precision of $\pm 10^{-8}$. The test function, which also test the preservation of the Frobenious norm, can be found in the program `p2_tests.ccp`.

Now, having asserted that the solver works, it is interesting to look at how efficient it is. I run the solver for a number of values for N, and interpolate my results for the number of necessary transformations (figure 1) and the necessary runtimes (figure 2). I use Geogebra for a quick indication

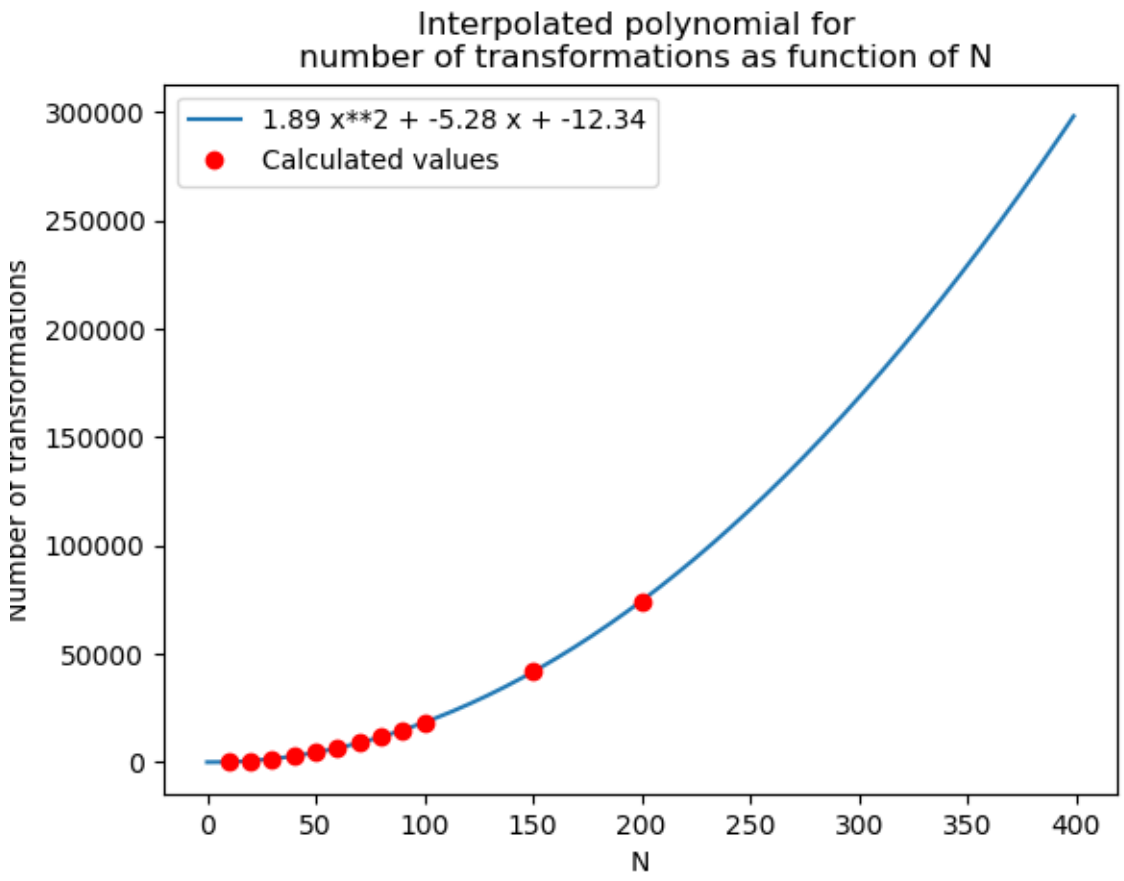

Figure 1: *Number of required transformations in the Jacobi Solver, as function of N, the dimensionality of the matrix. The red dots are actual computed values, the blue line is an interpolation*

of what interpolation function is best, and then I use matplotlib.pyplot to plot my results together with the interpolation.

In figure 1, you can see an interpolation of the number of necessary transformations, and we can see that it scales like N^2. The runtime, however, seems to scale like N^4 (figure 2), meaning that for a dimensionality of 400×400, you would require close to half an hour to run. This matches well with experience.

I have briefly compared these running times with Armadillo's in-built solver, `eig_sym`. I do not have numerical data, but suffice to say that the armadillo solver runs in "a blink of an eye" for N up to 1000 and therefore seems like a much better algorithm than the Jacobi solver. The precision seems to be almost identical for the two different solvers.

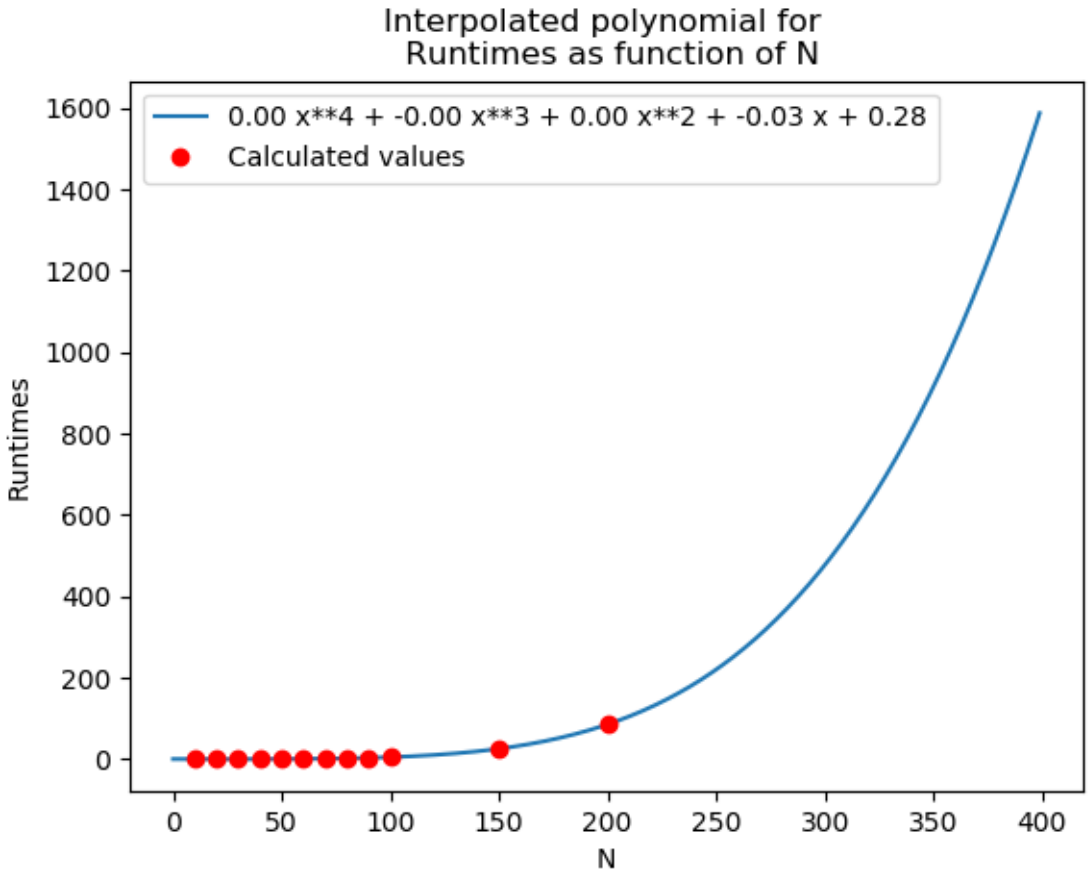

Figure 2: *Runtime for the Jacobi Solver, as function of N, the dimensionality of the matrix. The red dots are actual computed values, the blue line is an interpolation. The coefficients of the polynomial are approximately* $7.98e-08$, $-8.74e-06$, $8.31e-04$, $-2.91e-02$ *and* $2.80e-01$.

Electrons in 3D Harmonic Oscillator

Description of the problem

Now that we have developed a functional (albeit slow) eigenvalue solver, we can apply it to a proper physics problem, namely the situation of two electrons in a 3D harmonic oscillator. In the following, I rely heavily on Hjorth-Jensen (2019) for presenting the equations we need for this problem. Look there for details.

In spherical coordinates, the radial part of the Schroedinger equation for two non-interacting electrons in a harmonic oscillator can be written as follows:

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

with $k = m\omega^2$.

We now introduce the relative coordinate $\mathbf{r} = \mathbf{r}_1 - \mathbf{r}_2$ and the center-of-mass coordinate $\mathbf{R} = 1/2(\mathbf{r}_1 + \mathbf{r}_2)$. With these new coordinates, the radial Schroedinger equation reads

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

The equations for r and R can be separated via the ansatz for the wave function $u(r, R) = \psi(r)\phi(R)$ and the energy is given by the sum of the relative energy E_r and the center-of-mass energy E_R, that is $E^{(2)} = E_r + E_R$. This means that we can remove the R-dependent terms from our equation. Then we add the repulsive Coulomb interaction between the two electrons:

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

yielding the equation:

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

Similarly to the buckling beam equation, this equation can be scaled and made dimensionless. See Hjorth-Jensen for details. The resulting equation reads:

$$-\frac{d^2}{d\rho^2}\psi(\rho) + \omega_r^2\rho^2\psi(\rho) + \frac{1}{\rho} = \lambda\psi(\rho) \qquad\qquad u(0) = u(\infty) = 0$$

Where the frequency w_r is a parameter to be adjusted.

Now, this equation is quite similar to the one we had for the buckling beam equation, but it has different elements along the main diagonal. We can therefore apply our Jacobi solver to the problem.

Method

We start by looking at the case where $w_r = 1$ and there's no interaction between the electrons, that is, we remove the term $\frac{1}{\rho}$. Curiously, in its scaled form, this equation is exactly the same as for only one electron in the harmonic oscillator. This makes sense, because the two cases should only differ by a factor 2 for the energy levels.

The first thing to note about this problem, is that the code used for the buckling beam needs to be changed to allow for a vector d of diagonal elements that are not all the same. This could be done by writing d into a function and calling this, but I have chosen to simply store the diagonal elements in a vector. This poses few problems, as N will be relatively small, and storage space is plenty.

Secondly, we need to account for the fact that ρ now ranges from 0 to ∞, and we need to choose an upper limit ourselves. To investigate sensible values for ρ_{max}, I have run my program for a variety of values for N (up to 500) and ρ_{max} (4, 5, 7, 10 and 15), and compared my results to the first four analytical values of λ, namely 3, 7, 11 and 15. Table 1 shows what values of ρ_{max} and N produce the least error for the four eigenvalues. This analysis is made in the program `"Plotting Pro2.py"`. Having chosen the best possible values for ρ_{max} and N, we can plot the eigenstates of the electron and study the results.

λ_1	$\rho_{max} = 4$	$N = 300$
λ_2	$\rho_{max} = 5$	$N = 400$
λ_3	$\rho_{max} = 5$	$N = 400$
λ_4	$\rho_{max} = 5$	$N = 200$

Table 1: *Ideal values for N and ρ_{max}:*

It might be interesting to note that the largest N does not necessarily produce the best result - this is probably due to numerical and computational variations. After a brief look at the whole error table (which can be found in the appendix), I decided to plot the first four eigenfunctions for $\rho_{max} = 5$ and $N = 400$. These can be seen in figure 3.

Having checked our eigenvalues against the analytical results, we then investigate the behaviour of the ground state for different frequencies w_r, both for the non-interacting and the interacting cases. During this process, I found that ideal values for ρ_{max} changed considerably with different values of w_r. To find good values, I have run my program for a number of different ρ_{max} values, and seen when

the ground state energy seems to reach convergence. Also, I have run my program for the interacting case for frequencies used by Taut (1993). Comparisons of my results with his gives a good indication of what values for ρ_{max} we should use. In the end, for $w_r = [0.01, 0.5, 1, 5]$, I have used $N = 400$, and ρ_{max} as follows:

w_r	ρ_{max}
0.01	30
0.5	10
1	10
5	10

Table 2: *ρ_{max} values for different w_r*

In figure 4 you can see the different behaviours of the non-interacting and interacting cases.

Results and Discussion

Let us look first at the plot for the non interacting eigenfunctions, figure 3. We see that for higher energy levels, the electron tends to be at higher values of ρ (and, ultimately, r). Also, as we would expect, higher energy states have more nodes in the wave function.

Now we shift our focus to the ground state only. If we vary the strength of the oscillator potential, ω_r, figure 4 shows that a smaller ω_r leads to a broader wavefunction. This makes sense. If the potential is weaker, one would imagine an electron being able to make its way further out before being pulled back. Consequently, it is more probable to find the electron at greater values of ρ.

When the strength of the potential increases, the electron should find it harder to 'escape', which is indeed the case. For larger ω_r, we can see that the wave function moves towards the origin, meaning that the electron is more strongly confined to a smaller radius. This raises the question if one could use a sufficiently strong potential to "pinpoint" the location of an electron, in the origin.

Let us now 'turn on' the interacting Coulomb potential. This yields the ground states depicted by the red graph in figure 4. Note that as the harmonic oscillator potential gets stronger, the Coulomb interaction gets more insignificant in comparison. For $\omega_r > 1$, the difference between the two cases is almost indiscernable.

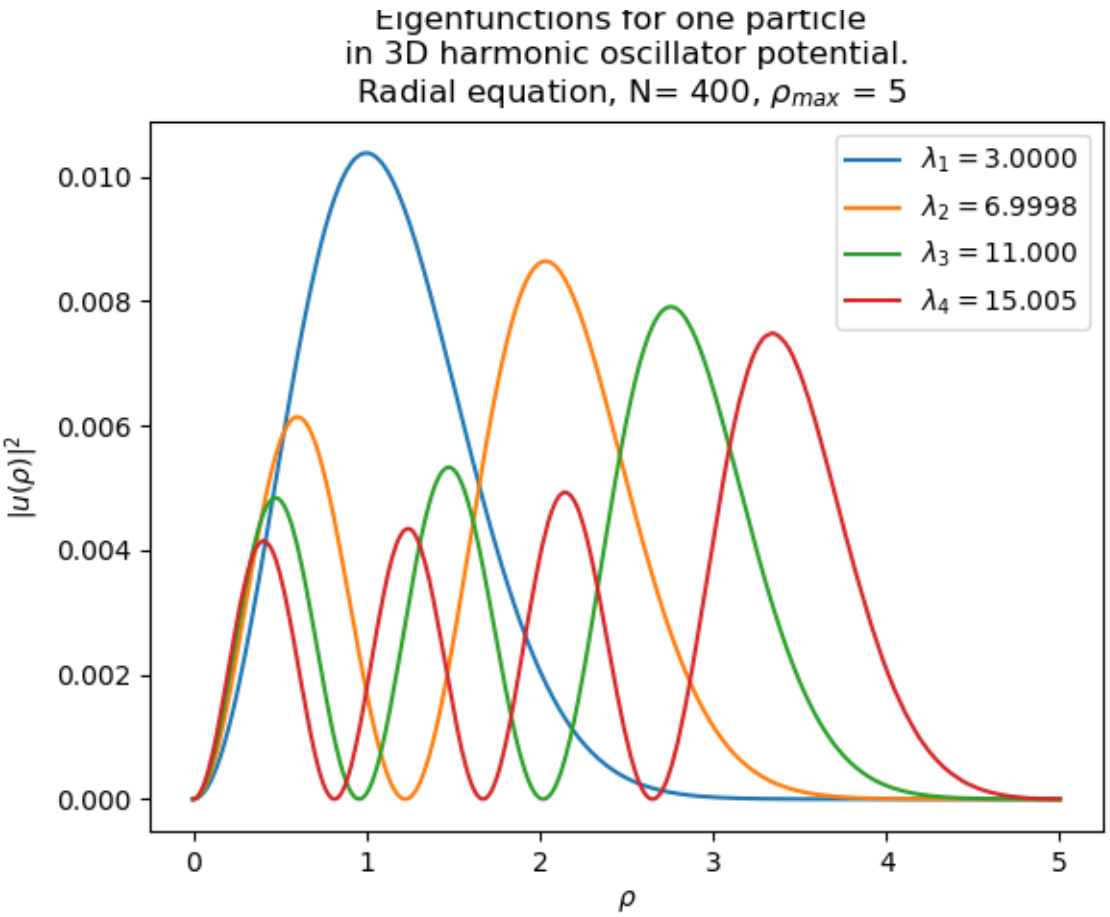

Figure 3: *The first four energy states for two non-interacting electrons in a harmonic oscillator. Note the number of nodes for each energy level.*

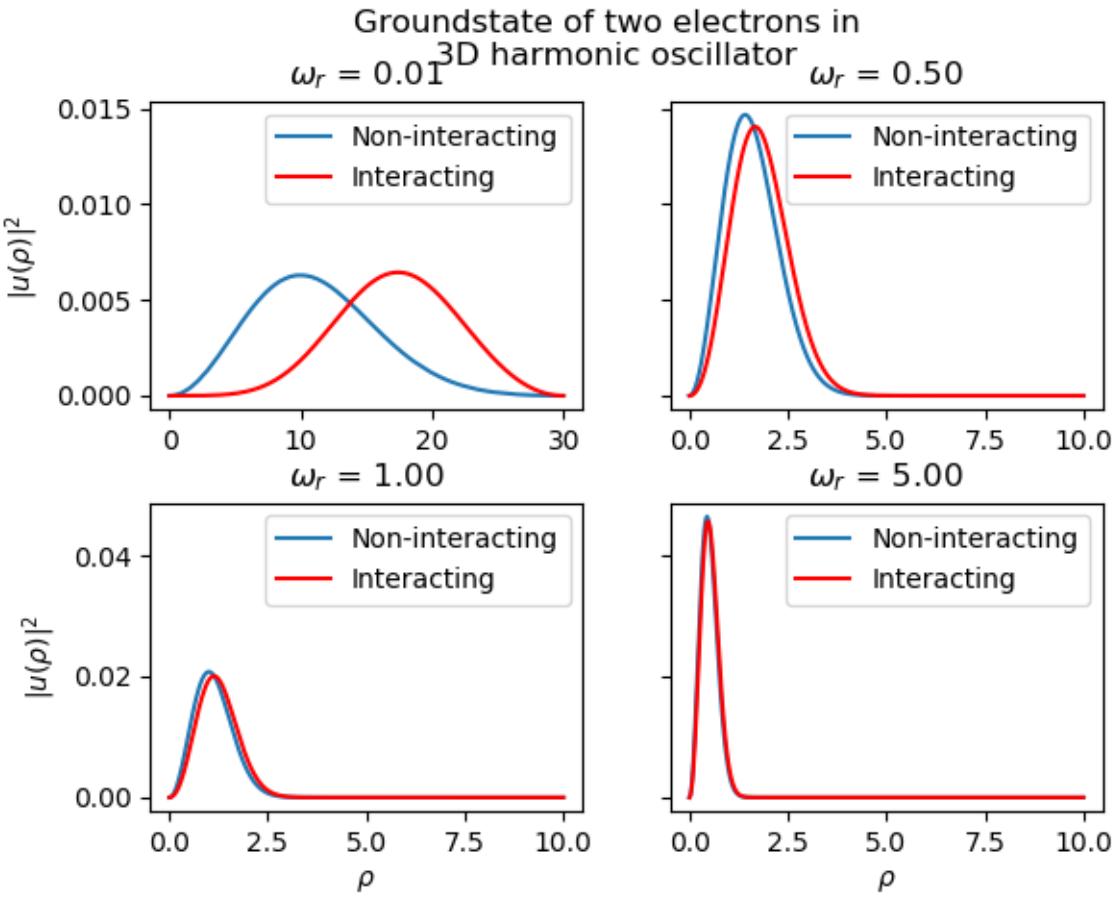

Figure 4: *Ground state for two electrons in a 3D harmonic oscillator, both with and without interaction. Note how the ground state varies depending on the strength of the potential, ω_r.*

For $\omega_r < 1$, we can see that the ground state for the interacting electrons is shifted towards greater ρ. This could be interprated as the electrons pushing each other away due to the repulsive Coulomb force. This interpretation is further enforced by the width of the wave functions. For higher ω_r, the interacting ground state is ever so slightly lower in amplitude than the non-interacting one. This means that it must also be slightly wider, to preserve normalization. Hence, the electrons take up slightly more space when they interact than when they don't, as expected.

However, a really close look at the case where $\omega_r = 0.01$ tells us that the interacting ground state is actually slightly taller in amplitude, and therefore less wide than the non-interacting case. This could indicate that for weaker potentials, the two repulsing electrons are actually closer to each other than they would be with no interaction. This seems to make no sense, and I would hesitate to draw conclusions before having tested for a number of ω_r in the range $[0.01, \cdots, 0.5]$, but it is certainly an interesting observation.

Conclusion

This report describes the implentation of Jacobi's algorithm for solving eigenvalue problems, and the application of this algorithm on the quantum mechanical problem of two electrons in a three dimensional harmonic oscillator potential. In the process, I have shown how the Jacobi Solver is quite slow, in fact the computing time scales as N^4, where N is the dimensionality of the matrix in question.

Looking at the electrons in the harmonic oscillator, I have looked at two cases. One where the two electrons do not interact with each other, and one where they do. For both cases I have varied the strength of the oscillator potential and looked at the differences in behaviour. I have shown how solving these problems only requires minor tweaks to the main diagonal of our second derivative matrix. Also, the acquired results have been compared to known results, and found to match quite well.

Having tested the solver on known cases, I applied my solver to a number of new potential strenghts. The most interesting (albeit uncertain) finding is that for weaker potentials, interacting electrons seem to be closer to each other than non-interacting ones, despite the repulsive Coulomb potential. Closer investigation of this phenomenon could be grounds for future research.

References

Hjorth-Jensen, M (2015) Lecture Notes. Department of Physics, University of Oslo. Downloaded 07.09.2019 from:
https://github.com/CompPhysics/ComputationalPhysics/tree/master/doc/Lectures

Hjorth-Jensen, M (2019), Project 2. Department of Physics, University of Oslo. Downloaded 28.09.2019 from:
https://github.com/CompPhysics/ComputationalPhysics/tree/master/doc/Projects/2019/Project2

Taut, M (1993). Two electrons in an external oscillator potential: Particular analytic solutions of a Coulomb correlation problem. Physical Review, volume 48

Developed Code files

This report, code files and selected output files can be found in github repository at:
https://github.com/HolyWaters95/FYS4150_Project_2

C++ programs:

N_trans_runtimes.cpp

quantum.cpp

quantum2particles.cpp

p2_functions.cpp

P2_tests.cpp

Python programs:

Plotting_Pro2.py

Py_Functions.py

Appendix

Complete Errortable:

Errors for lambda 1 = 3:

	N = 100	N = 200	N = 300	N = 400	N = 500
rho_max = 4	0.0005	0.0001	0.0000	0.0000	0.0000
rho_max = 5	0.0008	0.0002	0.0001	0.0000	0.0000
rho_max = 7	0.0015	0.0004	0.0002	0.0001	0.0001
rho_max = 10	0.0031	0.0008	0.0003	0.0002	0.0001
rho_max = 15	0.0070	0.0018	0.0008	0.0004	0.0003

Errors for lambda 2 = 7:

	N = 100	N = 200	N = 300	N = 400	N = 500
rho_max = 4	0.0009	0.0028	0.0031	0.0032	0.0033
rho_max = 5	0.0039	0.0010	0.0004	0.0002	0.0002
rho_max = 7	0.0077	0.0019	0.0009	0.0005	0.0003
rho_max = 10	0.0157	0.0039	0.0017	0.0010	0.0006
rho_max = 15	0.0353	0.0088	0.0039	0.0022	0.0014

Errors for lambda 3 = 11:

	N = 100	N = 200	N = 300	N = 400	N = 500
rho_max = 4	0.0720	0.0770	0.0780	0.0780	0.0790
rho_max = 5	0.0090	0.0020	0.0010	0.0000	0.0000
rho_max = 7	0.0190	0.0050	0.0020	0.0010	0.0010
rho_max = 10	0.0380	0.0100	0.0040	0.0020	0.0020
rho_max = 15	0.0860	0.0210	0.0100	0.0050	0.0030

Errors for lambda 4 = 15:

	N = 100	N = 200	N = 300	N = 400	N = 500
rho_max = 4	0.5720	0.5840	0.5860	0.5860	0.5870
rho_max = 5	0.0120	0.0010	0.0040	0.0050	0.0050

```
rho_max =  7 |  0.0350  |  0.0090  |  0.0040  |  0.0020  |  0.0010
rho_max = 10 |  0.0710  |  0.0180  |  0.0080  |  0.0040  |  0.0030
rho_max = 15 |  0.1610  |  0.0400  |  0.0180  |  0.0100  |  0.0060
```