Project 2 - FYS4150

Trude Hjelmeland, Oda Langrekken and Jostein Brændshøi

Department of Physics
University of Oslo

October 2, 2017

Abstract

We use the Jacobi eigenvalue method to solve the Schrödinger equation for an electron in a harmonic potential as well as for two interacting electrons, and thus find the energies and wavefunctions of the electrons. As we know the exact result for the former case, we find that the numerical method finds consistent eigenvalues. However we will see that the algorithm is not very efficient.

The address `https://github.com/jostbr/FYS4150-Projects/tree/master/Project2` on GitHub is associated with this project. Here one can find all code used in the project. This includes C++ source files containing the core of the project, header files, but also Python plotting scripts for result visualization. There are also available benchmark results from running the code as well as LaTeX source for this PDF.

1 Introduction

According to Wikipedia [1] physics "is the natural science that involves the study of matter and its motion and behavior through space and time". In classical mechanics the motion of a system through space and time is described by its position $\vec{r}$ which can be determined by applying Newton's second law: $\vec{F} = m\vec{a}$. In quantum mechanics, the state of a system is described by its wavefunction Ψ, and we get it by solving the Schrödinger equation [2]

$$-\frac{\hbar^2}{2m}\nabla^2\Psi + V\Psi = i\hbar\frac{\partial\Psi}{\partial t}$$

The Schrödinger equation can be solved analytically for a couple of simple systems, among them the hydrogen atom. For more complex systems, take the helium atom as an example, the equation is analytically unsolvable. Fortunately for us physicists, the equation can often be simplified and the solution approximated by numerical algorithms.

In this project we have developed a general eigenvalue solver utilizing Jacobi's rotational algorithm to solve the Schrödinger equation for a electron in a confining potential, visualized as a quantum dot or an electron in a box with infinite walls. We will further expand our model to include the repulsive coulomb forces acting between two electrons confined in a three dimensional harmonic oscillator potential. We will show that these are essentially eigenvalue problems on a tridiagonal matrix form which we can solve numerically utilizing Jacobi's rotational algorithm.

First we will examine some of the theory used in this project, most notably the solution of the Schrödinger equation and the concept of similarity transformations in linear algebra. We then move on to discuss how the Schrödinger equation can be reduced to an eigenvalue problem through discretization and how this problem can be solved by using the Jacobi method. Next we introduce some of the unit tests used to check the functionality of our program. We then present our results, both for the one-electron and the two-electron case. Finally we offer some concluding remarks about solving the Schrödinger equation numerically using the Jacobi method.

2 Theory

2.1 Schrödinger equation

The evolution of a quantum mechanical system is described by the Schrödinger equation [2]

$$-\frac{\hbar^2}{2m}\nabla^2\Psi + V\Psi = i\hbar\frac{\partial\Psi}{\partial t} \tag{1}$$

If the potential is independent of time, $V = V(\vec{r})$, separation of variables lead to the time-independent Schrödinger equation $\hat{\mathcal{H}}\psi_i = E_i\psi_i$ where $\hat{\mathcal{H}}$ is the Hamiltonian, the quantum mechanical energy operator

$$\hat{\mathcal{H}} = -\frac{\hbar^2}{2m}\nabla^2 + V$$

and E_i is the energy of the system described by the wave function ψ_i. The Schrödinger equation (1) has been reduced to a eigenvalue problem, where the energy E_i of a state ψ_i can be found by acting on the state with the Hamiltonian. The energies E_i are the eigenvalues of the Hamiltonian

matrix, and the states ψ_i are their corresponding eigenvectors.

In this project we will look at the radial part of the time-independent Schrödinger equation for an electron moving in a three-dimensional harmonic oscillator potential. Assuming spherical symmetry, the equation to be solved is [5]

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

Here E and $V(r)$ are the energy and potential of the harmonic oscillator respectively.
The potential of an harmonic oscillator is on the form $(1/2)kr^2$ with $k=m\omega^2$. Here m is the electron mass, k is a force constant and ω is the oscillator frequency. The energies are generally determined by [5]

$$E_{nl}=\hbar\omega\left(2n+l+\frac{3}{2}\right), \text{for } n=0,1,2,\dots \text{and } l=0,1,\dots n-1 \tag{3}$$

but we will limit ourselves to the case where the quantum number for the orbital angular momentum of the electron is zero, specifically that $l=0$. The spherical coordinate transformation means that $r\in[0,\infty)$, and we substitute $R(r)$ in equation (2) with $(1/r)u(r)$ where $u(r)$ is the wave function. We use the boundary conditions $u(0)=0$ and $u(\infty)=0$. The equation now reads [5]

$$-\frac{\hbar^2}{2m}\frac{d^2}{dr^2}u(r)+V(r)u(r)=Eu(r). \tag{4}$$

We want to scale our equations by introducing a dimensionless variable ρ instead of our position variable r. We do so by defining a dimensionless variable $\rho=(1/\alpha)r$, where the constant α has dimensions of length. Inserting this in equation (4) gives [5]

$$-\frac{\hbar^2}{2m\alpha^2}\frac{d^2}{d\rho^2}u(\rho)+V(\rho)u(\rho)=Eu(\rho). \tag{5}$$

Inserting the harmonic oscillator potential $V(\rho)=(1/2)k\alpha^2\rho^2$ we get [5]

$$-\frac{\hbar^2}{2m\alpha^2}\frac{d^2}{d\rho^2}u(\rho)+\frac{k}{2}\alpha^2\rho^2u(\rho)=Eu(\rho). \tag{6}$$

We want to simplify the equation (6), and get it on the form [5]

$$-\frac{d^2}{d\rho^2}u(\rho)+\rho^2u(\rho)=\lambda u(\rho). \tag{7}$$

We obtain equation (7) by first multiplying both sides of equation (6) with $2m\alpha^2/\hbar^2$ and defining λ as [5]

$$\lambda=\frac{2m\alpha^2}{\hbar^2}E, \tag{8}$$

This allows us to fix the constant α so that [5]

$$\alpha=\left(\frac{\hbar^2}{mk}\right)^{1/4}. \tag{9}$$

We move on to the interacting case, but will come back to the above in section 3.1.

2.2 The two-electron Schrödinger equation

So far our system has consisted of only one electron. If we want to study a two-electron system, the situation gets a little more complex. The two electrons will still be in a harmonic oscillator potential, but they now in addition interact with each other through Coulomb forces. If we forget about the Coulomb potential for a little while, the two-electron wave equation is [3]

$$\left(-\frac{\hbar^2}{2m}\frac{d^2}{dr_1^2} - \frac{\hbar^2}{2m}\frac{d^2}{dr_2^2} + \frac{1}{2}kr_1^2 + \frac{1}{2}kr_2^2\right)u(r_1, r_2) = E^{(2)}u(r_1, r_2) \tag{10}$$

where $E^{(2)} = E_1 + E_2$ is the energy of the two electrons.

We now introduce the relative coordinate $\vec{r} = \vec{r}_1 - \vec{r}_2$ and the center-of-mass coordinate $R = \frac{1}{2}(\vec{r}_1 + \vec{r}_2)$. The Schrödinger equation now reads

$$\left(-\frac{\hbar^2}{m}\frac{d^2}{dr^2} - \frac{\hbar^2}{4m}\frac{d^2}{dR^2} + \frac{1}{4}kr^2 + kR^2\right)u(r, R) = E^{(2)}u(r, R) \tag{11}$$

We assume the wave function can be separated into a function of r and a function of R, $u(r, R) = \psi(r)\phi(R)$. We are only interested in the r-dependent equation, so equation (11) becomes

$$\left(-\frac{\hbar^2}{m}\frac{d^2}{dr^2} + \frac{1}{4}kr^2\right)\psi(r) = E_r\psi(r) \tag{12}$$

Now that we have an equation for two electrons in a harmonic oscillator potential, we can add the repulsive Coulomb potential. The Coulomb potential between an electron at position $\vec{r}_1$ and an electron at position $\vec{r}_2$ is

$$V(\vec{r}_1, \vec{r}_2) = \frac{\beta e^2}{\vec{r}_1 - \vec{r}_2}$$

where e is the electron charge and β is a constant. Using our relative coordinate r , we have

$$V(\vec{r}_1, \vec{r}_2) = \frac{\beta e^2}{r}$$

Inserting this into (12) we get the Schrödinger equation for two interacting electrons in a harmonic oscillator potential

$$\left(-\frac{\hbar^2}{m}\frac{d^2}{dr^2} + \frac{1}{4}kr^2 + \frac{\beta e^2}{r}\right)\psi(r) = E_r\psi(r) \tag{13}$$

We again introduce the dimensionless variable ρ, and our scaled equation is

$$\left(-\frac{\hbar^2}{m\alpha^2}\frac{d^2}{d\rho^2} + \frac{1}{4}k\alpha^2\rho^2 + \frac{\beta e^2}{\alpha\rho}\right)\psi(\rho) = E_\rho\psi(\rho) \tag{14}$$

We now define a new frequency

$$\omega_r^2 = \frac{1}{4}\frac{mk}{\hbar^2}\alpha^4 \tag{15}$$

and be requiring $m\alpha\beta e^2/\hbar^2 = 1$, we have

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

By also defining

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

our wave equation reads

$$-\frac{d^2}{d\rho^2}\psi(\rho) + \omega_r^2\rho^2\psi(\rho) + \frac{1}{\rho}\psi(\rho) = \lambda\psi(\rho) \tag{16}$$

We now move on to some linear algebra, but as with the single electron case, we will come back to the above in section 3.1.

2.3 Some linear algebra

From linear algebra [3] we know that if we have a real symmetric matrix, meaning that $\mathbf{A} \in \mathbb{R}^{n\times n}$, which have n distinct or not eigenvalues $\lambda_1, ..., \lambda_n$ it is possible, through similarity transformations on the original matrix $\mathbf{A}$, to obtain the eigenvalues as the non zero elements found on the leading diagonal of a diagonal matrix $\mathbf{D}$. So if $\mathbf{A}$ is real and symmetric then a real orthogonal matrix $\mathbf{S}$ exists such that $\mathbf{S}^T\mathbf{A}\mathbf{S} = \mathbf{D}$. The statement that $\mathbf{B}$ is a similarity transform of $\mathbf{A}$ is true [3] if $\mathbf{B} = \mathbf{S}^T\mathbf{A}\mathbf{S}$ where the condition $\mathbf{S}^T\mathbf{S} = \mathbf{S}^{-1}\mathbf{S} = \mathbf{I}$, where $\mathbf{I}$ is the identity matrix, is met. So the strategy for the upcoming algorithm will be to preform a series of similarity transforms (where each transformation moves the matrix closer to diagonal from) on matrix $\mathbf{A}$ so that

$$\mathbf{S_N}^T \cdots \mathbf{S_1}^T\mathbf{A}\mathbf{S_1} \cdots \mathbf{S_N} = \mathbf{D} \tag{17}$$

and $\mathbf{D}$ then contains the eigenvalues along the diagonal. But for this to work, the eigenvalues must stay the same after a similarity transformation.

An important property of a similarity transformation is that the eigenvalues are conserved even though the eigenvectors are changed [3]. To see this we look at a general eigenvalue problem on the form

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{x} \tag{18}$$

and a similarity transformed matrix $\mathbf{B}$, more accurate that $\mathbf{B} = \mathbf{S}^T\mathbf{A}\mathbf{S}$. If we now multiply (18) with $\mathbf{S}^T$ from the left and insert $\mathbf{S}^{-1}\mathbf{S} = \mathbf{I}$ between $\mathbf{A}$ and $\mathbf{x}$ we have that

$$(\mathbf{S}^T\mathbf{A}\mathbf{S})(\mathbf{S}^T\mathbf{x}) = \lambda\mathbf{S}^T\mathbf{x} \tag{19}$$

which we immediately recognize is the same as

$$\mathbf{B}(\mathbf{S}^T\mathbf{x}) = \lambda\mathbf{S}^T\mathbf{x} \tag{20}$$

It is now clear to see that the eigenvalue λ of matrix $\mathbf{A}$ is also a eigenvalue of $\mathbf{B}$ but that the eigenvector of $\mathbf{B}$ equal $\mathbf{S}^T\mathbf{x}$ is not the same as the eigenvector of $\mathbf{A}$ equal to $\mathbf{x}$. We could repeat this argument for a series of such transforms and generally we then have that the eigenvalues are preserved for an arbitrary number of such transforms. This is very important for the algorithm below to make sense and produce correct results.

One additional theoretical result worth visiting before talking about the algorithm, regards the fact that a similarity transformation preserves orthogonality and the dot product. In order to prove this we look at a given basis $\mathbf{v}_i \in \mathbb{R}^n$ of vectors

$$\mathbf{v}_i = \begin{bmatrix} v_{i1} \\ \vdots \\ v_{in} \end{bmatrix}$$

and assume that the basis is orthogonal, i.e.

$$\mathbf{v}_j^T \cdot \mathbf{v}_i = \delta_{ij} \ , \qquad \delta_{ij} = \begin{cases} 1 \ , \ i = j \\ 0 \ , \ i \neq j \end{cases}$$

Then we define an orthogonal matrix $\mathbf{U} \in \mathbb{R}^{n \times n}$ in the orthogonal transformation $\mathbf{w}_i = \mathbf{U}\mathbf{v}_i$. Using matrix-vector multiplication properties, the transformed dot product evolves as

$$\begin{aligned} \mathbf{w}_j^T \cdot \mathbf{w}_i = (\mathbf{U}\mathbf{v}_j)^T \cdot (\mathbf{U}\mathbf{v}_i) = \mathbf{U}^T\mathbf{v}_j^T \cdot \mathbf{U}\mathbf{v}_i = \mathbf{U}^T\mathbf{U}\mathbf{v}_j^T \cdot \mathbf{v}_i \\ = \mathbf{U}^{-1}\mathbf{U}\mathbf{v}_j^T \cdot \mathbf{v}_i = \mathbf{I}\mathbf{v}_j^T \cdot \mathbf{v}_i = \delta_{ij} \end{aligned}$$

where $\mathbf{I} \in \mathbb{R}^{n \times n}$ is the identity matrix and the third last equality holds due to $\mathbf{U}^T = \mathbf{U}^{-1}$ for orthogonal matrices. So then we see that the dot product is preserved by an orthogonal transformation. And since the dot product is preserved we note that the transformed basis $\mathbf{w}_i$ also is orthogonal and thus, the orthogonal transformation also preserves orthogonality. We now move on to look at the algorithms utilizing this result for solving the eigenvalue problem relevant for this project.

3 Methods

3.1 Discretization

Now we want to numerically approximate equation (7) and recognize the first term, the second derivative, as the same we had in project 1 [4] and recall that this can be approximated trough the three point formula [3]

$$\frac{d^2}{d\rho^2}u(\rho) = \frac{u(\rho + h) - 2u(\rho) + u(\rho - h)}{h^2} + O(h^2) \tag{21}$$

where $O(h^2)$ is the truncation error we make by terminating the Taylor expansion after only a few terms instead of including the whole series. Here h is our step size and defined by the total number of mesh points N and our choice of ρ_0 and ρ_{max} [3]

$$h = \frac{\rho_{max} - \rho_0}{N}. \tag{22}$$

In our case we set ρ_0 equal to zero and preferably we would want ρ_{max} to be ∞ but in our numerical approximation we need to set ρ_{max} to a finite number. We discretize the rest of our ρ values through $\rho_i = \rho_0 + ih$ for $i = 1, 2, \ldots, N-1$. This discretization allows us to rewrite time-independent Schrödinger equation as [5]

$$-\frac{u(\rho_i+h)-2u(\rho_i)+u(\rho_i-h)}{h^2}+\rho_i^2u(\rho_i)=\lambda u(\rho_i) \tag{23}$$

Where ρ_i^2 is the harmonic oscillator potential, V_i. Now we see that this essentially is a problem on a tridiagonal matrix form and define the leading diagonal elements as d_i and the off diagonal elements e_i through [5]

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

and

$$e_i=-\frac{1}{h^2}.$$

Here we realize that all off-diagonal elements are equal constants further simplifying our expression for the time-independent Schrödinger equation to [5]

$$d_iu_i+e_{i-1}u_{i-1}+e_{i+1}u_{i+1}=\lambda u_i, \tag{24}$$

With u_i as unknown. To further visualize that we have reduced our problem to a eigenvalue problem on a tridiagonal matrix form we write [5]

$$\begin{bmatrix} d_0 & e_0 & 0 & 0 & \dots & 0 & 0 \\ e_1 & d_1 & e_1 & 0 & \dots & 0 & 0 \\ 0 & e_2 & d_2 & e_2 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & \dots & \dots & \dots e_{N-1} & d_{N-1} & e_{N-1} \\ 0 & \dots & \dots & \dots & \dots & e_N & d_N \end{bmatrix} \begin{bmatrix} u_0 \\ u_1 \\ \dots \\ \dots \\ \dots \\ u_N \end{bmatrix} = \lambda \begin{bmatrix} u_0 \\ u_1 \\ \dots \\ \dots \\ \dots \\ u_N \end{bmatrix} \tag{25}$$

Here it is important to note that the two end endpoints ρ_0 and ρ_{max} are known through the boundary conditions so we omit these values when we construct our matrix giving us the matrix [5]

$$\begin{bmatrix} \frac{2}{h^2}+V_1 & -\frac{1}{h^2} & 0 & 0 & \dots & 0 & 0 \\ -\frac{1}{h^2} & \frac{2}{h^2}+V_2 & -\frac{1}{h^2} & 0 & \dots & 0 & 0 \\ 0 & -\frac{1}{h^2} & \frac{2}{h^2}+V_3 & -\frac{1}{h^2} & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & \dots & \dots & -\frac{1}{h^2} & \frac{2}{h^2}+V_{N-2} & -\frac{1}{h^2} \\ 0 & \dots & \dots & \dots & \dots & -\frac{1}{h^2} & \frac{2}{h^2}+V_{N-1} \end{bmatrix} \tag{26}$$

where $V_i=\rho_i^2$.

The two-electron wave equation (16) can be discretized in much the same way. The expressions for the diagonal and off-diagonal matrix elements still hold, but the potential is now given by

$$V_i=\omega_r^2\rho_i^2+\frac{1}{\rho_i}$$

3.2 Jacobi rotation algorithm

Now that we have arrived at the final discretized formulation (24) of the differential equation (7) we can look in to methods for solving the equation. As stated before, we are dealing with an eigenvalue problem illustrated by (25) with the matrix given by (26). Thus we want to implement a numerical method for solving for eigenvalues and eigenvectors in order to obtain the wavefunction for the different energies. In particular we will discuss and implement the Jacobi rotation algorithm. This algorithm is designed for dense symmetric matrices (which is the case for us) and is based on performing a series of orthogonal transformations in order to get the matrix in question on diagonal form (as discussed in section 2.3). Orthogonal transformations require an orthogonal matrix $\mathbf{S}$ and, for the Jacobi algorithm, this matrix is a classical rotation matrix generalized for higher dimensions. That is

$$\mathbf{S} = \begin{bmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & & \cos(\theta) & \cdots & -\sin(\theta) & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & & \sin(\theta) & \cdots & \cos(\theta) & \cdots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{bmatrix} \tag{27}$$

Then, a single orthogonal transformation $\mathbf{A}'$ of $\mathbf{A}$ can be written as

$$\mathbf{A}' = \mathbf{S}^T\mathbf{A}\mathbf{S} \tag{28}$$

And since $\mathbf{S}$ is orthogonal, this transformation corresponds to the similarity transformation $\mathbf{A}' = \mathbf{S}^{-1}\mathbf{A}\mathbf{S}$. Also, due to the sparse form of $\mathbf{S}$, the double matrix product in (28) results in only a few elements a_{ij} of $\mathbf{A}$ being affected. Assume that the upper and lower "trigonometric row" in $\mathbf{S}$ are located at index k and l, respectively. Such a rotation "sub matrix" is also square implying that column-wise the indices are k and l as well ($k < l$). Also let $i \in [1, 2, ..., n]$ where $n \times n$ is the size of the matrix. Then when performing the product (28) only the following elements of $\mathbf{A}$ are affected [3]:

$$\begin{aligned} a'_{ii} &= a_{ii} \ , \qquad i \neq k, \ i \neq l \\ a'_{ik} &= a_{ik}\cos(\theta) - a_{il}\sin(\theta) \ , \qquad i \neq k, \ i \neq l \\ a'_{il} &= a_{il}\cos(\theta) + a_{ik}\sin(\theta) \ , \qquad i \neq k, \ i \neq l \\ a'_{kk} &= a_{kk}\cos^2(\theta) - 2_a kl\cos(\theta)\sin(\theta) + a_{ll}\sin^2(\theta) \\ a'_{ll} &= a_{ll}\cos^2(\theta) + 2a_{kl}\cos(\theta)\sin(\theta) + a_{kk}\sin^2(\theta) \\ a'_{kl} &= (a_{kk} - a_{ll})\cos(\theta)\sin(\theta) + a_{kl}\left[\cos^2(\theta) - \sin^2(\theta)\right] \end{aligned} \tag{29}$$

The fact that only these elements are affected can be visualized easier by for example doing this transformation by hand for a 3×3 matrix. We see that generally one such transformation does not form a diagonal matrix as only some elements in the matrix change. Although, the idea performing such a transformation many times while changing k and l for each transformation, seems reasonable. Also, while performing many such transformations, eventually leading to a diagonal matrix, we know, from the linear algebra [6], that the "total" transformation matrix converting $\mathbf{A}$ to diagonal form, contains the eigenvectors of $\mathbf{A}$ as columns. In our case this total

transformation matrix is the product of all the $\mathbf{S}$ matrices we compute for each transformation, i.e. $\mathbf{V} = \mathbf{S_1}\mathbf{S_2}\cdots\mathbf{S_m}$ (if m iterations produces the diagonal matrix). Using this we can update the eigenvectors for each transformation [3]

$$v'_{ik} = v_{ik}\cos(\theta) - v_{il}\sin(\theta) \tag{30}$$
$$v'_{il} = v_{il}\cos(\theta) + v_{ik}\sin(\theta) \tag{31}$$

Again, the reason for these simple expressions is that $\mathbf{S}(k, l, \theta)$ is quite sparse in its form and thus only affects a few elements when updating the eigenvectors. Even though this looks like a decent start, we don't yet know the values of $\cos(\theta)$ and $\sin(\theta)$. Let's look into this issue.

We wish to obtain a diagonal matrix through a series transformations (such as (29) and thus would like to require the non-diagonal elements to be zero. Here this corresponds to $a'_{kl} = 0$ (the lower equation in (29)). We would like to set this to zero for every such transformation and reasonable choices for k and l are discussed below. Requiring $a'_{kl} = 0$ puts a constraint on $\cos(\theta)$ and $\sin(\theta)$. In particular, if $a_{kl} = 0$ as well, we see that $\cos(\theta) = 1$ and $\sin(\theta) = 0$. If $a_{kl} \neq 0$ we can divide by $\cos^2(\theta)$ and recognize $\tan(\theta) = \sin(\theta)/\cos(\theta)$ to get

$$\frac{a_{kk} - a_{ll}}{a_{kl}}\tan(\theta) - \tan^2(\theta) + 1 = 0 \qquad \Rightarrow \qquad \tan^2(\theta) + 2\tau\tan(\theta) - 1 = 0 \tag{32}$$

with $\tau = (a_{ll} - a_{kk})/2a_{kl}$. This is a quadratic equation for $\tan(\theta)$ and has solutions

$$\tan(\theta) = -\tau \pm \sqrt{1 + \tau^2} \tag{33}$$

and further yields

$$\cos(\theta) = \frac{1}{\sqrt{1 + \tan^2(\theta)}}\,, \qquad \sin(\theta) = \cos(\theta)\tan(\theta) \tag{34}$$

through trigonometric identities. Having obtained the values for $\cos(\theta)$ and $\sin(\theta)$ allows us to calculate the elements of the transformed matrix through (29) and update the eigenvectors through (30) and (31). The only remaining question now is how to choose k and l for each transformation. Letting k and l be the indices of the largest non-diagonal element in $\mathbf{A}$, i.e.

$$a_{kl} = \max_{i \neq j} a_{ij} \tag{35}$$

seems reasonable because then, for every transformation, we force the largest non-diagonal element of $\mathbf{A}$ to be zero through (32). This choice of k and l gives the largest shift towards a diagonal matrix for every transformation.

The above was an attempt at describing the functionality of the algorithm, but now it's time to look at the actual algorithm that we have implemented. See algorithm 1. Our implementation follows the basic ideas presented in algorithm 1, but the structure of the code is not identical to that of the algorithm. This is due to a lot of other code technical issues that need to be taken care of.

Algorithm 1 Jacobi algorithm

procedure JACOBI_EIGEN(A, N)
  while max_non_diag > tolerance **do**
    // Find max non-diagonal element

    for $i = 0$ to $i = N - 2$ **do**
      for $j = i + 1$ to $i = N - 1$ **do**
        if abs(a_{ij}) > max_non_diag **then**
          max_non_diag = abs(a_{ij})
          $k = i$
          $l = j$

    // Find cosine, sine values for transformation matrix
    if $a_{kl} \neq 0$ **then**
      $\tau = (a_{ll} - a_{kk})/2a_{kl}$
      $\tan(\theta) = -\tau - \sqrt{1 + \tau^2}$ *// Choose smallest*
      $\cos(\theta) = 1/\sqrt{1 + \tan(\theta)^2}$ *// Trig identity for cosine*
      $\sin(\theta) = \tan(\theta) * \cos(\theta)$ *// Trig identity for sine*
    else
      $\cos(\theta) = 1$
      $\sin(\theta) = 0$

    // Loop over diagonal
    for $i = 0$ to $i = N - 1$ **do**

      // Execute transformation $A' = S^T AS$
      if $i \neq k$ **and** $i \neq l$ **then**
        $a'_{ii} = a_{ii}$
        $a'_{ik} = a_{ik}\cos(\theta) - a_{il}\sin(\theta)$
        $a'_{il} = a_{il}\cos(\theta) + a_{ik}\sin(\theta)$

      $v'_{ik} = v_{ik}\cos(\theta) - v_{il}\cos(\theta)$ *// Update eigenvector component*
      $v'_{il} = v_{il}\cos(\theta) + v_{ik}\sin(\theta)$ *// Update eigenvector component*

    $a'_{kk} = a_{kk}\cos^2(\theta) - 2a_{kl}\cos(\theta)\sin(\theta) + a_{ll}\sin^2(\theta)$
    $a'_{ll} = a_{ll}\cos^2(\theta) + 2a_{kl}\cos(\theta)\sin(\theta) + a_{kk}\sin^2(\theta)$
    $a_{kl} = 0$ *// Force max-non diag to be zero*
    $a_{lk} = 0$ *// Force max-non diag to be zero*
  return (A, V) *// Return eigenvalues and eigenvector*

3.3 Testing

When writing code and developing software in general one often writes some functionality and then later update or make enhancments to that piece of code. Then it's easy to make changes that suddenly results in that particular code not working as it should anymore, but having the effect of the entire program producing unwanted behaviour. As a programmer, it's then often not trivial to pinpoint where in the code the source of error is. This is one area where unit tests become useful. Constructing tests for various parts of the program will help the programmer quickly localize the faulty logic and thus save time. In addition, unit tests acts as a great way to make sure some code snippet being worked on, functions as expected before movin on to write other parts of the program. Also they act as additional documentation for the code by exemplifying functionality. For computer science in general these unit tests could be testing arbitrary functionality. However in our contex, it's particularly useful for testing mathematical properties, which is the focus below. For this project we have implemented four unit tests.

3.3.1 Max non-diagonal element test

We have constructed a unit test to ensure that our function that finds the element with the highest absolute value of a matrix actually returns the element with the highest absolute value. We have done this by constructing a small matrix with a known maximum element and perform a function call to our function that find the maximum element. We then preform an if test to see if the maximum value actually was returned by our function. If the function finds the right element a message is prompt in the terminal window telling us that the maximum value test is passed.

3.3.2 Eigenvalue test

We also constructed a unit test to check if we have implemented the Jaocbi algorithm correctly in our program. We defined a 4x4 matrix and used Matlab to find the eigenvalues of the matrix. We then preform a function call to our Jacobi solver which returns the eigenvalues and eigenvectors. An if test checks if the correct eigenvalues are returned within a reasonable level of accuracy, 10^{-5}, then a message is prompt in the terminal window telling the user whether the eigenvalue test is passed or not.

3.3.3 Orthogonality test

Another thing we wanted to test is if our implementation of algorithm 1 preserves orthogonality as discussed in section 2.3. In light of this we made a unit test using a known 5×5 matrix and starting with an identity matrix $\mathbf{I}$ as a basis. Then calling on `jacobi_eigen()` we get back the resulting eigenvectors as columns in a matrix $\mathbf{V}$. To check for orthogonality of the columns of $\mathbf{V}$ we loop through every possible combination of columns of $\mathbf{V}$ and compute the inner product of each combination. This can be done as illustrated in the code snippet below:

```
    for (int c_1 = 0; c_1 < N; c_1++)       // Loop through cols in V.
        for (int c_2 = 0; c_2 < N; c_2++)  // Loop through cols in V per col
            inner_prod = 0.0;

            for (int k = 0; k < N; k++)
                inner_prod += V(k,c_1)*V(k,c_2);     // Compute inner product
```

For the orthgogonality to be preserved we would need

$$\mathbf{v}_i \cdot \mathbf{v}_j = \begin{cases} 1 \ , \ i = j \\ 0 \ , \ i \neq j \end{cases} \tag{36}$$

where i and j correpond respectively to column i (`c_1`) and j ((`c_2`) in $\mathbf{V}$. And indeed while running the test we observe that our implementation preserves orthogonality.

3.3.4 Transformation matrix test

Finally we also implemented a unit test for checking that our program finds the correct transformation mastrix, i.e. computes the correct $\cos(\theta)$ and $\sin(\theta)$ values. As with the above tests we used a known small matrix where we hand computed the $\cos(\theta)$ and $\sin(\theta)$ values and then compared the values produced by the code.

4 Results

We want to investigate how the resulting three lowest lying eigenvalues behave as a function of the matrix size, n. For the non interacting case we have analytical answers for the eigenvalues and know that the three lowest eigenvalues, namely λ_1, λ_2 and λ_3, should be equal to 3, 7 and 11 respectively [5]. Our results are enclosed in table 4.1 where we have used $\rho_{max} = 4.0$ and that our tolerance is $\epsilon = 10.0^{-9}$ for when the off diagonal elements are essentially zero. We have found that our solution is sensitive to choices of ρ_{max} and through trial and error found that $\rho_{max} = 4.0$ is a good choice for us.

From table 4.1 we see that the eigenvalues generally get closer to the exact eigenvalues as we increase the matrix size. When we discretized the Schrödinger equation, we chose to neglect terms of order h^2. As h is inversely proportional to the matrix size n through $h = \frac{\rho_{max} - \rho_0}{n+1}$ we expect the error to decrease as we increase the matrix size. The lowest eigenvalue converges beautifully as the matrix size increases. The second and third eigenvalues seem to converge to a value a little higher than their excact value. This is probably related to our choice of ρ_{max}, but since the values are reasonably close to the exact values we chose not to experiment further with values for ρ_{max}.

With two electron confined in a harmonic oscillator potential repulsive Coulomb forces acting between the two bodies are included. We have used four different oscillator frequencies, ω_r, namely 0.01, 0.5, 1.0 and 5.0. The resulting wave vectors are shown in figure 4.2 for the three lowest lying energy states for different oscillator frequencies, ω_r.

We have found that our solution is sensitive to the our choice of ρ_{max} for both the interacting and non interacting case. In both instances the wave function for the different energy levels should all converge towards zero in both tails. So if we chose a big ρ_{max}, we will include many data points of little interest where the wave function is ~ 0. If we look at equation (22) we see how our choice of ρ_{max} dictates together with the matrix size, n, our step size h. So a choice of a too large ρ_{max} will lead us to loose valuable information in the regions where the wavefunctions is not equal to zero. On the other hand, choosing ρ_{max} too small will make us lose part of the wavefunction. Figure 4.2 clearly shows why $\rho_{max} = 4.0$ was such a good choice for the non-interacting case. All three wavefunctions are essentially zero above this value, and we thus include all the interesting

Matrix size, n	λ_1	λ_2	λ_3
5	2.8530	6.2118	9.0339
10	2.9581	6.7896	10.5214
20	2.9886	6.9461	10.9307
50	2.9981	6.9937	11.0540
100	2.9995	7.0009	11.0725
150	2.9998	7.0023	11.0760
200	2.9999	7.0028	11.0772
300	3.0000	7.0031	11.0781
400	3.0000	7.0032	11.0784
500	3.0000	7.0033	11.0786

Table 4.1: A table showing the convergence rate for the non interacting case of the three lowest eigenvalues, λ_i, as function of the matrix size, n.

parts of the wavefunctions in our computation. And since we do not include uninteresting regions where the wavefunctions are zero, we ensure that our step size h is as small as possible while still including all regions of interest.

From figure 4.2 we see that for smaller values of ω_r a larger value of ρ_{max} is needed to include all areas of interest, while when ω_r is increasing the electrons are confined to a smaller area and a smaller ρ_{max} is needed. It is physically meaningful that when the oscillating potential is increasing, the repulsive energy acting between the two electrons play a less significant role and the electrons are confined in a smaller area, while when ω_r is increasing the Coulomb interaction forces the electrons to favor positions far away from each other.

If we compare the plot for the one electron 4.1 with the two electron 4.2 situation we see that when repulsive forces are included, the wavefunctions gets stretched over a larger interval of ρ for ω_r larger then 1.0.

It is interesting to note that our wavefunctions are orthonormal meaning that they are both orthogonal and normalized. We start our Jacobi algorithm with a identity matrix as the matrix in which the eigenvectors are to be contained. This matrix is orthonormal and the norm is conserved through our similarity transformations because the similarity transformation matrix is orthogonal [6].

Further we want to investigate how many similarity transformations are needed before all the off diagonal matrix elements are essentially equal to zero, and the matrix is diagonal. The results are enclosed in table 4.2. We see that our algorithm converges towards $\sim 1.7n^2$ similarity transformations needed already for $n = 50$. From our lecture notes [3] we know that the number of similarity transforms needed to diagonalize a symmetric matrix is estimated to be between $3n^2$ and $5n^2$. Our program does not completely diagonalize the matrix, but terminates when the absolute value of the largest off-diagonal matrix elements is smaller than a certain tolerance, in our case 10.0^{-9}. A plausible explanation for our algorithm utilizing fewer transformations that listed in our lecture

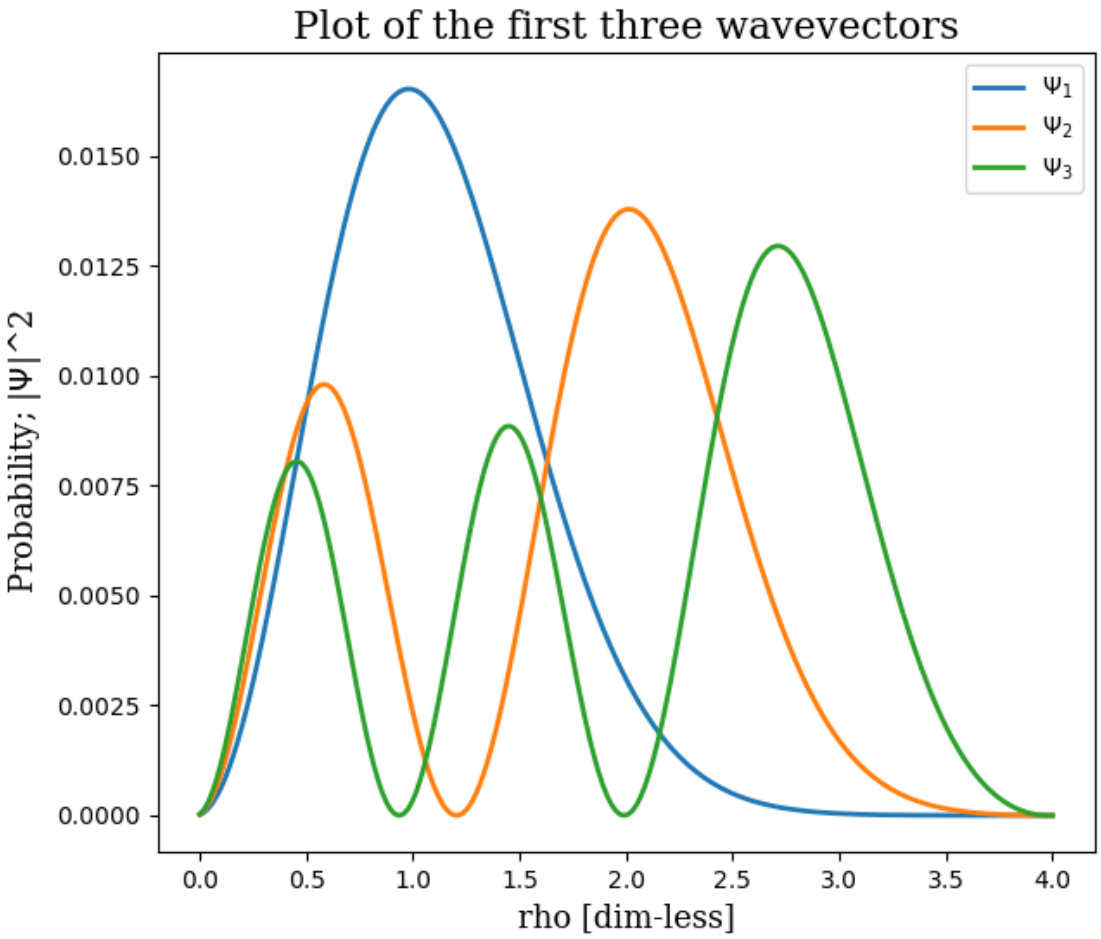

Figure 4.1: Plot of the wavefunctions squared, ψ^2, corresponding to the three lowest eigenvalues , $\lambda_1 = 3, \lambda_2 = 7$ and $\lambda_3 = 11$ of an electron in a harmonic potential.

notes [3] is our choice of ϵ. If we run the algorithm for $n = 200$ and use $\epsilon = 10^{-20}$, 86 087 similarity transformations are needed, resulting in a behavior of $\sim 2.2n^2$. Where we to choose an infinitively small tolerance ϵ we would probably need between $3n^2$ and $5n^2$ transformations. Table 4.1 does however show that our eigenvalues are reasonably good for $\epsilon = 10.0^{-9}$ transformations, and since this tolerance nearly halves the expected number of needed transformations, we see that there is a lot of time to be saved by choosing our tolerance wisely.

How efficient is algorithm 1? As it turns out, not very efficient. To answer this question in more detail, lets consider a short analysis of the cost of the algorithm. When searching for the largest non-diagonal element, we utilised the fact that we have a symmetric matrix $(n \times n)$. In particular this means we only need to search through the upper triangular part of the matrix, giving $0.5n^2 - n$ steps (subtracting n due to main diagonal not being part of search). Computing $\cos(\theta)$ and $\sin(\theta)$ only requires a couple of steps so it doesn't contribute significantly. Then performing the transformation requires $4n$ steps (neglecting what happens outside the loop). Now, all of the above happens each iteration, and, as we see from table 4.2, we need $\sim 1.75n^2$ iterations before

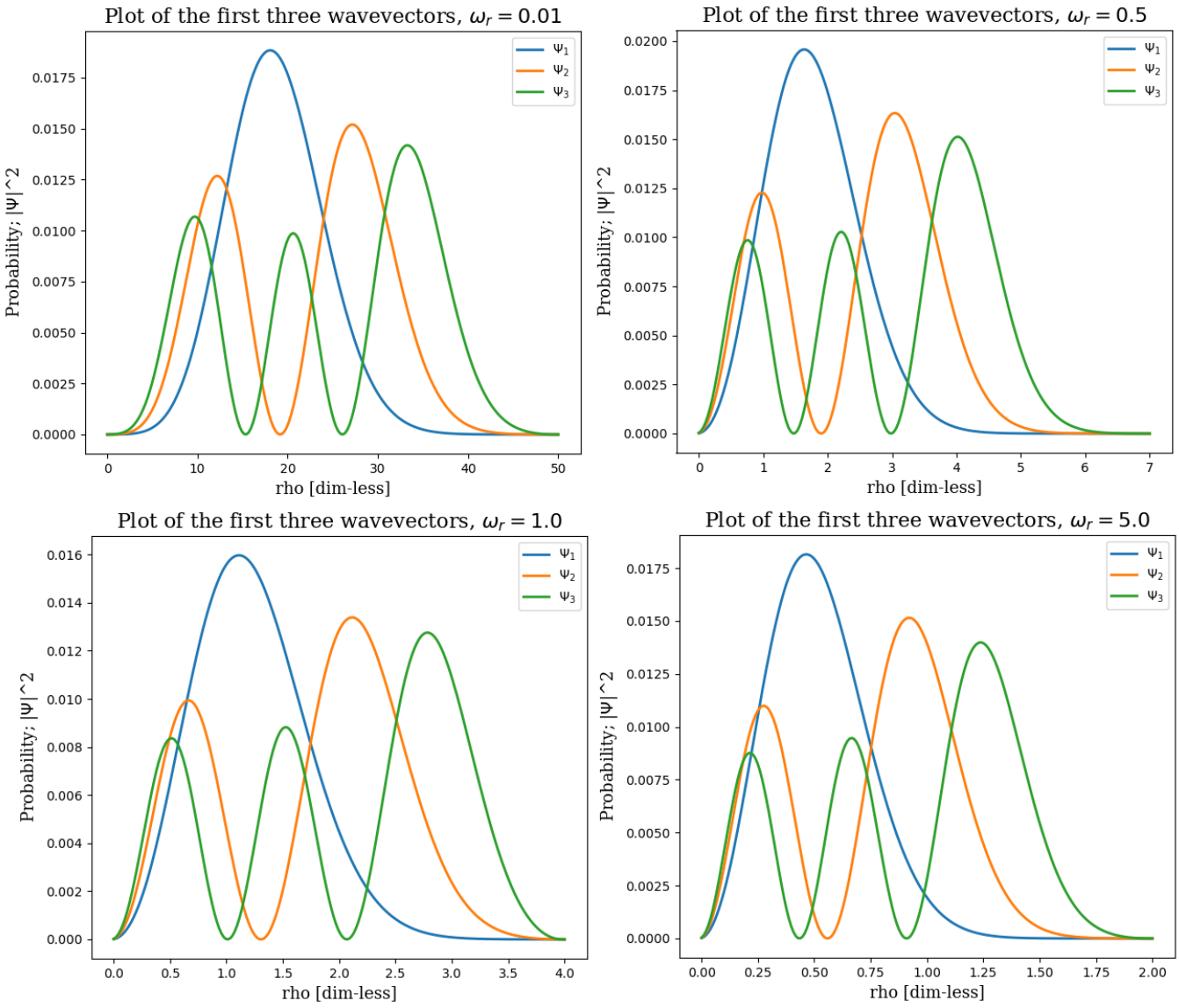

Figure 4.2: Plots of the wavefunctions squared, ψ^2, corresponding to the three lowest eigenvalues of two interacting electrons in a harmonic potential of frequency ω_r.

satisfactory convergence. Combining these results give a total cost of

$$\text{COST} = 1.75n^2(0.5n^2 - n + 4n) = 1.75n^2(0.5n^2 + 3n) = 0.875n^4 + 5.25n^3 = \mathcal{O}(n^4) \qquad (37)$$

where the last equality illustrates the asymptotic cost of the algorithm, that is, when $n \to \infty$. So we note that the algorithm is not very efficient and this we got to experience personally when working on the project. Attempting to run the algorithm for large matrices had the potential to make us impatient at times. It's worth noting that the above used factor of 1.75 is dependent of what we choose as the tolerance for the max value (in algorithm 1) of the non-diagonal elements. In this case we used a tolerance of 10^{-9}.

As mentioned earlier the Jacobi rotational algorithm is not the most efficient way of finding eigenvalues and eigenvectors, especially for large matrices. To clarify this point we have compared our code to Armadillos [7] eigenvalue solver eig_sym(). Here we have found that our eigenvalues and

Matrix size, n	Number of iterations	Iterations/n^2
5	28	1.12
10	148	1.48
20	629	1.57
50	4 234	1.69
100	17 279	1.73
150	39 013	1.73
200	69 626	1.74
300	157 213	1.75
400	280 178	1.75
500	439 315	1.76

Table 4.2: A table showing the number of transformations needed to diagonalize a matrix of size n. We have terminated the transformations when the square of the off-diagonal matrix elements are smaller than 10.0^{-9}

eigenvector are in good correlation with those found by Armadillo, but the time used differs significantly, see table 4.3. As Armadillo uses far less time, it is obvious that there are more efficient ways of solving an eigenvalue problem numerically.

Matrix size, n	Armadillo time [s]	Jacobi time [s]
10	1.13×10^{-4}	2.5×10^{-4}
100	3.03×10^{-3}	1.39
200	1.48×10^{-2}	21.99
300	4.13×10^{-2}	113.96

Table 4.3: A table showing the time used by our Jacobi rotational algorithm and Armadillos function eig_sym() as function of the matrix size, n, for the non interacting case.

5 Concluding remarks

For many physics students, the Schrödinger equation is part of their first introduction to quantum mechanics. Throughout a quantum mechanics course, the student will most likely solve the Schrödinger equation analytically quite a few times. The student will, however, soon realize that finding an exact solution for the energies and wavefunctions is exclusive to a couple of simple cases. As soon as more complicated potentials are introduced, i.e. two interacting electrons in a harmonic potential, the Schrödinger equation is unsolvable. Fortunately the time-independent Schrödinger equation can be reduced to an eigenvalue problem, which can be solved numerically.

In this project we wanted to familiarize ourselves with one of these methods, namely the Jacobi method. As we solved the Schrödinger equation for a potential where the exact solution for the energies is known, we were able to compare our results to the exact ones. Table 4.1 clearly shows how the solutions obtained by using the Jacobi method quickly converges towards the exact solutions, even for matrices as small as 100x100. This is fortunate, as the method is very slow for larger matrices (see table 4.3) as uses more than a minute for matrices of size 300x300. If efficiency is not important or if we want to find the eigenvalues and eigenvectors of a small matrix, the method is an excellent choice.

If one were to work further on this project, it might be worthwhile to look into other more time efficient methods. In this case the matrix is tridiagonal, so a lot of time could be spared by using i.e. the bisection method to compute the eigenvalues.

Personal notes:
All in all we were positively surprised by the precision of the Jacobi method. When solving equations numerically one often have to choose between numerical precision and time efficiency, and if time is not an issue this is a perfect solver of eigenvalue problems. In this project we were also introduced to working with unit tests (see section 3.3), and we felt it really helped during development of the code. For instance, the unit test that checks if the max non-diagional element was found, saved one of us a lot of time by making us realize that our function did not work when the largest element (in absolute value) was negative.

References

[1] Physics. `https://en.wikipedia.org/wiki/Physics`. Accessed: 29.09.2017.

[2] David Griffiths. *Introduction to Quantum Mechanics*. Pearson, 2014.

[3] Morten Hjorth-Jensen. Computational physics. University Lecture Notes, 2015.

[4] Morten Hjorth-Jensen. Project 1. University Project Decription, 2017.

[5] Morten Hjorth-Jensen. Project 2. University Project Decription, 2017.

[6] David Lay. *Linear Algebra and Its Applications*. Pearson, 2014.

[7] Conrad Sanderson and Ryan Curtin. *Armadillo: a template-based C++ library for linear algebra.* Journal of Open Source Software, http://dx.doi.org/10.21105/joss.00026, 2016.