Numerical Eigenvalue Solver:
Jacobi's Algorithm (Project 2)

Numpy Nat Hawkins, Venv Victor Ramirez, Matplotlib Mike Roosa, Pandas Pranjal "Danger" Tiwari

March 2, 2017

Abstract

The goal of this project is to explore a model of quantum dots. We will be investigating the behavior of two electron in a 3-D simple harmonic potential while comparing the models with and without the particles interacting. To do this we will be solving the Schrodinger equation using the Jacobi method. What we found is that with our Jacobi eigensolver, one of the many issues surrounding it is that we do not know the maximum number of iterations needing to be performed on the matrix in question in order to get the eigenvalues. This lead to some issues in our attempts at writing the program for the eigensolver. We were able to calculate the eigenvalues for a square symmetric matrix that agree with the eigenvalues of standard python library solvers (i.e. numpy.linalg.eig). The eigensolver was also used to compute eigenvalues for a specific value of frequency, ω, which we then compared to the analytic results. Our numerical values were within 0.5% of the analytic value.

1 Introduction

1.1 Mathematical Motivation

The aim of this project is to solve Schroedinger's equation for two electrons in a three-dimensional harmonic oscillator well with and without a repulsive Coulomb interaction. We aimed to solve this equation by reformulating it in a discretized form as an eigenvalue equation to be solved with Jacobi's method.

Electrons confined in small areas in semiconductors, so-called quantum dots, form a hot research area in modern solid-state physics, with applications spanning from such diverse fields as quantum nano-medicine to the contruction of quantum gates.

Here we will assume that these electrons move in a three-dimensional harmonic oscillator potential (they are confined by for example quadrupole fields) and repel each other via the static Coulomb interaction. We assume spherical symmetry.

We are first interested in the solution of the radial part of Schroedinger's equation for one electron. This equation reads

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

In our case $V(r)$ is the harmonic oscillator potential $(1/2)kr^2$ with $k=m\omega^2$ and E is the energy of the harmonic oscillator in three dimensions. The oscillator frequency is ω and the energies are

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

with $n=0,1,2,\ldots$ and $l=0,1,2,\ldots$.

Since we have made a transformation to spherical coordinates it means that $r\in[0,\infty)$. The quantum number l is the orbital momentum of the electron. Then we substitute $R(r)=(1/r)u(r)$ and obtain

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

The boundary conditions are $u(0) = 0$ and $u(\infty) = 0$.

We introduce a dimensionless variable $\rho = (1/\alpha)r$ where α is a constant with dimension length and get

$$-\frac{\hbar^2}{2m\alpha^2}\frac{d^2}{d\rho^2}u(\rho) + \left(V(\rho) + \frac{l(l+1)}{\rho^2}\frac{\hbar^2}{2m\alpha^2}\right)u(\rho) = Eu(\rho).$$

We will set in this project $l = 0$. Inserting $V(\rho) = (1/2)k\alpha^2\rho^2$ we end up with

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

We multiply thereafter with $2m\alpha^2/\hbar^2$ on both sides and obtain

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

The constant α can now be fixed so that

$$\frac{mk}{\hbar^2}\alpha^4 = 1,$$

or

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

Defining

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

we can rewrite Schroedinger's equation as

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

This is the first equation to solve numerically. In three dimensions the eigenvalues for $l = 0$ are $\lambda_0 = 3, \lambda_1 = 7, \lambda_2 = 11, \ldots$.

We use the by now standard expression for the second derivative of a function u

$$u'' = \frac{u(\rho + h) - 2u(\rho) + u(\rho - h)}{h^2} + O(h^2), \tag{1}$$

where h is our step. Next we define minimum and maximum values for the variable ρ, $\rho_{\min} = 0$ and $\rho_{\max}$, respectively. We needed to check our results for the energies against different values $\rho_{\max}$, since we cannot set $\rho_{\max} = \infty$. For the sake of our initial numerical analysis, we set $\rho_{\max}$=10. In comparing to analytc results at ω=0.25, we set $\rho_{\max}$=40, corresponding to 10 divided by our chosen value of ω. This was as a result of the work done by Tout and his choice of $\rho_{\max}$, also, we had to choose appropriate values for the interacting case based on our substituions made in order to simplify Schrodinger's Equation for said case.

With a given number of mesh points, N, we define the step length h as, with $\rho_{\min} = \rho_0$ and $\rho_{\max} = \rho_N$,

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

The value of ρ at a point i is then

$$\rho_i = \rho_0 + ih \qquad i = 1, 2, \ldots, N.$$

We can rewrite the Schroedinger equation for a value ρ_i as

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

or in a more compact way

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

where $V_i = \rho_i^2$ is the harmonic oscillator potential.

We define first the diagonal matrix element

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

and the non-diagonal matrix element

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

In this case the non-diagonal matrix elements are given by a mere constant. *All non-diagonal matrix elements are equal.* With these definitions the Schroedinger equation takes the following form

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

where u_i is unknown. We can write the latter equation as a matrix eigenvalue problem

$$\begin{bmatrix} d_0 & e_0 & 0 & 0 & \dots & 0 & 0 \\ e_1 & d_1 & e_1 & 0 & \dots & 0 & 0 \\ 0 & e_2 & d_2 & e_2 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & \dots & \dots & \dots e_{N-1} & d_{N-1} & e_{N-1} \\ 0 & \dots & \dots & \dots & \dots & e_N & d_N \end{bmatrix} \begin{bmatrix} u_0 \\ u_1 \\ \dots \\ \dots \\ \dots \\ u_N \end{bmatrix} = \lambda \begin{bmatrix} u_0 \\ u_1 \\ \dots \\ \dots \\ \dots \\ u_N \end{bmatrix}. \tag{2}$$

Since the values of u at the two endpoints are known via the boundary conditions, we can skip the rows and columns that involve these values. Inserting the values for d_i and e_i we have the a matrix form we can now use in Jacobi's Algorithm to solve for the energies. [2] The Hamiltonians that we will be concerned with will be in the form of a tridiagonal matrix and tridiagonal matrices are simple to get eigenvalues from, but if the matrix were 100x100, it would be too much to compute by hand. Therefore, the discretized method we will implement will allow for ease of computation via numerical methods.

We also now studied two electrons in a harmonic oscillator well which also interact via a repulsive Coulomb interaction. We started with the single-electron equation written as

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

where $E^{(1)}$ stands for the energy with one electron only. For two electrons with no repulsive Coulomb interaction, we used the following equation

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

Note that we deal with a two-electron wave function $u(r_1, r_2)$ and two-electron energy $E^{(2)}$.

With no interaction this can be written out as the product of two single-electron wave functions, that is we have a solution on closed form. We 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

$$E^{(2)} = E_r + E_R.$$

We add then the repulsive Coulomb interaction between two electrons, namely a term

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

with $\beta e^2 = 1.44$ eVnm. Adding this term, the r-dependent Schroedinger equation becomes

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

This equation is similar to the one we had previously with the noninteracting case and we introduce again a dimensionless variable $\rho = r/\alpha$. Repeating the same steps as we did prior, we arrive at

$$-\frac{d^2}{d\rho^2}\psi(\rho) + \frac{1}{4}\frac{mk}{\hbar^2}\alpha^4\rho^2\psi(\rho) + \frac{m\alpha\beta e^2}{\rho\hbar^2}\psi(\rho) = \frac{m\alpha^2}{\hbar^2}E_r\psi(\rho).$$

We want to manipulate this equation further to make it as similar to the previous equation we solved as possible. We define a new 'frequency'

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

and fix the constant α by requiring

$$\frac{m\alpha\beta e^2}{\hbar^2} = 1$$

or

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

Defining

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

we can rewrite Schroedinger's equation as

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

We treat ω_r as a parameter which reflects the strength of the oscillator potential.

Here we will study the cases $\omega_r = 0.25$ for the sake of verifying our results against the analytic results, $\omega_r = 0.01$, $\omega_r = 0.5$, $\omega_r = 1$, and $\omega_r = 5$ for the ground state only, that is the lowest-lying state.

With no repulsive Coulomb interaction we expect to get a result which corresponds to the relative energy of a non-interacting system.

We are only interested in the ground state with $l = 0$. We omit the center-of-mass energy. We can reuse the same code, but we need to change the potential from ρ^2 to $\omega_r^2\rho^2 + 1/\rho$.

For specific oscillator frequencies, the above equation has answers in an analytical form, see the article by Taut [1].

These are the two mathematically distinct cases that we are interested in exploring in our program. Summary of results will be included in our analysis.

1.2 Methods

Given this background, we sought to develop a model for the non-interacting and interacting cases of the Schrodinger Equation for two electrons and write an algorithm that would solve for the eigenvalues of our tridiagonal matrix. In the next section, we will discuss how we set up our tridiagonal matrix and how the

Jacobi Algorithm solves for the eigenvalues. We ensured that these eigenvalues are accurate by conducting unit tests, such as confirming the orthogonality of our eigenvalues. We also studied the deviation of our calculated results based on varying our parameters as well as the error between our calculated eigenvalues and the accepted values of these eigenvalues according to the literature.[1]

We elected to do this numerical analysis in Python. While we are beginning to become more familiarized with C++, we sought to continue to play to our strengths and produce a Python code for our results. C++ would likely be much more computationally effecient and would allow us to solve larger and larger matrix systems, but in the interest of advancing our knowledge of Python as well as producing quality work, Python was our choice for programming language.

For a good discussion on the Linear Algebra methods explored in this project, a good textbook for reference is G. Strang. [3] For a more thorough discussion on the Jacobi Method and other numerical methods that we could have implored for this project, as we will discuss Jacobi's algorithms aren't the most effecient or wisest choice for this type of problem, see Kelley's book on linear and nonlinear methods. [4]

1.3 Development

In order to achieve the models for non-interacting and interacting electrons, we needed to develop several functions. We found it important to outline the main points about the functions we needed to create prior to the discussion of our work and results.

The first function that we needed to define was a function to set up our matrix. This outlines a basic $n \times n$ matrix, which we could outline, that takes in arguments for the potential and outputs a tridiagonal symmetric matrix that we will discuss in our Setup section to come. Then, we set up a function that took in a a tridiagonal matrix and applies the Jacobi algorithm one time to provide us with a matrix that is gradually more diagonalized. The calculated eigenvalues converge with each iteration up to a specified decimal point. The final main function we needed to develop was one that would iterate the matrix through the Jacobi algorithm function several times until the largest off diagonal matrix elements fell below our defined value of tolerance ($10^{-6} - 10^{-7}$). The other functions we chose to define performed operations for unit testing, checking for the preservation of orthogonality, and for removing small matrix elements that, after several iterations, became ≈ 0. These will be outlined and commented in our final code and can be seen in the ipython notebook attached at the end of this report. The following sections will discuss the implementation and setup of our algorithms and functions.

2 Solution

2.1 Setup

We know that the Hamiltonian is a tridiagonal matrix, where the diagonals are $2/\hbar^2$ +V_N and the elements on either side of the diagonals are -$1/\hbar^2$ for an NxN matrix, is given by multiplying out the matrix from the mathematical motivation (2), is given as:

$$H = \begin{bmatrix} \frac{2}{\hbar^2} + V_1 & -\frac{1}{\hbar^2} & 0 & 0 & \dots & 0 \\ -\frac{1}{\hbar^2} & \frac{2}{\hbar^2} + V_2 & -\frac{1}{\hbar^2} & 0 & \dots & 0 \\ 0 & -\frac{1}{\hbar^2} & \frac{2}{\hbar^2} + V_3 & -\frac{1}{\hbar^2} & \dots & 0 \\ 0 & 0 & -\frac{1}{\hbar^2} & \frac{2}{\hbar^2} + V_4 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & -\frac{1}{\hbar^2} \\ 0 & 0 & 0 & 0 & -\frac{1}{\hbar^2} & \frac{2}{\hbar^2} + V_{N-1} \end{bmatrix}$$

This matrix is what we need to get the eigenvalues of. There is a function in python which tells us the eigenvalues of a matrix and we use this at the beginning of our code to see what eigenvalues to expect, so that once we create out Jacobi solver, we know whether the values returned are correct or not. The built in Python library we implored is Numpy.Linalg, which has functions such as numpy.linalg.eig, which will give the eigenvalues and eigenvectors of the matrix input, and numpy.linalg.eigvals, which gives a list of numerical eigenvalues. This was used to check against our computed eigenvalues. The "eig" function was used to extract the eigenvectors of our matrix in question. This became important when looking at whether or not the orhtogonality was preserved, to be discussed later.

We began by intializing the matrix for the electrons. Initially, we set the matrix up for the basic harmonic oscillator potential using a potential terms of $\frac{1}{\rho^2}$. This was added to the diagonal elements. We included the term for the step size into our intialization as well. One challenge we were faced with in this intialization was the realization of our boundary conditions. We fixed the V[0] and V[n] term to be zero, in accordance to our wavefunction going to zero at both ends in the limit of Schrodinger's equation and also in accordance with our numerical methods. The function is fairly simple and returns a matrix.

We then decided to develop the Jacobi algorithm one set of operations at a time on a 2×2 matrix we defined as

$$\begin{bmatrix} 2 & -4 \\ -4 & 1 \end{bmatrix}$$

This was the first of several unit tests we performed in our analysis, but the one that allowed us to develop the algorithms necessary piece by piece. We ran the numpy linear algebra functions to extract expected eigenvalues and eigenvectors.

The first functionality that needed to be integrated into our program was the function that extracted the largest non-diagonal matrix element and its associated index. The index is used in calculating the various terms of the Jacobi algorithm. To find the largest element, we set some arbitrary lower bound, 0. While looping over the matrix elements in the off-diagonal indeces, if the found value exceeded 0, it replaced the value and the process continued until we found the largest element. The absolute value of this number was returned. We verified for our test matrix that the largest off-diagonal element was indeed found.

Then, we set to define our trig functions in accordance with our numerical methods (to be discussed explicitly in the following sections). These were calculated in relation to a defined value, τ. Once the trig values were calculated, we could then write the Jacobi algorithm for a single iteration. This would produce a diagonal matrix for the 2×2 case, but in higher order matrices we would need additional iterations to get the fully diagonalized matrix. This testing gave us a matrix with the proper eigenvalues, and unit testing on the eigenvectors after the transformation showed the preservation of the orthogonality.

Once each component was functional, we combined the multiple functions into one that could be run over the matrix encompassing all of the steps we had verified. This was then redefined into an iterative function and a unit test was performed on two 3×3 matrices.

The gist of our setup comes from setting up programs and functions to carry out the algorithms as well as defining the matrices for the potentials we decide to subject the electrons to.

2.2 Jacobi Algorithm

We chose the Jacobi method to solve for our eigenvalues. The Jacobi method is an iterative method that transforms a symmetric tridiagonal matrix by rotating the matrix until it converges to a solution. The algorithm is as follows:

1. Search for the largest matrix element $|a_{pq}|$, where indices p and q denote the row and column of the max non-diagonal element of the matrix.

2. Given p and q, we will perform the Jacobi rotation. We define the quantities s, c, t as $\sin\theta$, $\cos\theta$, and

$\tan\theta$ respectively. To obtain s and c, we use the following relationships:

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

We also define a quantity τ as:

$$\tau = \frac{a_{qq} - a_{pp}}{a_{pq}} \text{ where } t^2 + 2\tau t - 1 = 0.$$

Truncation errors occur when τ is very large which skew the value of t and thus s and c. To avoid this, we redefine t as:

$$t = \begin{Bmatrix} \frac{1}{\tau+\sqrt{1+\tau^2}}, & \text{for } \tau > 0 \\ \frac{1}{-\tau+\sqrt{1+\tau^2}}, & \text{for } \tau < 0 \end{Bmatrix}$$

3. We use t and τ to compute c and s. With values for c and s and indices p and q, we calculate the elements of our new matrix as follows:

$$\begin{aligned} b_{ip} &= a_{ip}c - a_{iq}s \quad i \neq p, i \neq q \\ b_{iq} &= a_{iq}c + a_{ip}s \quad i \neq p, i \neq q \\ b_{pp} &= a_{pp}c^2 - 2a_{pq}cs + a_{qq}s^2 \\ b_{qq} &= a_{pp}s^2 + 2a_{pq}cs + a_{qq}c^2 \\ b_{pq} &= 0 \\ b_{pi} &= b_{ip} \\ b_{qi} &= b_{iq} \end{aligned}$$

   The first two expressions transform the tridiagonal elements to converge to 0. The following two expressions are further corrections to the max elements remaining in the matrix. The last expression "forces" the matrix to stay symmetric as the Jacobi method only works for symmetric matrices.

4. We then repeat 1. and 2. until the largest non-diagonal element a_{pq} is less than some desired accuracy ϵ. We can then read off the eigenvalues as the diagonal elements of the transformed matrix A.

This method is a straightforward, albeit inefficient way to solve for the eigenvalues. We chose this method for its simplicity as it allowed us to easier understand the nuances of eigenvalue solvers. For future work that requires solving for eigenvalues, it's best to stick to faster algorithms such as the Householder algorithm or use libraries such as numpy's linalg module for Python or armadillo for C++.

2.3 Preservation of Orthogonality and Unit Testing

A unitary transformation preserves the orthogonality of the obtained eigenvectors. To see this consider first a basis of vectors $\mathbf{v}_i$,

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

We assume that the basis is orthogonal, that is

$$\mathbf{v}_j^T \mathbf{v}_i = \delta_{ij}.$$

We set out to show that an orthogonal or unitary transformation

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

preserves the dot product and orthogonality.
In the code, which will be provided at the end of the report, we implemented multiple unit tests, one of which having to do with the preservation of orthogonality. Out[17] shows the unit test that we performed for a 3×3 matrix we created. This is found on page 8 of 14 in the attached python notebook. The goal was to extract the eigenvectors from the intermediate matrices while we were implementing transformations on the initial 3×3 matrix. The iterative Jacobi solver then returned various dot products every other iteration. If the dot products returned values approximately equal to 0, then the vectors are still orthogonal.

The outputs in Out[17] show that the dot products in our unit test yielded values of the order 10^{-16} or smaller. There is one random output of 0.00372, but this can be explained by a lagging dot product associated with loss of numerical precision at the end of the matrix before the "clean" function is implemented. The clean function merely scans the matrix and removes points set below a certain tolerance. I set said tolerance to approximately 10^{-9}. The values in the matrix were small prior to this cleaning, and the carrying forward floating points, which we found to be a problem in our error analysis in Project 1.

We also conducted testing to see whether or not orthogonality was preserved in the matrix that we used to compare to the analytic results. The dot products yield ≈ 0 values. Thus, we could see this preservation of orthogonality in the larger matrices as well. See last page of ipython notebook.

This unit testing proved that the orthogonality was conserved carrying forward through multiple transformations. Multiple other unit tests were performed throughout our analysis. The first 8 input cells of the ipython notebook are actually unit tests of the individual functions before we compiled them into one large function that could fulfill all of the needed tests. We then repeated unit tests of the fully compiled functions for a 2×2 matrix and two 3×3 matrices. This was to test whether or not the eigenvalue solver was achieving the correct eigenvalues without scaling up to the large size matrices. These tests can be seen in outputs, In[8] (page 4 of 14), Out[15] (page 7 of 14), and Out[19] (page 9 of 14). We found that implementing unit tests was critically important for our overall solutions because it allowed for a more step-by-step natured problem-solving environment as well as checking our results before we were no longer able to debug.

Moving forward in whatever projects we work on, implementing unit testing and small scale trial runs of functions and numerical methods in our programs will be included for the sake of ensuring proper functionality and properties, as we were able to show with the preservation of orthogonality and proper functionality carrying forward from small scale to large scale matrices.

2.4 Comparison to Analytic Solution

Following the completion of our Jacobi eigenvalue solver, one of the tasks we wanted to complete was to look at how our numerically calculated eigenvalues compared to the analytic solutions provided by Taut [1].

The example that we chose to evaluate to compare to the analytic solutions was the case where $\omega = 0.25$. Taut describes this as $\frac{1}{\omega}$, but in our case it was much simpler to define the value of ω as we had previously.

In this analytic result, the ground state energy, corresponding to the lowest eigenvalue, was given as $\epsilon_s = 0.6250$. Through our numerical analysis, the Jacobi eigenvalue we created yielded an eigenvalue of 1.2436. There was an ancillary factor of two involved in the way that they defined the rearranged Schrödinger equation. Thus, we yield a final ground state energy, or lowest eigenvalue, of $\epsilon = 0.6218$. This gives a 0.5% error when compared to the published analytic result.

From this, we can conclude that with proper rescaling of our ρ_{max} value, we can achieve the same results proposed by Taut. With respectably low percent error values as well. In short, our numerical methods are successful in comparison to the published analytic results.

2.5 Time Comparison

We evaluated a test for a 40×40 Matrix in the interacting case to see how the Jacobi method compared to the built in eigensolvers that numpy offers. Sample code found in our final program, not the attached pdf ipyhton notebook, yielded the following:

```
A = makemat(40,.25,40)
eigenvals = esolver(A,.00001)
cleaned = clean(eigenvals)
print(cleaned)
[  0.62181583   1.06526775   1.47963271   1.85087367   2.14990337
   2.40874635   2.77068912   3.22976528   3.76667366   4.37568805
   5.05435585   5.80137175   6.61596051   7.49762833   8.44604449
   9.4609788   10.54226605  11.68978461  12.90344296  14.18317098
  15.52891409  16.94062916  18.41828171  19.96184385  21.57129279
  23.24660973  24.98777904  26.79478762  28.66762441  30.60628002
  32.61074643  34.68101672  36.81708493  39.0189459   41.28659509
  43.6200286   46.01925596  48.48604367  51.10945762]


startJ = time.clock()
esolver(A,.00001)
dTJ = time.clock()-startJ

startL = time.clock()
la.eigvals(A)
dTL = time.clock()-startL

print('Jacobi:',dTJ)
print('Linalg',dTL)

Jacobi: 0.20473100000000066
Linalg 0.0008759999999998769
```

The results were surprising to us. We know that our numerical method is not the most computationally efficient method, but we were surprised to see that even for a relatively small matrix, the numpy functions beat Jacobi by 3 orders of magnitude. That means that our numerical approach was approximately 1000× slower than defined functions already at our disposal. We can only imagine that for large matrices, but not exceedingly large to run in to memory issues with storing matrices, we could expect that factor to grow by some unknown order of magnitude.

In summary, analysis has shown a clear comparison to established functions and resulted in several orders of amgnitude difference.

2.6 Comparing Results for Different ω

We wanted to look at some more values aside from ω=0.25, which we used to compare to the analytic results give by Taut. We decided to test $\omega_r = 0.01$, $\omega_r = 0.5$, $\omega_r = 1$, and $\omega_r = 5$.

ω	Lowest Eigenvalue, ϵ
0.01	0.10793
0.5	2.2399
1	4.1020
5	17.7836

The table above summarizes the reuslts we got for varying values of omega in the interacting case. For low values of ω, we see that the lowest eigenvalues are relatively low, corresponding to low energy systems in the well. As we increase ω to the order of magnitude of 1, we see eigenvalues similar to the non-interacting case. As omega continues to get larger and larger, the potential well becomes narrower, so our energy values would be expected to be larger because the electrons would be forced to be in close proximity, showing the dominance of the Coulombic repulsion at short distances. Keeping electrons at a short distance would increase their potential energy, so we would expect larger eigenvalues.

Now, there could be some issues with our numerical results. Unfortunately, these values of frequency do not correspond to any of the analytic results found by Taut. The values are similar, but hist results are for ω <1. This no longer becomes a useful reference for our results when we exceed the maximum ω values. However, for ω=0.01, which would correspond to 100 in Taut's results, we saw ϵ, the lowest eigenvalue of the system, on the order of Magnitude of 10^{-2}, which, if we divide our merica result in half (as we do in comparing results to Taut based on differences in Mathematical assumptions between our approaches), then we see that our lowest eigenvalue is on the same order of magnitude and approximately 0.05, which resembles Taut's results to within 10%. [1] This gives us hope in our results that our numerical solutions are approximately correct.

Further resources should be saught out to compare the larger values of ω for comparison sake. We do see the expected trends in our results for varying values of ω Numerical analysis will be included in the ipython notebook attached to this report.

3 Future Work

There are a number of interesting problems that we could look in to exploring more after working on this project. First and foremost, we have a program written now that can effectively, albeit not effeciently, solve eigenvalues for a symmetric tridiagonal matrix. This matrix can also be adapted to imbibe any potential we see fit to introduce to the electrons (or other applicable physical scenarios) as well as varying parameters for the given potential, both interacting and non-interacting cases. This could lead to some very interesting results when it comes to electrons in a harmonic oscillator potential. While we didn't explore this much, one thing we expect to see is that when we make the value of ω larger and larger, correpsonding to a wider potential well, the $\frac{1}{r}$ term in the Coulombic potential becomes dominant and we see that the electrostatic term dominates in the eigenvalue. Physically speaking, this is where we would be able to see the long range interactions fo this force at work. We could conduct some more extensive numerical investigations by scaling our value for ω and see if we get the results we expect. Our selected values for ω listed in subsection 2.6 could be a much longer list and we could even consider making some plots of the lowest level eigenvalue varying with our frequency. [2]

We also only compared a couple of analytic solutions to those provided by Tout. [1] One in particular for verification that we could get the expected results of ω=0.25. Once we achieved one comparable numerical result, we decided our model could be effectively adapted to coincide with the analytic solutions to this problem. Further work could be conducted for the sake of comparing more results.

Further work could be done in translating our program into a C++ program and utilizing this algorithm in conjunction with armadillo (lapack and blas functions) to explore two main ideas. First, does C++ offer a more effecient implementation of this algorithm than Python? We expect this to be true given what we know about computation. And, how does the effeciency of the C++ program compare to the armadillo libraries? In Python, we saw that Jacobi's algorithm was not a very effecient means of solving for eigenvalues when compared to the numpy linear algebra functions, and it would be interesting to see if C++ is similar in nature in that regards and by what magnitude.

4 Conclusion

We have found that the Jacobi method is a slower way of finding the eigenvalues of a matrix compared to the built in eigenvalue solver function. But when we have large matrices, simply storing it could take up a significant amount of memory, which is one of the downfalls of using the built in functions. They require a defined matrix to work, which may be impractical with a sufficiently large matrix. The Jacobi method is one solution to a matrix that is so large that it is not feasible to store it, where we can create a matrix using vectors, while still being able to find eigenvalues to what is returned from the built in functions.

We were also able to get comparable results from our model to analytic results and explroe unit testing in our programs. We found unit testing to be invaluable in developing our functions. This will be a concept we carry forward in future numerical work. We offer some ideas for future work and consideration in regards to this problem.

All functions and outputs can be referenced in the following ipython notebook and in our source code included in the final project.

References

[1] M. Taut. *Two Electrons in an External Oscillator Potential: Particular Analytic Solutions of a Coulomb Correlation Problem*. Physical Review A, November, 1993.

[2] Morten Hjorth-Jenson. *PHY 480 Github*. `https://github.com/CompPhysics/ComputationalPhysicsMSU`. 2016-2017.

[3] Strang, G. *Linear Algebra and its Applications, Fourth Edition*. Thomson Learning Inc., 2006.

[4] Kelley, C.T.. *Iterative Methods for Linear and Nonlinear Equations*. Society for Industrial and Applied Mathematics, 1995.