FYS4150 - Computational Physics
Project 2

Alex Ho

30. september 2016

1 Introduction

In this project, we will have a look at the so called *Eigenvalue problem*. We will use the well known Schrödinger equation from quantum mechanics and apply it to a system consisting of a single electron in a harmonic oscillator potential. To achieve this, we will implement Jacobi's method to solve the Schrödinger equation and then find the eigenvalues, which will physically be the energies of the electrons.

We will also look at the case where we have two interacting electrons in a three dimensional harmonic oscillator, with different frequencies.

2 Method

2a) Mathematical intermezzo

We first want to look at the orthogonality properties of a specific basis. Let us consider the following basis of vectors

$$\mathbf{v}_i = \begin{pmatrix} v_{i1} \\ v_{i2} \\ \vdots \\ v_{in} \end{pmatrix} \tag{1}$$

We will also assume that this is an orthogonal basis

$$\mathbf{v}_j^T \mathbf{v}_i = \delta_{ij} \tag{2}$$

We want to look at the orthogonality properties for a orthogonal and unitary vector transformation in the form

$$\mathbf{w}_i = \mathbf{U}\mathbf{v}_i \tag{3}$$

For an orthogonal matrix, we know that $\mathbf{U}^T\mathbf{U} = \mathbf{I} = 1$, where $\mathbf{I}$ is the identity matrix, so we have that

$$\mathbf{w}_j^T\mathbf{w}_i = (\mathbf{U}\mathbf{v}_i)^T(\mathbf{U}\mathbf{v}_i) = \mathbf{v}_j^T\mathbf{U}^T\mathbf{U}\mathbf{v}_i = \mathbf{v}_i^T\mathbf{v}_i = \delta ij \tag{4}$$

For a unitary matrix, we have that $\mathbf{U}^*\mathbf{U} = \mathbf{I}$ and $\mathbf{U}^\dagger\mathbf{U} = \mathbf{I}$, where $\mathbf{U}^\dagger = (\mathbf{U}^*)^T$ is the hermitian conjugate. Doing the hermitian conjugate on the transformation vector, we can show that

$$\mathbf{w}_j^\dagger\mathbf{w}_i = (\mathbf{U}\mathbf{v}_j)^\dagger(\mathbf{U}\mathbf{v}_i) = \mathbf{v}_j^\dagger\mathbf{U}^\dagger\mathbf{U}\mathbf{v}_i = \mathbf{v}_j^\dagger\mathbf{v}_i = \delta_{ij} \tag{5}$$

Here we have assumed that the basis vector is real, so $\mathbf{v}_i^* = \mathbf{v}_i$.

2b) Jacobi's rotation method

Jacobi's rotation method (or Jacobi's method) is an algorithm that can be used to solve the eigenvalue problem. Consider an $n \times n$ orthogonal transformation matrix

$$\mathbf{S} = \begin{pmatrix} 1 & 0 & \cdots & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 & \cdots & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & \cos\theta & 0 & \cdots & 0 & \sin\theta \\ 0 & 0 & \cdots & 0 & 1 & \cdots & 0 & 0 \\ 0 & \cdots & \cdots & \cdots & \cdots & & & \\ 0 & 0 & \cdots & 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & \cdots & -\sin\theta & 0 & \cdots & 0 & \cos\theta \end{pmatrix} \tag{6}$$

with the property $\mathbf{S}^T = \mathbf{S}^{-1}$. A similarity transformation, on a matrix $\mathbf{A}$

$$\mathbf{B} = \mathbf{S}^T\mathbf{A}\mathbf{S} \tag{7}$$

gives us

$$b_{ii} = a_{ii}, \quad i \neq k, i \neq l \tag{8}$$
$$b_{ik} = a_{ik}\cos\theta - a_{il}\sin\theta, \quad i \neq k, i \neq j \tag{9}$$
$$b_{il} = a_{il}\cos\theta + a_{ik}\sin\theta, \quad i \neq k, i \neq j \tag{10}$$
$$b_{kk} = a_{kk}\cos^2\theta - 2a_{kl}\cos\theta\sin\theta + a_{ll}\sin^2\theta \tag{11}$$
$$b_{ll} = a_{ll}\cos^2\theta + 2a_{kl}\cos\theta\sin\theta + a_{kk}\sin^2\theta \tag{12}$$
$$b_{kl} = (a_{kk} - a_{ll})\cos\theta\sin\theta + a_{kl}(\cos^2\theta - \sin^2\theta) \tag{13}$$

The angle θ is arbitrary. The idea here is to choose a θ that causes all non-diagonal matrix elements to become zero (or within a given tolerance). However, not all non-diagonal matrix elements will become zero after one iteration, so we will have to run this algorithm a sufficient amount of times to get the desired result.

First we have to choose a value a_{ij}. This value will be the largest non-diagonal value in the matrix $\mathbf{A}$ and needs to be larger within a tolerance ϵ, that is

$$\max(a_{ij}^2) > \epsilon \tag{14}$$

Since we want the non-diagonal matrix elements to become zero, we require that $b_{kl} = b_{lk} = 0$, which gives

$$b_{kl} = (a_{kk} - a_{ll})\cos\theta\sin\theta + a_{kl}(\cos^2\theta - \sin^2\theta) = 0 \tag{15}$$

Defining $\tau = \frac{a_{ll}-a_{kk}}{2a_{kl}}$ and $\tan\theta = t = \frac{\sin\theta}{\cos\theta}$, we can write the above expression as

$$\begin{aligned} -2a_{kl}\tau t + \frac{a_{kl}}{\cos^2\theta}(\cos^2\theta - \sin^2\theta) &= 0 \\ \implies -2\tau t + 1 - t^2 &= 0 \\ \implies t^2 + 2\tau t - 1 &= 0 \end{aligned} \tag{16}$$

Where we have in the first line inserted the expression for τ and then divided by $\cos^2\theta$. We can now use this second order equation to determine $\tan\theta$.

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

Once that is determined, one can easily calculate the new values of $\cos\theta$ and $\sin\theta$ given as

$$\cos\theta = \frac{1}{\sqrt{1+\tan^2\theta}} \tag{18}$$

$$\sin\theta = \cos\theta\tan\theta \tag{19}$$

When these are calculated, we can compute the new values of b which results in a new matrix. We then repeat the same procedure, with this new matrix, until

$$\max(a_{ij}^2) > \epsilon \tag{20}$$

is no longer satisfied. The end result is that the matrix $\mathbf{A}$ will only contain values along the diagonal, which will be the eigenvalues (or energies) of the system. Note that the eigenvalues along the diagonal are not ordered from the lowest to highest eigenvalue (or vice versa).

The reason we decided to choose the largest value of the non-diagonal elements is because of this test. It ensures that all the non-diagonal elements will be close to zero, thus giving us the diagonal matrix with the desired eigenvalues.

2c) Unit tests

From exercise 2a, we have shown that the orthogonality is preserved for an orthogonal and/or unitary vector transformation. The Jacobi's method uses an orthogonal transformation, so one can expect that the resulting eigenvectors are remains orthogonal.

For a $n \times n$ matrix, we will have n eigenvectors when we have diagonalized the matrix $\mathbf{A}$. We first have to calculate how many possible vector dot product combinations there are for all these n eigenvectors. Let us first consider $n = 2$, which gives us two eigenvectors so the only vector product combination we have is $\mathbf{v}_1 \cdot \mathbf{v}_2$. For $n = 3$ we have

$$\mathbf{v}_1 \cdot \mathbf{v}_2, \quad \mathbf{v}_1 \cdot \mathbf{v}_3, \quad \mathbf{v}_2 \cdot \mathbf{v}_3$$

That is three combinations, for $n = 4$

$$\mathbf{v}_1 \cdot \mathbf{v}_2, \quad \mathbf{v}_1 \cdot \mathbf{v}_3, \quad \mathbf{v}_1 \cdot \mathbf{v}_4$$
$$\mathbf{v}_2 \cdot \mathbf{v}_3, \quad \mathbf{v}_2 \cdot \mathbf{v}_3, \quad \mathbf{v}_3 \cdot \mathbf{v}_4$$

Which is 6 combinations. For $n = 5$ one can show that there are 10 combinations. We can clearly see a recursive pattern here, and can by this say that

$$\text{Possible vector dot product combinations} = \frac{1}{2}n(n-1) \tag{21}$$

This will already give us an exact estimate on how many orthogonal combinations we can expect from the eigenvectors.

3 Implementation

For the C++ program, the first part was to implement the algorithm. I first create a function that initializes the matrix **A** as well as creating an identity matrix **R** which will contain the eigenvectors when **A** is fully diagonalized. In the Jacobi's method function, I also calculate the eigenvector after one iteration.

Setting the tolerance to $\epsilon = 10^{-8}$, I run the algorithm until $\max(a_{ij}^2) > \epsilon$ is no longer satisfied. Each iteration is counted to give us an idea on how many iterations is required to diagonalize **A**, which now contains the eigenvalues. As mentioned, the eigenvalues in the diagonalized matrix **A** is not ordered in ascending order, so I have to do it by using the *sort* function from the *algorithm* library.

For the interacting case, we use the exact same code but with different values of ω. In this part I will also save the eigenvector for the ground state which is saved within the matrix **R**. However, like the diagonalized matrix **A**, the eigenvectors in **R** are also not ordered properly, so the program will have to find the eigenvector that corresponds to the ground state eigenvalue by doing certain tests.

4 Results

In the C++ program, by selecting the number of mesh points to $n = 400$, with $\rho_{max} = 10$, the output for the non interacting case is the following:

```
Using a 400x400 matrix (n = 400)
with rho_max = 10
```

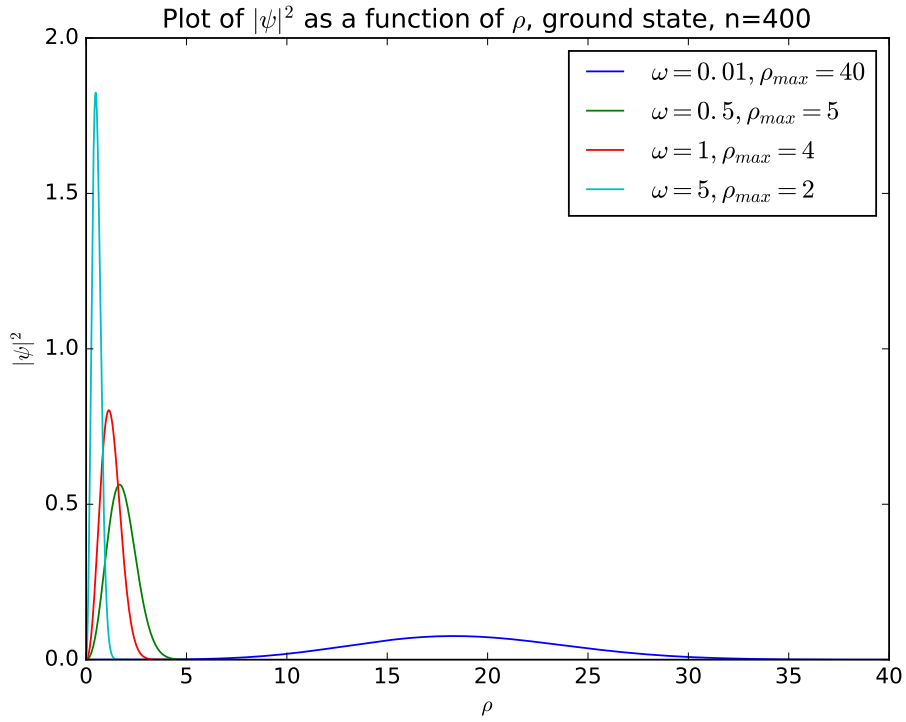
```
Max (absolute) value in the test matrix: 1000
max_diag does give out the largest value
max_diag result: 1000

Starting Jacobi's algorithm for a single electron
Time elapsed for non interactive case: 400.941s
Number of iterations: 260506
Lowest 3 eigenvalues are:
2.99981
6.99903
10.9976
Possible number of orthogonal vectors: 79800
Number of orthogonal vectors: 79800
Number of non-orthogonal vectors: 0

Using Armadillo to compare the time

Lowest 3 eigenvalues of the Armadillo simulation:
2.99981
6.99903
10.9976
Time elapsed for Armadillo's algorithm: 0.347s
```

The eigenvalues, even though they are not within 4 decimal precision, are very close to the exact analytical eigenvalues. We can also see that the number of iterations needed to fully diagonalize the matrix $\mathbf{A}$ (or at least within a tolerance ϵ) was 260506 iterations. The unit test of the orthogonality between the eigenvectors is also shown, where every possibly combination of the scalar product between the eigenvectors are orthogonal (also within a tolerance). The unit test, where the largest matrix element in the test matrix was -1000, also worked out.

The time it took for the program to run was roughly 6 minutes. This seems like a very long simulation time, so there is probably something that is not quite optimal in the program, thus causing the program to run for a longer time. Compared to Armadillo's *eig_sym*, we can clearly see that Armadillo is much faster, using less than a second to finish. The lowest three eigenvalues are identical to our Jacobi's method, so Armadillo's method can

serve as a good unit test for the eigenvalues.

Figure 1 shows the the probability distribution of the wave function (or the eigenvectors we calculated), which have been normalized, for different frequencies ω. These results are not quite surprising. Let us for a moment



Figur 1: Probability distribution of the (normalized) wave functions of the ground state, for different values of ω. Plotted for $n = 400$ meshpoints.

imagine that two electrons are bound together by a spring. The electrons oscillate back and forth due to the oscillation of the spring. They interact once when they are closest to each other (that is, the distance between them is the minimum).

For lower frequencies, the electrons will interact less frequently which means, by the analogy above, they are further apart from each other a lot more often. Another way to think of this is that the spring can be drawn over a larger distance before it is forced to contract, thus allowing the electrons to go further out. This can be seen from figure 1 (the blue line with $\omega = 0.01$)

where the probability of them being close to each other is very small, while them being further apart is slightly larger.

For higher frequencies, the electrons will interact more frequently. Using the same analogy, we can think of the electrons bouncing back and forth very quickly. The spring is very stiff, so it cannot be stretched that far. Because of this, the electrons are forced to be very close to each other, which is exactly what we see in figure 1. In the case of $\omega = 5$ (the cyan line), we can clearly see that the probability of them being very close to each other is very large, while them being further apart is very low (practically zero).

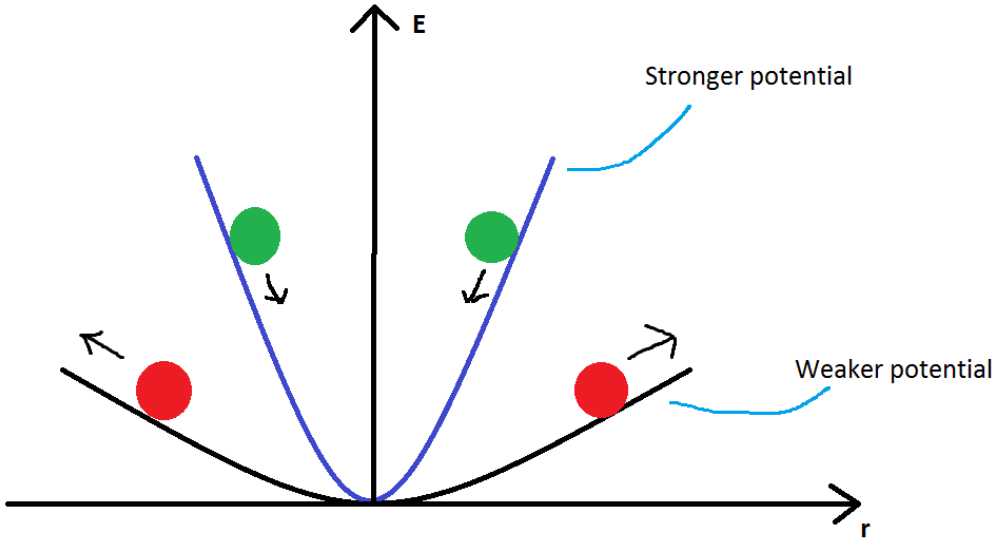
We also compared the interacting and non-interacting case. From figure (???) we can see that the distance between the electrons are closer in the non-interactive case compared to the interactive case. The electrons will just oscillate back and forth when they are not affected by the Coulomb repulsion, and the max distance between the electrons are determined by the energy. In the interacting case however, the Coulomb repulsion causes the electrons to be pushed further apart. It serves as an extra pushing force between the electrons when they are close to each other.

Let us first think of a new analogy. We want to roll a ball upward a hill. In this case, the ball is the electron and the hill is the potential. If the hill is very steep (stronger potential), the distance the ball travels will be very short before it falls back down. On the other hand, if the hill is not very steep (weaker potential), the ball will be able to travel further away from us before it is forced to roll back. Figure 3 shows an illustration of this.

The probability distribution for the interacting and non-interacting case differ quite significantly for lower frequencies. The potential is not very steep for lower frequencies, so the electrons will not fall back down the potential as quick. With Coulomb repulsion, the electrons will be pushed further apart as the potential will not drag the electrons back fast enough.

With higher frequencies, the probability distributions will be very much alike. Higher frequencies gives a steeper potential, so the electrons will fall back down quicker because they are unable to climb the potential fast enough. Even with Coulomb repulsion, the electrons will not get very far due to the steep potential.

A different, but interesting, result can be seen in figure 2. What we can see here is the tunnelling effect. The red line shows the potential while the green line is the energy of the eigenvector. The energy and potential crosses each other at two points. What this means is that the distance of the electrons is bound between these two points and it cannot be anywhere out-



Figur 2: An illustration of two different potentials and how they affect the electron.

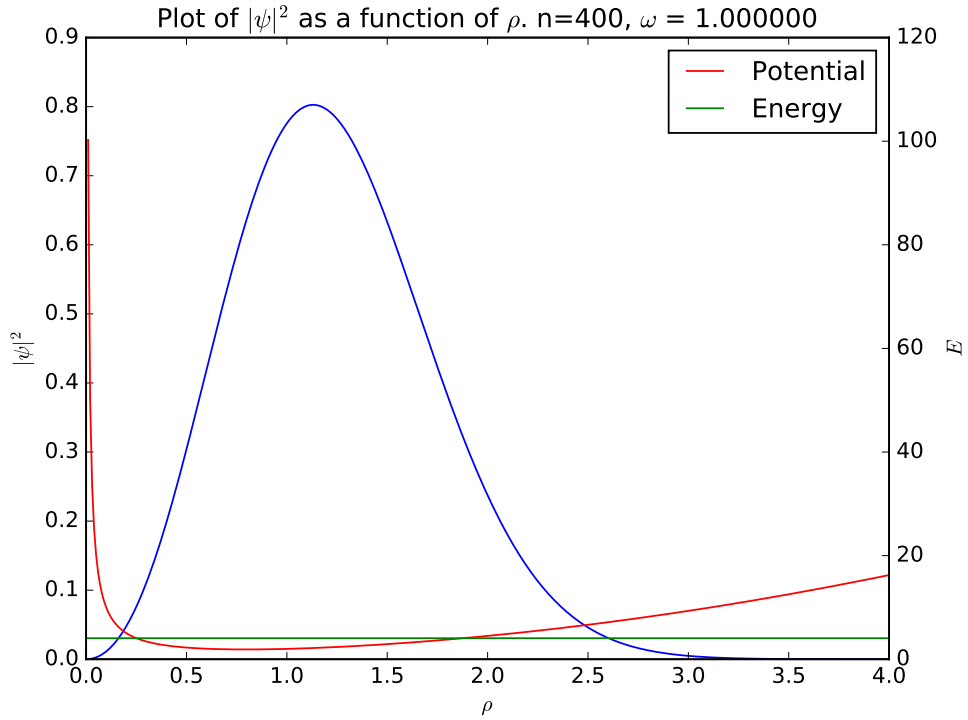
side them. However, the probability distribution says that the distance of the electrons can be smaller/larger than these two points. In classical mechanics, this would not be possible, but this is possible in quantum mechanics due to the tunnelling effect.

5 Conclusion

The run time of the Jacobi's method is quite slow, which implies that this algorithm is not exactly the smartest algorithm to use in order to solve eigenvalue problems. This is especially true if we compare Jacobi's method to other algorithms, e.g. the eig_sym function of Armadillo, which finished in less than a second, compared to the 6 minutes in the C++ program.

Our eigenvalue results did not reach 4 decimal precision, but did come very close. One could possibly fix that by increasing the number of mesh points n as well as ρ_{max}. However, as shown from the results, increasing n would also increase the run time of the program, which is probably not practical for our purpose.

From the results, we saw that the average distance between the electrons



Figur 3: A plot showcasing the tunnelling effect. Plot for $\omega = 1$.

will be larger when the harmonic oscillator potential is smaller (and vice versa for larger potentials). One can also see that the Coulomb repulsion will cause a larger effect for smaller potentials.

6 References

M. Hjort-Jensen, 2015, *Computational Physics*, accessible at course GitHub repository; `https://github.com/CompPhysics/ComputationalPhysics/tree/master/doc/Lectures` (as of 14.09.16), 551 pages.