University of Oslo

FYS3150

Computational Physics

Project 2 - Schrödinger's equation for two electrons in a three-dimensional harmonic oscillator well

Author:
Vebjørn Gilberg

Author:
Trond-Wiggo Johansen

October 2016

Contents

Abstract

In computational quantum mechanics it is of great interest to solve Schrödinger's equation in a simple and efficient way. In this project we will solve this equation by reformulating it in a discretized form as an eigenvalue equation, $\boldsymbol{Au} = \lambda \boldsymbol{u}$, and solve it by Jacobi's method. First we will solve it for one electron and then for two electrons. We will make use of unitary operators to diagonalize $\boldsymbol{A}$ through a series of rotations, ultimately resulting in a diagonal matrix $\boldsymbol{B}$. Too se if Jacobi's method is time consuming we will test it against a specialized Armadillo[1] function called ***eig_sym***. We compare the numerical results with known eigenvalues $\{\lambda_0, \lambda_1, \lambda_2\}$ to an accuracy of 4 leading digits. In the case where we have two electrons we compare the results to analytic solutions from an article by M. Taut [1], and the plots correspond to what one would expect classically with decreasing ω_r. We treat ω_r as a parameter which reflects the strength of the oscillator potential.

Github repository:
`https://github.com/wiggoen/UiO/tree/master/FYS3150/Project_2`

1 Introduction

The harmonic oscillator potential is perhaps one of the best known potentials there is, because nearly everything in physics can be approximated to fit that particular potential. In this project we will look at Scrödinger's equation for one and two electrons in a three-dimensional harmonic oscillator well with and without a repulsive Coulomb interaction where we will assume spherical symmetry. We want to solve the equation with Jacobi's method in a discretized form as an eigenvalue equation and compare the three lowest energy eigenvalues to the analytical results. To test the speed of the method we compare Jacobi's rotation algorithm to a standard function in the Armadillo library called ***eig_sym***.

[1] Armadillo is a C++ linear algebra library. It can be found at: http://arma.sourceforge.net

2 Theory

2.1 Single electron

Given that the hamiltonian for the three dimensional system is spherically symmetric, it is very useful to transform from Cartesian to spherical coordinates which ultimately gives us solutions of the Schrödinger equation which are separable and dependent on r, ϕ and θ. The radial part of the three-dimensional Schrödinger equation reads

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

with energy $E_{nl}=\hbar\omega\left(2n+l+\frac{3}{2}\right)$ where ω is the oscillatory frequency and $\forall n,l\in\mathbb{N}$ starting from 0. Moreover, n is the energy level and l tells us about the orbital momentum of the electron. Nearly everything in physics, including this problem, can be approximated to fit the harmonic oscillator potential which is defined as $V(r)=\frac{1}{2}kr^2=\frac{1}{2}mw^2r^2$. Furthermore, because we've transformed our equation from Cartesian to spherical coordinates we must define r on the interval $r\in[0,\infty)$, i.e. $r\geq 0$.

It's a lot easier to work with a dimensionless equation, so we can substitute $R(r)=\frac{1}{r}u(r)$ with boundary conditions $u(0)=u(\infty)=0$, and introduce a dimensionless variable $\rho=r/\alpha$ where α is a constant and has dimension length. Thus

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

In this project, our only area of interest when it comes to l will be at $l=0$. Also, when we insert $V(\rho)=(1/2)k\alpha^2\rho^2$ and multiply both sides by $2m\alpha^2/\hbar^2$ we get

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

We define the eigenvalues as $\lambda\equiv(2m\alpha^2/\hbar^2)E$ and we can fix the constant α by $\alpha^4mk/\hbar^2=1$ such that $\alpha=(\hbar^2/mk)^{1/4}$. This ultimately leads us to the final analytic form of the Schrödinger equation, namely

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

Now, we want to solve this numerically, so we must discretize the second derivative using the usual definition.

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

The step length we will be working with is defined as

$$h = \frac{\rho_N - \rho_0}{N + 1}$$

where $\rho_0 = 0$ and $\rho_{max} = \rho_N$. The i-th element of ρ takes the value $\rho_i = \rho_0 + ih$ for $i \in \mathbb{N}_0$. With this in mind, we can rewrite this equation as

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

where the potential $V_i = \rho_i^2$. We want to break this down into a numerical linear algebra problem, and to do so we can define the diagonal and the non-diagonal matrix elements as $d_i = \frac{2}{h^2} + V_i$ and $e_i = -\frac{1}{h^2}$, respectively. Also, the non-diagonal elements e_i do not change. The Schrödinger equation now reads

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

which is the matrix equation, $\boldsymbol{Au} = \lambda \boldsymbol{u}$, that we want. If we write out the equation we get

$$\begin{bmatrix} \frac{2}{h^2} + V_1 & -\frac{1}{h^2} & 0 & 0 & \dots & 0 & 0 \\ -\frac{1}{h^2} & \frac{2}{h^2} + V_2 & -\frac{1}{h^2} & 0 & \dots & 0 & 0 \\ 0 & -\frac{1}{h^2} & \frac{2}{h^2} + V_3 & -\frac{1}{h^2} & 0 & \dots & 0 \\ \vdots & \dots & \dots & \dots & \dots & \dots & \vdots \\ 0 & \dots & \dots & \dots & -\frac{1}{h^2} & \frac{2}{h^2} + V_{N-2} & -\frac{1}{h^2} \\ 0 & \dots & \dots & \dots & \dots & -\frac{1}{h^2} & \frac{2}{h^2} + V_{N-1} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ \vdots \\ \vdots \\ u_{N-1} \end{bmatrix} = \lambda \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ \vdots \\ \vdots \\ u_{N-1} \end{bmatrix} \tag{1}$$

2.2 Two electrons

As we mentioned in the introduction, we are going to work with two electrons, and so the state u must be dependent on two positions $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$. In the non-interacting case the Schrödinger equation reads

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

where we denote $E^{(2)}$ as the energy for the two electrons. When the two electrons interact electrostatically we introduce two new vectors, the relative vector $\boldsymbol{r} = \boldsymbol{r}_1 - \boldsymbol{r}_2$ and the center of mass vector $\boldsymbol{R} = 1/2(\boldsymbol{r}_1 + \boldsymbol{r}_2)$ which yields

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

Again, we can apply separation of variables so that we get an equation which is dependent on r. Moreover, we can go through similar steps as we did for the single electron case where we ended up with a dimensionless equation dependent on ρ. It's important to note that the energy changes accordingly, $E^{(2)} = E_r + E_R$. If we add the Coulomb potential $V = \beta e^2/|\boldsymbol{r}_1 - \boldsymbol{r}_2|$ where $\beta e^2 = 1.44$ eVnm we get the following equation for $\psi(r)$

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

We can use the exact same dimensionless variable $\rho = r/\alpha$. Furthermore, we define a new frequency $\omega_r^2 = (mk\alpha^4/4\hbar^2)$, which reflects the strength of the oscillator potential, and eigenfunctions $\lambda = \frac{m\alpha^2}{\hbar^2}E$. This yields

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

The only difference now is that our diagonal elements is on the form $d_i = \frac{2}{h^2} + \omega_r^2\rho_i^2 + \frac{1}{\rho_i}$.

2.3 Unitary Operators

An operator U is unitary if $U^\dagger U = UU^\dagger = \mathbb{I}$ and is therefore a particular isometric operator.[2]

Definition. *Let X and Y be metric spaces with metrics d_X and d_Y. A map $T : X \to Y$ is called an isometry if $\forall a, b \in X$ we have*

$$d_Y(f(a), f(b)) = d_X(a, b)$$

[2]In simpler terms than the definition, isometric operators preserve distances such that $||T(x-y)|| = ||x-y||$, $\forall x, y$ in that space. In terms of the inner product, this implies that $T^*T = \mathbb{I}$.

However, not all isometric operators are unitary. One property of unitary operators is that they must be surjective, which is not a demand for isometries. To get a better understanding of these kinds of operators, we can use an analogy between complex numbers and unitary operators where $u = e^{i\theta}$. In the same way $U^\dagger U = \mathbb{I}$, $u^* u = 1$. We usually say that unitary operators are generalized rotation operators from $\mathbb{V}^3(R)$ to $\mathbb{V}^n(C)$ because they preserve the lengths of vectors and their dot products as we will prove below.[2]

Theorem. *In Hilbert space, unitary operators preserve the inner product and orthogonality of the vectors they act on.*

Proof. Let $\boldsymbol{w}_i = U\boldsymbol{v}_i$ and $\boldsymbol{v}_j^T \boldsymbol{v}_i = \delta_{ij}$. Then

$$\begin{aligned}\langle \boldsymbol{w}_j, \boldsymbol{w}_i \rangle &= \langle U\boldsymbol{v_j}, U\boldsymbol{v}_i \rangle \\ &= \langle \boldsymbol{v}_j, U^\dagger U \boldsymbol{v}_i \rangle \\ &= \langle \boldsymbol{v}_j, \boldsymbol{v}_i \rangle \\ &= \delta_{ij}\end{aligned}$$

□

3 Methods and Algorithms

3.1 Jacobi's Method

Before we go into detail on Jacobi's method we urge you to have a look at the appendix on similarity transformations if you're not familiar with it.

The idea of Jacobi's method is to eliminate all of the non-diagonal elements so that we're left with the diagonal which will contain the eigenvalues because the eigenvalues are invariant under the similarity transformations. This will be accomplished by implementing the rotation matrix $\boldsymbol{S}^T = \boldsymbol{S}^{-1}$ as a similarity transformation, that is $\boldsymbol{A} \mapsto \boldsymbol{S^T A S = B}$ where $\boldsymbol{A}$ is a symmetric tridiagonal matrix. The operation is

illustrated below

$$\begin{bmatrix} * & & & \dots & & & * \\ & \ddots & & & & & \\ & & a_{kk} & \dots & a_{kl} & & \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ & & a_{lk} & \dots & a_{ll} & & \\ & & & & & \ddots & \\ * & & & \dots & & & * \end{bmatrix} \mapsto \begin{bmatrix} * & & & \dots & & & * \\ & \ddots & & & & & \\ & & b_{kk} & \dots & 0 & & \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ & & 0 & \dots & b_{ll} & & \\ & & & & & \ddots & \\ * & & & \dots & & & * \end{bmatrix}$$

We will consider an $n \times n$ orthogonal transformation matrix

$$\boldsymbol{S} = \begin{bmatrix} 1 & 0 & \cdots & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 & \cdots & 0 & 0 \\ \vdots & \dots & \ddots & \vdots & \vdots & \cdots & \vdots & \vdots \\ 0 & 0 & \cdots & \cos\theta & 0 & \cdots & 0 & \sin\theta \\ 0 & 0 & \cdots & 0 & 1 & \cdots & 0 & 0 \\ \vdots & \dots & \cdots & \vdots & \cdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & \cdots & -\sin\theta & 0 & \cdots & 0 & \cos\theta \end{bmatrix}$$

that performs a plane rotation around an angle θ in the Euclidean n-dimensional space. In other words, we only rotate the coordinate system to diagonalize $\boldsymbol{A}$. This means that the matrix elements that differ from zero are given by

$$s_{kk} = s_{ll} = \cos\theta, \quad s_{kl} = -s_{lk} = -\sin\theta, \quad s_{ii} = -s_{ii} = 1, \quad \text{where } i \neq k \text{ and } i \neq l$$

with a similarity transformation $\boldsymbol{B} = \boldsymbol{S}^T\boldsymbol{A}\boldsymbol{S}$ we get

$$\left.\begin{aligned} b_{ii} &= a_{ii} \\ b_{ik} &= a_{ik}\cos\theta - a_{il}\sin\theta \\ b_{il} &= a_{il}\cos\theta + a_{ik}\sin\theta \end{aligned}\right\} \; i \neq k \text{ and } i \neq l$$

$$\begin{aligned} b_{kk} &= a_{kk}\cos^2\theta - 2a_{kl}\cos\theta\sin\theta + a_{ll}\sin^2\theta \\ b_{ll} &= a_{ll}\cos^2\theta + 2a_{kl}\cos\theta\sin\theta + a_{kk}\sin^2\theta \\ b_{kl} &= (a_{kk} - a_{ll})\cos\theta\sin\theta + a_{kl}(\cos^2\theta - \sin^2\theta) \end{aligned}$$

where we want to choose the angle θ so that all non-diagonal matrix elements b_{kl} become zero.

In order for the computer to understand what to do, we will have to do this in steps which means that we have to make the off-diagonal elements smaller and smaller. To check when they are small enough for the computer to call them zero we will implement the Frobenius norm of $\boldsymbol{A}$ which will run until the norm is smaller than some tolerance ϵ. The Frobenius norm is defined as

$$||\boldsymbol{A}||_F = \sqrt{\sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n}|a_{ij}|^2}$$

We will write a function that implements Jacobi's rotation algorithm in order to solve equation (1), but first we define the trigonometric functions

$$\sin\theta = s \qquad \cos\theta = c \qquad \tan\theta = t = \frac{s}{c}$$

and

$$\cot 2\theta = \tau = \frac{a_{ll} - a_{kk}}{2a_{kl}} \qquad \text{where } l, k \in \mathbb{N}$$

We will define the angle θ so that the non-diagonal matrix elements of the transformed matrix a_{kl} become non-zero and by using the trigonometric relation

$$\cot 2\theta = \frac{1}{2}(\cot\theta - \tan\theta)$$

we obtain the quadratic equation

$$t^2 + 2\tau t - 1 = 0$$

resulting in obtaining tangent, cosine and sine by

$$t = -\tau \pm \sqrt{1+\tau^2} \qquad c = \frac{1}{\sqrt{1+t^2}} \qquad s = tc$$

We have structured our program with two classes where one of them contains the potentials, and the other solves the problem using the Jacobi algorithm. Our jacobi function is as follows

```
    // Runtime for the Jacobi method
    clock_t start, finish;
    // Start clock
    start = clock();

    while (fabs(max_OffDiag) > epsilon && (double) iterations <
       max_number_iterations) {
```

```
        max_OffDiag = maxOffDiag(A);
        rotate(A, Z);
        iterations++;
    }
    // End clock
    finish = clock();
```

maxOffDiag(A) finds the largest value for $|A_{kl}|$, that is, it finds the largest l and k for which the former is true. In the while loop we have included the function **rotate(A, Z)** which applies N similarity transformations to $S^T AS$ when the program has run. Since we're dealing with large matrices, it's a good idea to change the rest of the elements by hand.

```
// Rotation of the matrix
  double a_kk, a_ll, a_ik, a_il, r_ik, r_il;
  a_kk = A(k, k);
  a_ll = A(l, l);

  // Changing the matrix elements with indices k and l
  A(k, k) = c*c*a_kk - 2.0*c*s*A(k, l) + s*s*a_ll;
  A(l, l) = s*s*a_kk + 2.0*c*s*A(k, l) + c*c*a_ll;

  // Hard coding zeros
  A(k, l) = 0.0;
  A(l, k) = 0.0;

  // Change the remaining elements
  for (int i = 0; i < N; i++) {
      if (i != k && i != l) {
          a_ik = A(i, k);
          a_il = A(i, l);
          A(i, k) = c*a_ik - s*a_il;
          A(k, i) = A(i, k);
          A(i, l) = c*a_il + s*a_ik;
          A(l, i) = A(i, l);
      }
      // Computing the new eigenvectors
      r_ik = Z(i, k);
      r_il = Z(i, l);
      Z(i, k) = c*r_ik - s*r_il;
      Z(i, l) = c*r_il + s*r_ik;
  }
```

4 Code validation

To test the mathematical properties of our algorithm we implemented two unit tests using Catch[3]. Unit tests are important to test that our code works as expected. The first test searched for the largest non-diagonal element and returned the correct answer. When implementing this we had to make a symmetric 3×3 test matrix that we knew all the values of. We called the function **maxOffDiag** in our System class using the test matrix and checked that we got the right results. The code is listed below. You can read the code and we will give a simple explanation of it afterwards.

```
TEST_CASE( "Find max value", "Approximation" ) {
    int N = 3;

    // test matrix
    mat test_mat = { {1, 3, 2},
                     {3, 4, 6},
                     {2, 6, 7} };

    // getting max value from maxOffDiag
    System* system = new System(N);
    double max_value = system->maxOffDiag(test_mat);
    int &l = system->l;
    int &k = system->k;

    // testing that the matrix gets the correct largest non-diagonal
       elements in upper triangle
    REQUIRE(max_value == Approx(6));

    // hard coding elements to zero and testing for the next elements
    test_mat(k, l) = 0; test_mat(l, k) = 0;
    max_value = system->maxOffDiag(test_mat);
    REQUIRE(max_value == Approx(3));

    test_mat(k, l) = 0; test_mat(l, k) = 0;
    max_value = system->maxOffDiag(test_mat);
    REQUIRE(max_value == Approx(2));

    test_mat(k, l) = 0; test_mat(l, k) = 0;
    max_value = system->maxOffDiag(test_mat);
    REQUIRE(max_value == Approx(0));
}
```

The matrix starts indexing at 0. First the program will find that 6 is the largest

[3]Catch is an automated test framework for C++ and Objective-C. It can be found at: https://github.com/philsquared/Catch

non-diagonal element, and return the values of $l = 1$ (row) and $k = 2$ (column). After this we hard code the test matrix to have zeros at (k, l) and (l, k). Then we use the same test to find the next largest non-diagonal element, and return the values of l and k. This goes on until the test matrix has all zeros at the non-diagonal elements. Since we use computers we don't get the exact value of the non-diagonal elements, but we approximate against the largest values using the **Approx()** function.

The second test we implemented was to check if we got the same and correct eigenvalues for a simple matrix, irrespective of changes made. We used a symmetric 5×5 test matrix that we knew the eigenvalues of. The 'test_mat' and the 'test_mat_Z' is used to test that the Jacobi method does what it is suppose to do. It should give us the correct eigenvalues of the matrix. You can read the code below.

```
TEST_CASE( "Testing eigenvalues", "Approximation" ) {
    int N = 5;

    // test matrix
    mat test_mat = { {1, 2, 3, 4, 5},
                     {2, 1, 2, 3, 4},
                     {3, 2, 1, 2, 3},
                     {4, 3, 2, 1, 2},
                     {5, 4, 3, 2, 1} };

    // setting up the rotation matrix
    mat test_mat_Z = eye<mat>(N, N);

    // known eigenvalues from test matrix
    double lambda_0 = -5.236067977499790;
    double lambda_1 = -1.635237730041817;
    double lambda_2 = -0.763932022500212;
    double lambda_3 = -0.556294915312374;
    double lambda_4 = 13.191532645354183;

    // using Jacobi method to find the eigenvalues
    System *test_matrix = new System(N);
    test_matrix->Jacobi_method(test_mat, test_mat_Z);

    // setting up the eigenvalue vector
    vec eigvals = zeros<vec>(N);
    for (int i = 0; i < N; i++) {
        eigvals(i) = test_mat(i, i);
    }
    // sorting the eigenvalues
    eigvals = sort(eigvals);
```

```
    // testing that the eigenvalues are correct
    REQUIRE(eigvals(0) == Approx(lambda_0));
    REQUIRE(eigvals(1) == Approx(lambda_1));
    REQUIRE(eigvals(2) == Approx(lambda_2));
    REQUIRE(eigvals(3) == Approx(lambda_3));
    REQUIRE(eigvals(4) == Approx(lambda_4));
}
```

After running the tests we got an output telling us if the tests passed or failed. We got this output

```
All tests passed (9 assertions in 2 test cases)
```

which feels good.[4]

5 Results and discussion

5.1 Single electron

We implemented the Jacobi method discussed in section 3.1 which was tested against the prewritten Armadillo function **eig_sym** from the Armadillo library. It's a good thing that the three lowest eigenvalues are known, because then we can find the parameters which reproduce those numbers as close as possible. The way we find the fitting ρ_{max} and N is simply by trying multiple values, and since we want them to be as close to the analytic result as possible we need 4 leading digits. With $\rho_{max} = 5.0$ and $N = 350$ we found a good fit for λ_0 as well as for λ_1 and λ_2. They all agree quite good with the known eigenvalues $\{3, 7, 11\}$ as shown in *table 1* below, in fact, after about $N = 250$ we can see that we get four leading digits.

The similarity transformations roughly increase with a factor 4 from $N = 50$ to $N = 100$, but further down the line it's closer to a factor of 2. We can estimate how the similarity transformations depend on N by simple trail and error (or by using a two variable regression analysis using GeoGebra[5]), and if we do so we find that it runs as $S(N) \approx 1.7N^2$. One could argue that we should use a better method

[4] We could have implemented more unit tests, i.e. to make sure that the unitary transformation preserves the dot product and orthogonality of the obtained eigenvectors, but we didn't do that in this project.

[5] GeoGebra is a graphing calculator for functions, geometry, algebra, calculus, statistics and 3D math. It can be found at: https://www.geogebra.org

Grid points N	Eigenvalues computed with Jacobi's method	$Runtime_{Jacobi}$ [sec]	$Runtime_{Arma}$ [sec]	Similarity transformations
50	$\{2.99674, 6.98370, 10.96029\}$	0.06483	0.02067	3997
100	$\{2.99918, 6.99593, 10.99013\}$	0.49762	0.05309	16462
150	$\{2.99963, 6.99819, 10.99563\}$	1.23624	0.02979	37228
200	$\{2.99979, 6.99898, 10.99759\}$	3.58941	0.07705	66813
250	$\{2.99986, 6.99935, 10.99847\}$	8.71784	0.10106	104485
300	$\{2.99990, 6.99954, 10.99897\}$	17.59154	0.09754	151043
350	$\{2.99993, 6.99966, 10.99927\}$	33.95408	0.21353	205634

Table 1: Table of the three lowest eigenvalues computed at different values for N with a fixed $\rho_{max} = 5.0$. The runtime increases drastically when we go from $N = 250$ to $N = 300$ and up to $N = 350$.

of finding how the similarity transformations behave as a function of N, but that would be tedious for our purposes. The three first wavefunctions corresponding to $N = 350$ and $\rho_{max} = 5.0$ is shown in *figure 1.*

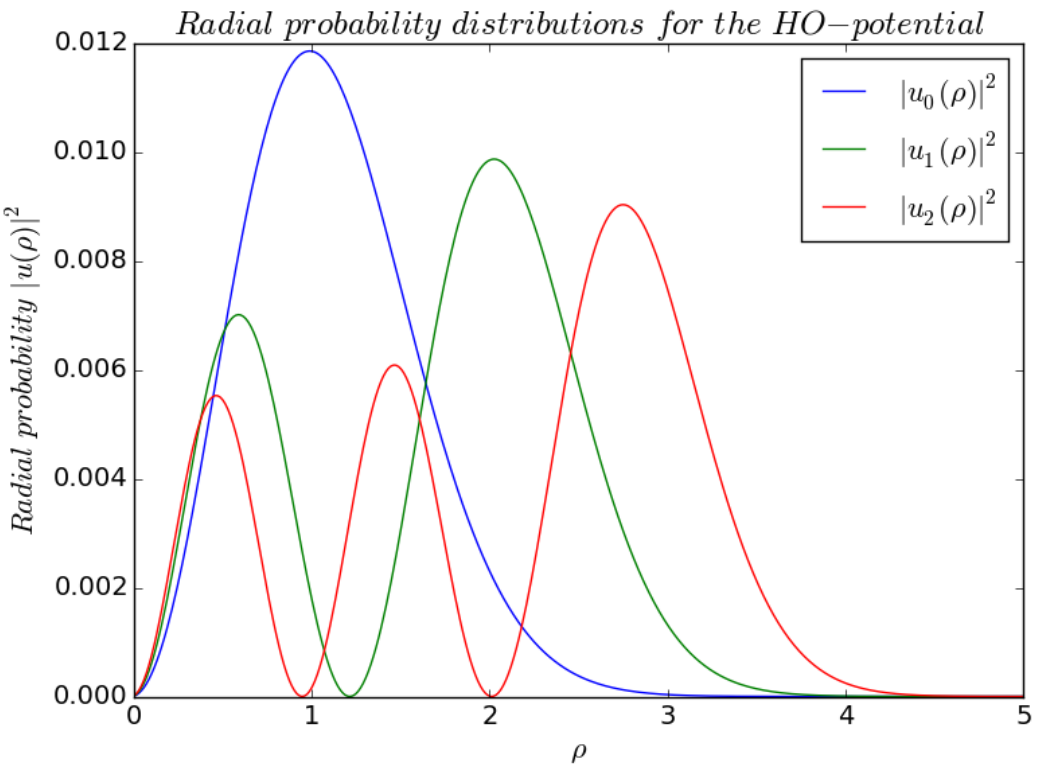

Figure 1: The three lowest probability distributions were plotted at $\rho_{max} = 5$ and $N = 350$ with the harmonic oscillator potential in three dimensions for one electron.

There should be no question as to which method is the quickest when it comes to finding the eigenvalues and eigenvectors when we look at table 1. At $N = 50$ the runtime for **eig_sym** is about 3 times lower than the Jacobi method. However, as N increases, we quickly realise that the Jacobi method is incredibly inferior to the **eig_sym** function as far as runtime[6] is concerned.

5.2 Two electrons

In section 2.2 we said that the most notable change for our purposes is the new potential which results in a new matrix $\boldsymbol{A}$. The program written to deal with the single electron in the harmonic oscillator potential can be altered to solve the

[6]To get better runtime results we should have run the program at least ten times for each N, taking the average of time used. This is because the PC runs other processes simultaneous and will use more or less time for each run.

Schrödinger equation for two electrons with and without the Coulomb interaction which is what we have done. Several plots are included in *figure 2* and *3* to illustrate the behavior of the ground state as we increase ω_r.

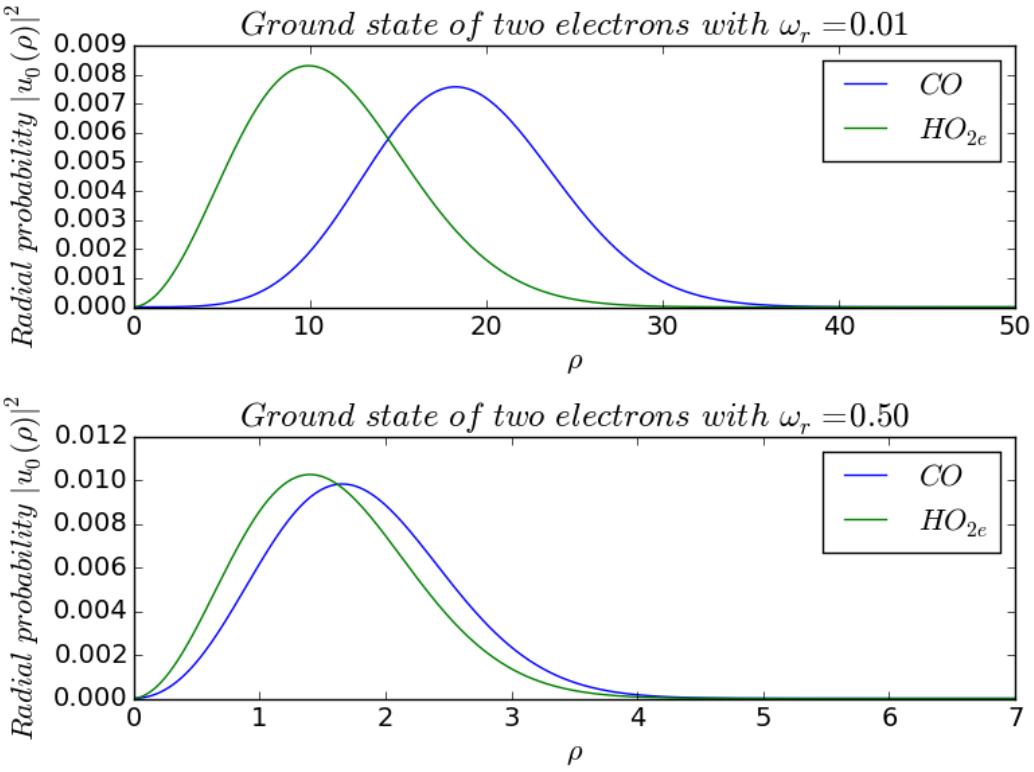

Figure 2: The plots consists of the ground state with and without the Coulomb interaction. HO_{2e} corresponds to the non-interacting case and CO corresponds to the interacting case. In the first plot we've used $\rho_{max} = 50$, $N = 500$ and $\omega_r = 0.01$. In the second, $\rho_{max} = 7$, $N = 400$ and $\omega_r = 0.5$. We can clearly see that the second plot for the Coulomb interaction has moved farther towards the curve without interaction.

In *figure 2* and *3*, we can see a trend, namely that the probability distribution where we include the Coulomb interaction move more towards the non-interacting case. This is what we would expect classically. What this means is that when the oscillatory frequency is low, the Coulomb interaction becomes more dominant and the particles are farther apart and the distribution gets wider. In the case where oscillatory frequency is higher, we get a less dominant Coulomb interaction meaning that the electrons are closer together ultimately resulting in a probability

distribution which is close to the non-interacting case, but not quite due to the fact that there still is an interaction between the electrons.

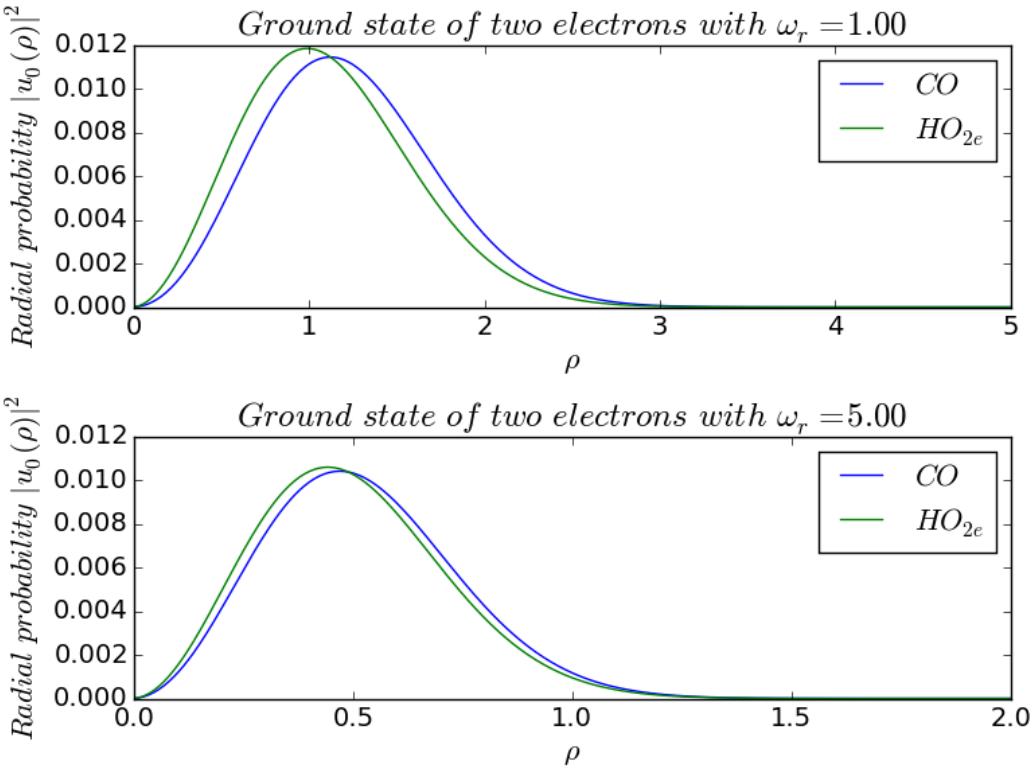

Figure 3: The plots consists of the ground state with and without the Coulomb interaction. HO_{2e} corresponds to the non-interacting case and CO corresponds to the interacting case. In the first plot we've used $\rho_{max} = 5$, $N = 350$ and $\omega_r = 1.0$. In the second, $\rho_{max} = 2$, N is the same and $\omega_r = 5.0$. As in figure 2, we can clearly see that the second plot for the Coulomb interaction has moved farther towards the curve without interaction.

Earlier we estimated that the number of similarity transformations as a function of the dimensionality N runs as $S(N) \approx 1.7N^2$. Although it took quite some time to run the rotation algorithm on a symmetric matrix, the full non-symmetric matrix case would require many more similarity transformations and thus a lot more time to run. In fact, it would require about $5N^2$ transformations in the full non-symmetric case [3].

Oscillatory frequency, ω_r	Eigenvalue λ_0 of the groundstate with Jacobi's method	Reference article $2\epsilon^{'}$[1]
0.05	0.34998	0.35000
0.25	1.24985	1.25000

Table 2: Table of the eigenvalue given by the ground state computed at $N = 350$ with varying $\rho_{max} \in \{50, 30\}$ with the Coulomb interaction. These values are compared with the ones from the article by M. Taut [1].

It is of importance that our program produces the results of the article quite accurately, because then we know that our program does what it's supposed to. Although we could alter ρ_{max} and N to get an even better fit, we think this is close enough to illustrate the success of the method. In this context, success does not take into account the time used to run the program, of course. The eigenvalues used in the article had to be multiplied by two because of the difference in the nature of the article and this report.

6 Conclusion

The world is much more chaotic than the impression we get when we study basic quantum mechanics. There aren't that many systems we can solve analytically compared to the amount of quantum mechanical systems which exists. However, the case where we had one electron is an exactly solvable problem. We discretized the Schrödinger equation for the three dimensional harmonic oscillator potential into a matrix equation with a tridiagonal matrix $\boldsymbol{A}$ where we used a time consuming algorithm named Jacobi's method to rotate $\boldsymbol{A}$ in order to obtain a diagonal matrix containing the eigenvalues. The Armadillo function ***eig_sym*** produced the same results in a fraction of the time spent running the Jacobi algorithm. Moreover, the known eigenvalues and those that our algorithm produced was in good agreement to about 4 leading digits when we adjusted N and ρ_{max}.

When we modified the program to handle two electrons, we compared the non-interacting and the interacting cases for $\omega_r \in \{0.01, 0.05, 0.25, 0.5, 1.0, 5.0\}$. This showed a behavior which we would also expect classically, namely that the probability distributions spread out when the electrons interact with smaller ω_r, and thus when $\omega_r \to 5.0$ the distribution tends toward the non-interacting case. To validate

that our results we conducted a more thorough investigation where we compared the eigenvalues we got with the article by M. Taut [1], and they were successfully reproduced.[7]

7 Appendix

7.1 Similarity transformations

In a similarity transformation, the resulting matrix has the same eigenvalues, but the eigenvectors are in general different. We assume that $\boldsymbol{A}$ is a $n \times n$ real and symmetric matrix that has n eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_n$. If we let $\boldsymbol{D}$ be the diagonal matrix with the eigenvalues on the diagonal, then there exists a real orthogonal matrix $\boldsymbol{S}$ such that

$$\boldsymbol{S}^T\boldsymbol{A}\boldsymbol{S} = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)$$

and for $j = 1$ to $j = n$ we have $\boldsymbol{A}\boldsymbol{S}(:, j) = \lambda_j \boldsymbol{S}(:, j)$. The strategy to obtain eigenvalues of $\boldsymbol{A}$ is to perform a series of similarity transformations on the original matrix $\boldsymbol{A}$, in order to reduce it either into a tridiagonal form or into a diagonal form. A matrix $\boldsymbol{B}$ is a similarity transform of $\boldsymbol{A}$ if

$$\boldsymbol{B} = \boldsymbol{S}^T\boldsymbol{A}\boldsymbol{S} \qquad \text{where} \qquad \boldsymbol{S}^T\boldsymbol{S} = \boldsymbol{S}^{-1}\boldsymbol{S} = \boldsymbol{I}$$

If we do a similarity transformation on our matrix $\boldsymbol{A}$ then we get

$$\begin{aligned}
\boldsymbol{A}\boldsymbol{x} &= \lambda\boldsymbol{x} \\
\boldsymbol{S}^T\boldsymbol{A}\boldsymbol{x} &= \boldsymbol{S}^T\lambda\boldsymbol{x} \\
\boldsymbol{S}^T\boldsymbol{A}\boldsymbol{I}\boldsymbol{x} &= \lambda(\boldsymbol{S}^T\boldsymbol{x}) \qquad\qquad \boldsymbol{I} = \boldsymbol{S}\boldsymbol{S}^T \\
\boldsymbol{S}^T\boldsymbol{A}\boldsymbol{S}\boldsymbol{S}^T\boldsymbol{x} &= \lambda(\boldsymbol{S}^T\boldsymbol{x}) \\
(\boldsymbol{S}^T\boldsymbol{A}\boldsymbol{S})(\boldsymbol{S}^T\boldsymbol{x}) &= \lambda(\boldsymbol{S}^T\boldsymbol{x}) \\
\boldsymbol{B}(\boldsymbol{S}^T\boldsymbol{x}) &= \lambda(\boldsymbol{S}^T\boldsymbol{x})
\end{aligned}$$

We see that λ is an eigenvalue of $\boldsymbol{B}$ as well as $\boldsymbol{A}$, but with the eigenvector $\boldsymbol{S}^T\boldsymbol{x}$ instead of $\boldsymbol{x}$.

If we want to obtain the eigenvalues in a diagonal matrix, we can apply subsequent similarity transformations so that

$$\boldsymbol{S}_N^T \ldots \boldsymbol{S}_1^T\boldsymbol{S}_1 \ldots \boldsymbol{S}_N = \boldsymbol{D}$$

[7] We didn't get exactly the same values, but we got close enough to illustrate our point.

Another way is to apply subsequent similiarity transformations so that $\boldsymbol{A}$ becomes tridiagonal, and then use techniques for obtaining eigenvalues from tridiagonal matrices. [3]

7.2 Unitary and hermitian matrices

There is a relationship between all unitary operators and hermitian operators in the sense that we can write any unitary operator as

$$U = exp(iC) \tag{2}$$

where C is hermitian. We can write U as a sum of two hermitian operators A and B such that

$$U = A + iB$$
$$U^\dagger = A - iB$$

Because U is unitary we can write

$$U^\dagger U = A^2 + B^2 + i[A, B] = \mathbb{I}$$
$$UU^\dagger = A^2 + B^2 - i[A, B] = \mathbb{I}$$

which means that A and B must commute and also that $A^2 + B^2 = \mathbb{I}$. The magic trick is that now we're allowed to write A and B as

$$A = cosC$$
$$B = sinC$$

And this, in turn, implies eq. (2).[4] We take advantage of this in quantum mechanics when we deal with the time evolution of our systems. That is, eq. (2) is essentially how we apply the unitary operator to states in order for them to evolve in time.

References

[1] M. Taut. Two electrons in an external oscillator potential: Particular analytic solutions of a coulomb correlation problem. Phys. Rev. A 48, 3561 (1993).

[2] Ramamurti Shankar. Principles of quantum mechanics, second edition. pg 28, chp 1. Kluwer Academic, New York, 1994.

[3] Morten Hjort-Jensen. Computational physics, lecture notes fall 2015, chapter 7. Department of Physics, University of Oslo, 2015. `https://github.com/CompPhysics/ComputationalPhysics/blob/master/doc/Lectures/lectures2015.pdf`.

[4] Stephen M. Barnett. Quantum information. Oxford University Press, first edition, 2009.