Project 2

FYS4150 Computational Physics

Even S. Håland

Abstract

1 Introduction

The purpose of this project is to develop a program for solving eigenvalue problems by using Jacobi's algorithm.

2 The Schrödinger equation

(This section follows very closely the theoretical introduction given project description, but for completeness sake I thought it would be nice to also include it in the report.)

2.1 One-particle case

We start by considering the radial part of the Schrödinger equation (SE) for one electron, which is given as

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

where (in our case) $V(r)$ is the harmonic oscillator (HO) potential

$$V(r)=\frac{1}{2}kr^2,$$

where $k=m\omega^2$. E is then the energy of the three dimensional HO, ω is the oscillator frequency, and these quantities are related by

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

where $n=0,1,2,\ldots$ and $l=0,1,2,\ldots$. Throughout this project only cases with $l=0$ will be considered.

By making the substitution $R(r)=(1/r)u(r)$ and introducing the dimensionless variable $\rho=(1/\alpha)r$ eq. (1) can be simplified to

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

with boundary conditions $u(0)=u(\infty)=0$. Further we can multiply the equation by $2m\alpha^2/\hbar^2$, and fix α so that

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

If we also define

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

we end up with the eigenvalue equation

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

We must of course make a discrete approximation to the equation in order to solve the problem numerically. However, before moving on to that we should have a quick look at the changes that are introduced by adding a second electron to the problem.

2.2 Two-particle case

The radial SE for two electrons in an HO potential without any interactions is given by

$$\left(-\frac{\hbar^2}{2m}\frac{d^2}{dr_1^2} - \frac{\hbar^2}{2m}\frac{d^2}{dr_2^2} + \frac{1}{2}kr_1^2 + \frac{1}{2}kr_2^2\right)u(r_1, r_2) = E^{(2)}u(r_1, r_2), \tag{3}$$

where $E^{(2)}$ is the two-electron energy. The solution to this equation is just the product of the wave functions for each electron. However, when we introduce the Coulomb interaction (which depends on the distance r between the electrons) eq. (3) is not very useful. For that reason we would like to introduce a new set of coordinates, namely the relative distance between the electrons,

$$\mathbf{r} = \mathbf{r}_1 - \mathbf{r}_2,$$

and the centre-of-mass coordinate for the system,

$$\mathbf{R} = \frac{1}{2}(\mathbf{r}_1 + \mathbf{r}_2).$$

The SE (still without interactions) can then be written as

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

where $r = |\mathbf{r}_1 - \mathbf{r}_2|$ and $R = \frac{1}{2}|\mathbf{r}_1 + \mathbf{r}_2|$. We then assume that the wave function is separable, so that $u(r, R) = \psi(r)\phi(R)$, and that the total energy is given by the sum of relative energy, E_r, and centre-of-mass energy, E_R, i.e.

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

We can then add the term for the Coulomb interaction, which is given by

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

where $\beta e^2 = 1.44$ eVnm, and the r-dependent part of the SE becomes

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

As we did for the one-electron SE we now introduce the dimensionless variable $\rho = r/\alpha$, and scale the equation appropriately. After a few steps of manipulation we arrive to

$$-\frac{d^2}{d\rho^2}\psi(\rho) + \omega_r^2\rho^2\psi(\rho) + \frac{1}{\rho} = \lambda\psi(\rho), \tag{4}$$

where

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

with α fixed so that

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

and λ is defined as

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

It is noteworthy that the only differences between equations (2) and (4) is the factor ω_r^2 in the HO potential term, and the Coulomb repulsion term. This means that it is quite easy to make the transition between these cases when writing the code, which is in fact the main point of doing the scaling.

2.3 Discrete Schrödinger equation

As mentioned previously we must make a discrete approximation to the SE, and when doing so we will jump back to the one-electron problem, i.e. eq. (2).

The second derivative is approximated (up to $O(h^2)$) by

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

where h is the step length. The minimum value of ρ is $\rho_{min} = \rho_0 = 0$, while the maximum value is in principle infinity. However, infinity is not a very practical "value" to work with, especially in a numerical context, which means that we must choose an appropriate maximum value $\rho_{max} = \rho_N$. The step length h is then defined as

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

where N is the number of mesh points we consider, and ρ is given as

$$\rho_i = \rho_0 + ih,$$

with $i = 1, 2, \ldots, N$. By using the short-hand notation $u(\rho_i + h) = u_{i+1}$, we can write the SE as

$$-\frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} + V_i u_i = \lambda u_i, \tag{5}$$

where $V_i = \rho_i^2$ is the HO potential. This is now an eigenvalue problem which can be compactly written as $\mathbf{Au} = \lambda\mathbf{u}$, where $\mathbf{A}$ is a matrix with diagonal elements

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

and non-diagonal elements

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

If we instead want to consider the two-electron case with the Coulomb interaction we simply change the potential V_i from ρ_i^2 to $\omega_r^2\rho_i^2 + 1/\rho_i$.

3 Jacobi's method

To summarize the situation we have now reduced the Schrödinger equation to an eigenvalue problem of the form $\mathbf{Au} = \lambda\mathbf{u}$, with $A \in \mathbb{R}^{n\times n}$. Our task is then to find the n eigenvectors $(\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_n)$ and eigenvalues $(\lambda_1, \lambda_2, \ldots, \lambda_n)$ of A.

The idea behind Jacobi's method is to do a series of orthogonal transformations of the kind

$$(\mathbf{SAS}^T)(\mathbf{Sv}_i) = \mathbf{Sv}_i,$$

where $\mathbf{S}$ is an orthogonal matrix satisfying $\mathbf{SS}^T = \mathbf{I}$, and $\mathbf{v}_i$ is an orthogonal basis of $\mathbb{R}^n$. Our goal is then to eventually end up with a matrix on the left-hand side where all non-diagonal elements are (close to) zero, which means that the elements on the diagonal are (close to) the eigenvalues.

This method works because when we do an orthogonal transformation

$$\mathbf{B} = \mathbf{SAS}^T,$$

the eigenvalues of $\mathbf{B}$ are the same as those of $\mathbf{A}$, and it can be shown that [1] if $\mathbf{A}$ is real and symmetric (which it is in our case) there exists an orthogonal matrix, $\mathbf{M}$, such that

$$\mathbf{MAM}^T = diag(\lambda_1, \ldots, \lambda_n).$$

(A more careful discussion of this is found in for example refs. [1, 2].)

We can also see that if the basis vectors, $\mathbf{v}_i$, are orthogonal, that is[1]

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

then after an orthogonal transformation

$$\mathbf{w}_i = \mathbf{Sv}_i,$$

[1] This relation actually states that the $\mathbf{v}_i$'s are *orthonormal*, and not just orthogonal.

the dot product of the new vectors is

$$\begin{aligned}\mathbf{w}_j^T\mathbf{w}_i &= (\mathbf{S}\mathbf{v}_j)^T\mathbf{S}\mathbf{v}_i \\ &= \mathbf{v}_j^T\mathbf{S}^T\mathbf{S}\mathbf{v}_i \\ &= \mathbf{v}_j^T\mathbf{v}_i \\ &= \delta_{ij},\end{aligned}$$

which means that orthogonality (and the dot product) is preserved by orthogonal transformations. This provides us with a nice way of testing our algorithm.

The next step is to choose a basis, $\mathbf{v}_i$, and a transformation matrix, $\mathbf{S}$. The basis vectors are chosen as simply as possible, namely

$$v_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \quad v_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \ldots, \quad v_n = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix},$$

where each vector (of course) has n elements. The transformation matrix is chosen to be the matrix that rotates our system by an angle θ in a plane in the n-dimensional Euclidean space. Such a matrix has elements

$$s_{kk} = s_{ll} = \cos\theta,\ s_{kl} = -s_{lk} = -\sin\theta,\ s_{ii} = 1,$$

where $i \neq k, l$, and for a specific rotation k and l are fixed numbers. All other elements of $\mathbf{S}$ are zero. So when doing the transformation

$$\mathbf{B} = \mathbf{S}\mathbf{A}\mathbf{S}^T,$$

the matrix $\mathbf{B}$ gets the following elements:

$$\begin{aligned}b_{ii} &= a_{ii},\ i \neq k, l \\ b_{ik} &= a_{ik}\cos\theta - a_{il}\sin\theta,\ i \neq k, l \\ b_{il} &= a_{il}\cos\theta + a_{ik}\sin\theta,\ i \neq k, l \\ b_{kk} &= a_{kk}\cos^2\theta - 2a_{kl}\cos\theta\sin\theta + a_{ll}\sin^2\theta \\ b_{ll} &= a_{ll}\cos^2\theta + 2a_{kl}\cos\theta\sin\theta + a_{kk}\sin^2\theta \\ b_{kl} &= b_{lk} = (a_{kk} - a_{ll})\cos\theta\sin\theta + a_{kl}(\cos^2\theta - \sin^2\theta)\end{aligned}$$

Since we eventually want all the non-diagonal elements to be zeros (within some tolerance) we should chose the angle θ so that $b_{kl} = b_{lk} = 0$. From the expression for b_{kl} above we can get the second order equation

$$\tan^2\theta + 2\tan\theta\tau - 1 = 0,$$

where $\tau = (a_{ll} - a_{kk})/2a_{kl}$, which has the roots

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

By using the relations

$$\tan\theta = \frac{\sin\theta}{\cos\theta} \quad \text{and} \quad \cos^2\theta + \sin^2\theta = 1,$$

we find that

$$\cos\theta = \frac{1}{\sqrt{1+t^2}} \quad \text{and} \quad \sin\theta = \cos\theta\tan\theta,$$

which we then apply to our matrix.

For every transformation we do we want to reduce the off-diagonal norm, defined as

$$\text{off}(A) = \sqrt{\sum_i \sum_j |a_{ij}|^2}, \quad i \neq j,$$

such that

$$\text{off}(B) < \text{off}(A),$$

and that this norm in the end should be approximately zero. To reduce the off-diagonal norm as much as possible we start each iteration by picking out the largest off-diagonal element, and hence determine the indices k and l.

4 Programs and implementation

All code written for the project can be found in the following git-repository:

https://github.com/evensha/FYS4150/tree/master/Project2/Programs

The code is mainly written in C++, while some plotting is done with python. The script which are relevant (and will be discussed in the following) are

- `Jacobi_algorithm.cpp`
- `Project2.cpp`
- `Project2_plotting_1p.py`
- `Project2_plotting_2p.py`
- `RunProject.py`

In addition to these scripts the repository also contains a repository named "Output", where all the output from the programs are stored.

4.1 Implementing the Jacobi algorithm

The first program, called `Jacobi_algorithm.cpp`, contains the implementation of the Jacobi algorithm. (The implementation follows quite closely the examples given in refs. [2, 3].) The program itself consists of the three following functions:

- `offdiag`
- `Jacobi_rotation`
- `do_Jacobi`

As mentioned in the previous section we should start by finding the largest off-diagonal element of the matrix we are considering, which is done by the `offdiag`-function. This function takes the matrix $\mathbf{A}$ as input argument, along with the indices of the largest off-diagonal element, and the dimension of $\mathbf{A}$, and returns the largest off-diagonal element as a double.

When we have located the largest off-diagonal element we are ready to perform the Jacobi-rotation, which is done with the `Jacobi_rotation`-function. The first input argument is the matrix, $\mathbf{A}$, on which we want to perform the transformation. The second input argument is the matrix $\mathbf{R}$, which contains the basis vectors. Then follows the indices k and l, which we get from the `offdiag`-function, and the dimension n of the space we are working with. The first thing that is done is to calculate τ, $\tan\theta$, $\cos\theta$ and $\sin\theta$ according to the formulas given in the previous section, and then the updated elements of $\mathbf{A}$ and $\mathbf{R}$ are calculated.

The last function is called `do_Jacobi`, and also takes $\mathbf{A}$, $\mathbf{R}$ and n as input, as well as a vector in which the eigenvalues will be stored. First we initialize $\mathbf{R}$, and define the tolerance and maximum number of iterations we want the algorithm to do. The tolerance is the value that we want to get all off-diagonal elements of $\mathbf{A}$ below, while the maximum number of iterations is just so that the algorithm can't go on "forever". Then we find the initially largest off-diagonal element, and start a "while"-loop that goes on until all off-diagonal elements are below the tolerance (or we have reached the maximum number of iterations). When the iterations are finished the eigenvalues should be on the diagonal of $\mathbf{A}$.

4.2 Solving the Schrödinger equation

5 Results

6 Summary and conclusions

References

[1] G. Golub, C. Van Loan (1996), *Matrix Computations*, John Hopkins University Press.

[2] M. Hjort-Jensen (2015), *Computational Physics - Lecture Notes Fall 2015*, Department of Physics, University of Oslo.

https://github.com/CompPhysics/ComputationalPhysics/blob/master/doc/Lectures/lectures2015.pdf

[3] M. Hjort-Jensen (2017), *Computational Physics Lectures: Eigenvalue Problems*. http://compphysics.github.io/ComputationalPhysics/doc/pub/eigvalues/pdf/eigvalues-beamer.pdf