Project 2, FYS 3150 / 4150, fall 2013

Nathalie Bonatout and Odd Petter Sand

October 8, 2013

1 Introduction

The problem was this time to solve the Schrödinger's equation for a system of two electrons in a three-dimensional harmonic oscillator well. The first step of the project was to do so for a system without a repulsive coulomb interaction, then with this interaction.

To do so, we implemented our own algorithms for the Jacobi Rotation, algorithm taking into account the presence of the interaction, or not. We gave a special attention to the structure of the code. We wanted it to be reusable, and the most convenient possible. And this is why we apart from the equations, we also had a huge fight with the pointers and other nice features of the C++ language.

In this report, you will find our approaches, step by step, to first implement the Jacobi solver, then the Armadillo solver, both of them outputting files which had to be plottable.

(Author's note: This text, and the text introducing the various exercises, is taken from the project description provided at the course website.)

2 Exercises

2.1 Jacobi's solver without Coulomb's interaction

We know that the difference between the matrices A and B is defined by the following expression:

$$\|B - A\|_F^2 = 4(1 - c) \sum_{i=1, i!=k,l}^{n} (a_{ik}^2 + a_{il}^2) + \frac{2a_{lk}^2}{c^2}$$

The main point here is to minimize the difference between A and B, which could be understood as the diagonalization of the matrix.

We already know that $a_{k,l}$ cannot be null since it is the biggest non diagonal element of A. Thus, the only possibility for this difference to tends towards zero is to have the difference $4(c-1)$ tending towards zero.

c can be expressed as

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

Since c should be as closed as possible from 1, we should have t close to 0. We too know that

$$c = s/t$$

so we should have $t \approx s$, which is true if and only if $\|\theta\| \lll \pi/4$.

Test of our algorithm:

We know that the first eigenvalues should be around 3,6 then 11. This is the result we got for rhoMax = 1000, with 1000 steps:

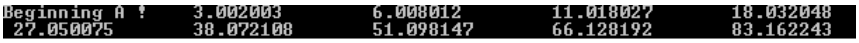

Figure 1: First test of our algorithm

We chose a tolerance threhold of 10^{-7}. And this test is satisfied: we can go on.

2.2 Comparison with the tqli algorithm

We fixed rhoMax at 1000. We are know trying to determine how many points we need in order to get the three lowest eigenvalues with four leading digits.

With experiments, we chose a number of 30 steps. With 30 steps, it implies that around 100 similarity transformations are required in order to have the non diagonal element equal to 0 (equal should be understood here as "$equal \pm toleranceThresold$").

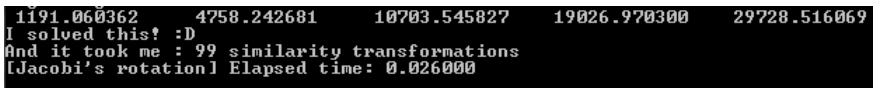

Figure 2: Timing of our Algorithm

So we had to compare the result of our algorithm with the tqli algorithm, found in the lib.cpp file. This algorithm is based on the Householder's algorithm, which is, as we saw it in the lecture, more efficient than the Jacobi's algorithm, doing a reflexion instead of a rotation. Furthermore, the Householder's algorithm is designed for tridiagonal matrices, which is not the case of our algorithm, designed to be used even for the most general cases.

Adding the lib.cpp file in our project was quite difficult, and in the end, the tqli function ran, but always returned a "Too many iterations" message. Thus, we can only predict that this algorithm would have been more efficient and faster than our.

2.3 Jacobi's solver with Coulomb's interaction

We know take into account the repulsive Coulomb's interaction. Thus, our expression of the potential, which was

$$V_i = \rho_i^2$$

now becomes

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

Here ω_r represents the stength of the oscillator potential. We treat four cases, four different values ofω_r:

- $\omega_r = 0.01$
- $\omega_r = 0.5$
- $\omega_r = 1$
- $\omega_r = 5$

And the result was quite rational: the smaller value of $\omega_r$we take, the more amortized the amplitude of the result is.

Without interaction between the two electrons, we only added their two wave functions. But now, we have a wave function dedicated to two electrons. Thus, by changing ω_r, we change the Coulomb force. And when ω_r becomes big, we see that we are getting closer from the "without interaction" case, which can be explained because term in ρ^2gain importance compared to the term in $\frac{1}{\rho^2}$.

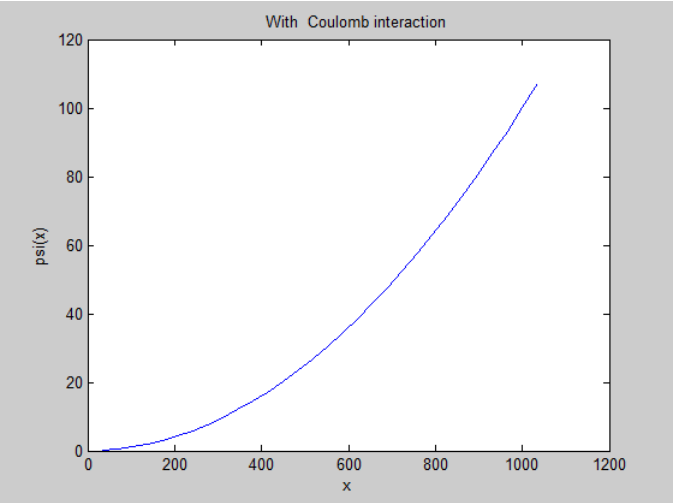

Figure 3: For $\omega_r = 0.01$

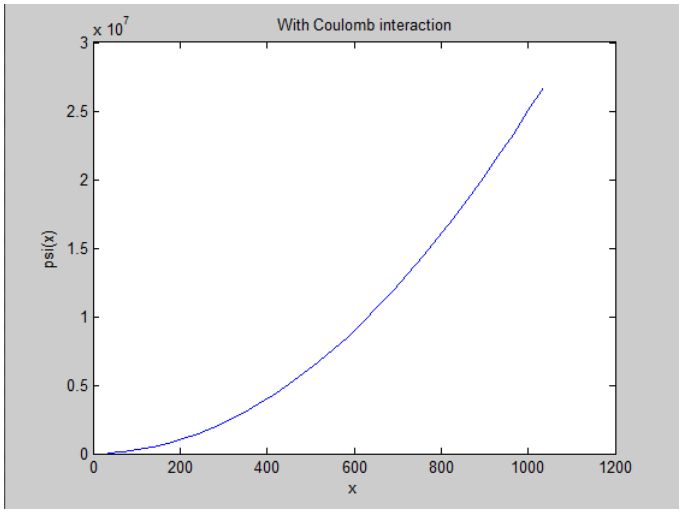

Figure 4: For $\omega_r = 5$

2.4 Plotting and explanations !

So we implemented algorithms for both the non-interaction and the interaction cases. After normalizing our eigenvectors, this is what we got.

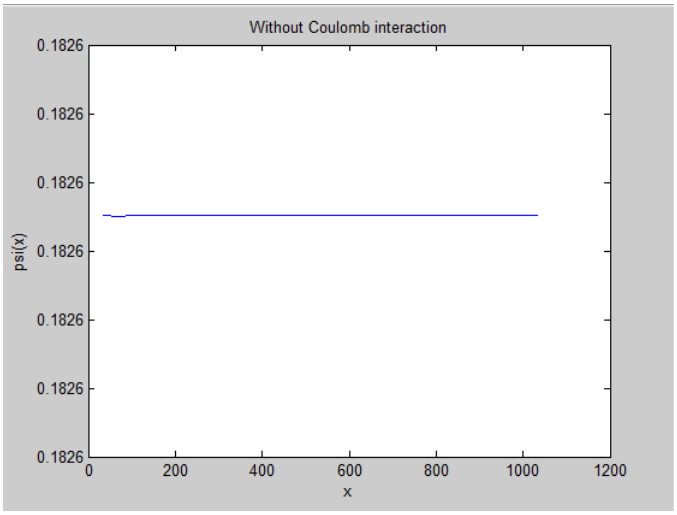

Figure 5: Wave function without Coulomb's interaction

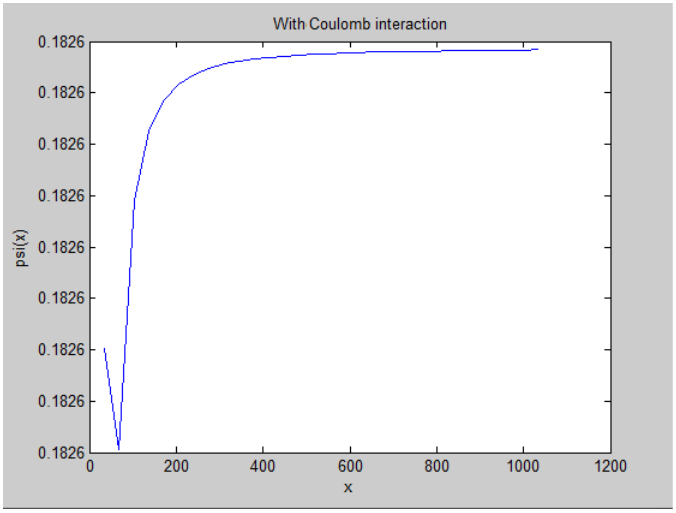

Figure 6: Wave function with Coulomb Interaction for $\omega = 0.01$

We saw that the wave function was more or less amortized according to the value given to ω_r, and the bigger ω_r was, the closer we were from the case without interaction.

3 Source code

All our source code can be found at our GitHub repository: https://github.com/OPSand/Project2

We attached a lot of importance to the structure of the code. We wanted to create a code reusable, and with an easily understandable organization. Thus, we tried to respect the concept of "Model View Controller":

- the main class, Project2.cpp, calling the other functions and fixing the important parameters
- the functional classes, Equation.cpp and Solver.cpp. Each of these classes has inherited classes, such as ArmaSolver.cpp or JacobiSolver.cpp, and contains the most global elements for each type of objects
- the view is dealt with inside the Equation.cpp package: the point here is to export the data towards a .txt file, in order to read them later on with Matlab.

4 Conclusion

From this project, we have learned the following things:

- writing an algorithm for the general case is a bit of an overkill when the problem can be seen as a special case. But writing our own algorithm for this special case is not always needed. Here, many algorithms already written were pretty convenient
- using classes allows to be more organized, and more structured in the writing of the code.

4.1 Comment

We would like to provide the following items of feedback for future versions of this project:

- Including the lib.cpp and lib.h into a Visual Studio project was a bit of a challenge, because to begin with, even the inclusions such as "sys/time.h", which are working pretty fine on Unix systems, do not work under a Windows operating system.