FYS 3150 - Project 2 LIMARE ULTIMI DETTAGLI: CAPTIONS, LABELS, REFS, RIVEDERE INTRO E ABSTRACT, RILEGGERE, INSERIRE FOOT NOTE, READ ME, CONSEGNARE.

Federico Nardi, Davide Saccardo, and Lorenzo Speri
University of Oslo, Department of Physics

(Dated: October 4, 2017)

Abstract In this report we analize how to solve time-independent Schrödinger equation with a given potential. In order to do this we scale the equation and we rewrite it as an eigenvalue problem. We describe Jacobi ("*brute-force*") method and its cyclic variation ("*cyclic Jacobi*") and we study the performance of both in terms of the dimensionality of the matrix, number of transformations needed and time elapsed. Then we focus our attention on the first eigenvalues as functions of the parameter ρ_{max} (that ideally should be set to infinity) and see that we can identify only some confidence intervals if we want our eigenvalues to be stable. In the last part of the project we make a physical analysis two electrons in a harmonic oscillator well which also interact via repulsive Coulomb potential and highlight the differences between the interacting solutions and the other ones.

- URL to GitHub folder of the code: `https://github.com/DavideSaccardo/Project2`

I. INTRODUCTION

The aim of this project is to study the solutions of time-independent Schrödinger equation in two cases: an electron in a harmonic oscillator potential well that then interacts by Coulomb potential with another electron. As shown in the following section, by scaling the equations properly we obtain similar expressions that can be solved as an eigenvalue problem. As first we analise Jacobi method "*brute-force*" and a variation of that (*cyclic Jacobi*) and discuss about timing and efficiency of each algorithm referred to their output results, arguing on which one could be more suitable for different purposes.
Later we discuss the output values of the solvers in both non-interacting and interacting case. We limit our analysis to the first three eigenvalues for the non interacting case, while for the part with Coulomb interaction we discuss the value corresponding to the ground-state with a certain value of harmonic potential comparing it with the analytical result from literature. We also focus on the dependence of the eigenvalues on the parameter ρ_{max}, that ideally should be set to infinity and should not influence our values. However, the behaviour of the eigenvalues when changing that parameters appears to be everything but trivial.
We finally give a physical interpretation of the lowest eigenstates of our problem and see how both interacting and non-interacting cases behave for different values of harmonic potential. STRESS RHO MAX

II. METHODS AND ALGORITHMS

a. Single particle - non interacting case. Radial time-independent Schrödinger equation for an electron in a potential well is:

$$-\frac{\hbar^2}{2m}\left(\frac{1}{r^2}\frac{d}{dr}r^2\frac{d}{dr}-\frac{l(l+1)}{r^2}\right)R(r)+\mathscr{V}(r)R(r) = E_{nl}R(r) \quad (1)$$

with $\mathcal{V}(r)=\frac{1}{2}kr^2=\frac{1}{2}m\omega^2r^2$ and $r\in[0,\infty)$. In this project we set $l=0$.
If we define $R(r)=:\frac{u(r)}{r}$ and introduce a dimensionless variable $\rho:=\frac{r}{\alpha}$ (α is a lenght constant whose value will be fixed later), equation 1 becomes:

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

with boundary conditions: $u(0)=0,\ u(\infty)=0$.
We now fix the constant α so that

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

and define

$$\lambda:=\frac{2m\alpha^2}{\hbar^2}E_n, \quad (2)$$

our final expression is:

$$-\frac{d^2}{d\rho^2}u(\rho)+\mathscr{V}u(\rho)=\lambda u(\rho) \quad (3)$$
$$\mathscr{V}=\rho^2$$

If we now discretise the equation the possible values of ρ can not be infinite in the algorithm, so we have to set a maximum value ρ_{max} in order to satisfy the boundary condition $u(\rho_{max}) \approx 0$. We will solve the discretised differential equation:

$$-\frac{u_{i+1}}{h^2} + \left(\frac{2}{h^2} + \mathscr{V}_i\right)u_i - \frac{u_{i-1}}{h^2} = \lambda u_i \qquad (4)$$

in the interval $\rho \in [\rho_0 = 0, \rho_{max} = \rho_N]$, where

$$N = \text{number of mesh points}, \quad \mathscr{V}_i = \rho_i^2,$$
$$h = \frac{\rho_N - \rho_0}{N}, \quad u_{i\pm1} = u(\rho_i \pm h).$$

If we neglect the last term ρ_N the equation 4 can be written as:

$$\begin{bmatrix} d_1 & e & 0 & \dots & 0 & 0 & 0 \\ e & d_2 & e & \dots & 0 & 0 & 0 \\ 0 & e & d_3 & \dots & 0 & 0 & 0 \\ 0 & 0 & e & \ddots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & e & d_{N-2} & e \\ 0 & 0 & 0 & \dots & 0 & e & d_{N-1} \end{bmatrix} \mathbf{u} = \lambda\mathbf{u}$$

where we have defined:

$$d_i := \frac{2}{h^2} + \mathscr{V}_i, \quad e := -\frac{1}{h^2},$$
$$\mathbf{u} = (u_o, ..., u_N)^t, \quad i = 1, \dots, N-1.$$

b. Two particles with the same mass - Interacting case If we neglect the Coulomb interaction and define r_1 and r_2 as the positions of the two particles, the 2-body Schrödinger equation is:

$$\left(-\frac{\hbar^2}{2m}\frac{d^2}{dr_1^2} - \frac{\hbar^2}{2m}\frac{d^2}{dr_2^2} + \frac{1}{2}kr_1^2 + \frac{1}{2}kr_2^2\right)u(r_1.r_2) = E^{(2)}u(r_1, r_2) \qquad (5)$$

Let's now introduce the relative position and mass center coordinates:

$$\mathbf{r} := \mathbf{r}_1 - \mathbf{r}_2$$
$$\mathbf{R} := \frac{1}{2}(\mathbf{r}_1 + \mathbf{r}_2)$$
$$\mathbf{p} := \frac{1}{2}(\mathbf{p}_1 - \mathbf{p}_2)$$
$$\mathbf{P} := \mathbf{p}_1 + \mathbf{p}_2$$

Hence the momentum operators become:

$$-\frac{\hbar^2}{2m}\left(\frac{d^2}{dr_1^2} + \frac{d^2}{dr_2^2}\right) = \frac{p_1^2 + p_2^2}{2m}$$
$$= \frac{p^2}{m} + \frac{P^2}{4m}$$
$$= -\frac{\hbar^2}{m}\frac{d^2}{dr^2} - \frac{\hbar^2}{4m}\frac{d^2}{dR^2}$$

and the harmonic oscillator potential is:

$$\mathscr{V}(r_1, r_2) = \frac{k}{2}(r_1^2 + r_2^2) = \frac{1}{4}kr^2 + kR^2$$

With those substitutions, equation 5 becomes:

$$\left[\left(-\frac{\hbar^2}{m}\frac{d^2}{dr^2} + \frac{k}{4}r^2\right) + \left(-\frac{\hbar^2}{4m}\frac{d^2}{dR^2} + kR^2\right)\right]u(r, R) = E^{(2)}u(r, R) = (E_r + E_R)u(r, R)$$

that can be split into:

$$\begin{cases} \left(-\frac{\hbar^2}{m}\frac{d^2}{dr^2} + \frac{1}{4}kr^2\right)\psi(r) = E_r\psi(r) \\ \left(-\frac{\hbar^2}{4m}\frac{d^2}{dR^2} + kR^2\right)\tilde{u}(R) = E_R\tilde{u}(R) \end{cases}$$

and the general solution will be

$$u(r, R) = \psi(r)\tilde{u}(R).$$

When we add the coulombian interaction, it does not affect the mass-centre equation. The potential for relative position equation will thus become:

$$\mathscr{V}(r) = \frac{1}{4}kr^2 + \frac{\beta e^2}{r}, \quad \beta e^2 = 1.44\,\text{eV nm}$$

and so the equation for variable r:

$$\left(-\frac{\hbar^2}{m}\frac{d^2}{dr^2} + \mathscr{V}(r)\right)\psi(r) = E_r\psi(r).$$

If we scale it by defining the same dimensionless variable $\rho := \alpha/r$

$$\left(-\frac{d^2}{d\rho^2} + \omega_r^2\rho^2 + \frac{\beta e^2 m\alpha}{\hbar^2}\frac{1}{\rho}\right)\psi(\rho) = E_r\psi(\rho) \qquad (6)$$

$$\omega_r^2 = \frac{m\,k\,\alpha^4}{4\,\hbar^2} = \frac{m^2\,\omega^2\,\alpha^4}{4\,\hbar^2}$$

and choose α such that $\frac{\beta e^2 m\alpha}{\hbar^2} = 1$, we get to an equation of the same form of equation 3 - with $\mathscr{V} = \omega_r^2\rho^2 + \frac{1}{\rho}$ - that can be solved using the same algorithm.

A. Jacobi eigenvalues solver

Our point becomes now basically solving an eigenvalue problem:

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{x}.$$

where $\mathbf{A} \in \text{Mat}(n, \mathbb{R})$ symmetric ($\mathbf{A}^t = \mathbf{A}$), $\mathbf{x} \in \mathbb{R}^n$, $\lambda \in \mathbb{R}$.
Let us now state some useful theorems:

Theorem II.1. *If* $\mathbf{A} \in Mat(\mathbb{R}, n)$ *is symmetric, then it is similar to a diagonal matrix* $\mathbf{\Delta} = diag(\lambda_1, \dots, \lambda_n)$ *where* $\{\lambda_i\}_{i=1}^n$ *are the eigenvalues of* $\mathbf{A}$. *I.e. there exists an orthogonal matrix* $\mathbf{S} \in Mat(\mathbb{R}, n)$ *such that:*

$$\mathbf{S}^t \mathbf{A} \mathbf{S} = \mathbf{\Delta}.$$

Def II.1. The *Frobenius norm* of a matrix $\mathbf{A} \in \mathrm{Mat}(\mathbb{R}, n)$ is defined as:

$$\|\mathbf{A}\|_F := \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij}^2}$$

Def II.2. We can define the sum of the off-diagonal elements of $\mathbf{A}$ as:

$$\mathrm{off}(\mathbf{A}) := \sqrt{\sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} a_{ij}^2} = \sqrt{\|\mathbf{A}\|_F^2 - \sum_{i=1}^{n} a_{ii}^2}.$$

Theorem II.2. *Similarity (orthogonal) transformations on a matrix* $\mathbf{A}$ *preserve eigenvalues and Frobenius norm.*

Theorem II.3. *Similarity (orthogonal) transformations preserve orthogonality and dot product.*

Proof. Let us define a dot product

$$\begin{aligned} \langle \, , \rangle &: \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}_0^+ \\ \langle \mathbf{v}, \mathbf{w} \rangle &:= \mathbf{v}^t \mathbf{w}, \quad \mathbf{v}, \mathbf{w} \in \mathbb{R}^n \end{aligned}$$

and let $\{\mathbf{b}_i\}_{i=1}^n$ be an orthogonal basis set:

$$\langle \mathbf{b}_i, \mathbf{b}_j \rangle = k_{ij} \delta_{ij} \quad k_{ij} \in \mathbb{R}$$

where δ_{ij} is the Kroenecker delta.
If we transform the basis vectors with an orthogonal matrix $\mathbf{S}$ s.t. $\mathbf{S}^t \mathbf{S} = \mathbb{I}$

$$\widehat{\mathbf{b}}_j := \mathbf{S}\, \mathbf{b}_j \in \mathbb{R}^n, \quad j = 1, \dots, n$$

and consider the dot product between the new vectors, we get:

$$\begin{aligned} \langle \widehat{\mathbf{b}}_i, \widehat{\mathbf{b}}_j \rangle &= (\widehat{\mathbf{b}}_i)^t\, \widehat{\mathbf{b}}_j = (\mathbf{S}\mathbf{b}_i)^t\, \mathbf{S}\mathbf{b}_j \\ &= \mathbf{b}_i^t \mathbf{S}^t\, \mathbf{S}\, \mathbf{b}_j \overbrace{=}^{\mathbf{S}^t\mathbf{S}=\mathbb{I}} \mathbf{b}_i^t\, \mathbf{b}_j \\ &= \langle \mathbf{b}_i, \mathbf{b}_j \rangle = k_{ij}\delta_{ij}. \end{aligned}$$

□

The key of Jacobi's method is to perform a series of rotations on matrix $\mathbf{A}$ to obtain a diagonal matrix and get the eigenvalues as in theorem II.1

$$\mathbf{\Delta} = (\mathbf{S_k^t S_{k-1}^t} \dots \mathbf{S_1^t}) \mathbf{A} (\mathbf{S_1} \dots \mathbf{S_k}) \tag{7}$$

where $\mathbf{S}_i = \mathbf{S}(\theta_\mathbf{i})$ is a generic rotation matrix:

$$\mathbf{S}(\theta) = \begin{bmatrix} 1 & \dots & 0 & 0 & \dots & 0 & 0 & \dots & 0 \\ \vdots & \ddots & \vdots & \vdots & \dots & \vdots & \vdots & \dots & \vdots \\ 0 & \dots & \cos(\theta) & 0 & \dots & 0 & \sin(\theta) & \dots & 0 \\ 0 & \dots & 0 & 1 & \dots & 0 & 0 & \dots & 0 \\ \vdots & \dots & \vdots & \vdots & \ddots & \vdots & \vdots & \dots & \vdots \\ 0 & \dots & 0 & 0 & \dots & 1 & 0 & \dots & 0 \\ 0 & \dots & -\sin(\theta) & 0 & \dots & 0 & \cos(\theta) & \dots & 0 \\ \vdots & \dots & \dots & \dots & \dots & \dots & \dots & \ddots & \dots \\ 0 & \dots & 0 & 0 & \dots & 0 & 0 & \dots & 1 \end{bmatrix}$$

$$\begin{aligned} s_{kk} &= s_{ll} = \cos(\theta) \\ s_{kl} &= -s_{lk} = -\sin(\theta) \\ s_{ii} &= 1 \quad \text{for } i \neq k, l. \end{aligned}$$

Every rotation $\mathbf{B} = \mathbf{S^t A S}$ preserves the original eigenvalues and acts on the elements of $\mathbf{A}$, $\{\mathbf{a_{ij}}\}_\mathbf{ij}$ in the following way:

$$\begin{aligned} b_{ii} &= a_{ii} \quad \text{for } i \neq k, l \\ b_{ik} &= c a_{ik} - s a_{il} \\ b_{kk} &= c^2 a_{kk} - 2cs a_{kl} + s^2 a_{ll} \\ b_{ll} &= c^2 a_{ll} + 2cs a_{kl} + s^2 a_{kk} \\ b_{kl} &= (a_{kk} - a_{ll}) cs - a_{kl}(c^2 - s^2) \end{aligned} \tag{8}$$

where $\{b_{ij}\}_{ij}$ are the elements of transformed matrix $\mathbf{B}$; we have used a short-hand notation for:

$$\begin{aligned} c &:= \cos(\theta) \\ s &:= \sin(\theta). \end{aligned}$$

By rotating the initial matrix $\mathbf{A}$ we want to systematically reduce the value of $\mathrm{off}(\mathbf{A})$. If we consider the rotated matrix $\mathbf{B}$:

$$\mathrm{off}^2(\mathbf{B}) = \|\mathbf{B}\|^2 - \sum_{i=1}^{n} b_{ii}^2 = \mathrm{off}^2(\mathbf{A}) - 2a_{kl}^2.$$

We can see that the algorithm will converge faster if we choose $|a_{kl}|$ to be the largest off-diagonal element; the number of rotations needed is $O(N^2)$. About that, as the convergence will not be perfect, in our code we choose a tolerance `eps` under which we can consider our transformed matrix to be diagonal. We can use this tolerance to limit the value of $\mathrm{off}(\mathbf{B})$, but the code will run faster if we limit the maximum off-diagonal element of the matrix instead. ADD CODE.
To choose the rotation angle θ we have to solve the quadratic equation:

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

obtained from the last line of eq. (8) by defining the following values:

$$\begin{aligned} t &:= \tan(\theta) = \frac{s}{c} \\ \tau &:= \frac{a_{ll} - a_{kk}}{2a_{kl}}. \end{aligned} \tag{9}$$

Hence we get:

$$\begin{aligned} t &= -\tau \pm \sqrt{\tau^2 + 1} \\ c &= \frac{1}{\sqrt{1+t^2}} \\ s &= tc. \end{aligned}$$

Furthermore, in order to limit the loss of numerical precision, we choose t to be the smaller (in absolute value) of the roots of eq. (9) and rationalize the expression for t to avoid subtractions between our values:

$$t = \begin{cases} \frac{-1}{-\tau+\sqrt{\tau^2+1}} & \tau < 0 \\ \frac{1}{\tau+\sqrt{\tau^2+1}} & \tau \geq 0. \end{cases}$$

B. Algorithm analysis

At first, we implement a program with a raw version of Jacobi method, in order to have a little bit of familiarity with the procedure. Then we progressively convert the program to a function in order to make it more compact and test it with unit tests. In particular, we create two functions: `define matrices`, which allows us to create matrix **A** depending on whether we are considering the non-interacting or interacting case through the declaration of a specific flag (`interacting`), and `jacobi`, the heart of the program, that contains the algorithm itself. Moreover, to make our code run faster, we implement a modified version of brute-force Jacobi: cyclic Jacobi. This algorithm avoids searching for the largest off-diagonal matrix element and proceeds by rotating the original matrix cyclically row by row to make the off-diagonal elements null under a selected tolerance `eps`. Selecting the largest off-diagonal elements takes in fact $O(n^2)$flops: even if the convergence will be faster (we need less iterations), it will take much more time than the cyclic row-by-row for the code to run. For both methods we choose the off-diagonal values to be considered null when

$$\max_{i\neq j}\{a_{ij}\} < \texttt{eps} = 10^{-8}.$$

Hence, we modify `jacobi` function to perform also the cyclic algorithm and we insert another flag (`method`), that allows the operator to decide which method (brute force or cyclic) to use. The result is:

```
void jacobi(int n,double** A,double** R,
double* eig_v,double eps,int method,
int& iterations ,double& timeused){

int p,q;
double max_off=get_max(A,p,q,n-1);
//gets the maximum off-diagonal value
and saves its position (p,q) in A
...
if(method==0){ //brute-force Jacobi
 while(max_off > eps && iterations<1e6){
 int p, q;
 max_off = get_max(A, p, q, n-1);
 rotate(A, R, p, q, n-1);
 //rotation of matrix A
 iterations++;
 }
}

if(method==1){ //cyclic Jacobi
 while(max_off> eps && iterations<1e6){
   for(int i=1;i<n-1;i++) {
     for(int j=i+1;j<n-2;j++){
     rotate(A, R, i, j, n-1);
     iterations++;
     }
   }
 }
}
...
for (int i = 0; i < n-1; i++){
 eig_v[i] = A[i][i];
 }

//order eigenvalues
order_eigpairs(n, eig_v, R);
}
```

Furthermore we place an external `for` loop, which is convenient for running the program as many times as needed. This is useful for comparing the two algorithms as regards the run time and the number of iterations performed to reach the requested tolerance. To check our results (eigenvalues) we use an eigenvalue solver function from "Numerical recipe" `tqlit` [?] and we compare the run time of this algorithm with the others for some values of N.
To check the correct behavior of our Jacobi algorithm, as anticipated above, we make our functions undergo two unit tests: one tests the conservation of Frobenius' norm for the non-interacting case, and the other compares the expected values of the eigenvalues[?] with the numerical ones. Each test is performed with both brute-force and cyclic method. Furthermore we develop another code with brute-force Jacobi focused on the coverage of the physics behind Schrödinger equations (3) and (6) for several

h

N	brute-force	cyclic	solver
500	635.592 s	14.386 s	1.052 s

Table I: Here we show the run-time needed to each solver to converge. For the first two we set the parameters $\rho_{max} = 25,\ \rho_{min} = 0$.

ω_r with the right scaling parameters ρ. The tolerance is now set to 10^{-10} and the plots are made with a MatLab code that allows us also to analyze the probability density function (PDF) of the first three eigenstates. It is important to stress that all the probability density functions are normalized to 1

$$1 = \int_0^\infty |\psi(x)|^2\, dx$$

In order to to that with our data points we use the MatLab function `trapz` which approximates the integral using the trapezoidal rule.

III. OUR RESULTS

A. Numerical results

As first we compare the run time and number of iterations (rotations) needed for both *brute-force* and *cyclic* Jacobi method to get the wished tolerance in the non-interacting case `eps`; we present our results in figure 1. We see that even if for smaller matrices brute-force Jacobi is much faster, the time starts increasing significantly for matrix size over 200×200, while the time needed for the cyclic algorithm is almost linear. Concerning the iterations, brute-force Jacobi converges after $O(N^2)$ rotations as expected, while the order of magnitude for cyclic Jacobi is $\sim 10^6$ iterations. That is because in the code we request the program to stop running after 10^6 rotations when `max(A) < eps` as can be seen above in the `jacobi` function. For example, for non-interacting case, if we choose N to be 200 and our tolerance `eps` $= 10^{-8}$, the number of iterations that each method requires to converge is:

$$n_{brute-force} = 50607, \quad n_{cyclic} = 1003912.$$

In addition we compare for the value $N = 500$ the run time of our codes with the `tqli` eigenvalue solver from [ref:Numerical recipes]. The results are shown in Tab.(I). Clearly, Jacobi method is not the most efficient to implement when compared to other eigenvalues solver (e.g. Lanczos').

If we move our analysis to the output values of the algorithms, we see that for the non-interacting case brute-force Jacobi algorithm gives results compatible with the expected ones (eq.(2)):

$$\lambda_1 = 3.00,\ \lambda_2 = 7.00,\ \lambda_3 = 11.00$$

with a precision of over 1 % even for relatively small matrices ($N = 200$) (Fig. (IIa)). On the other hand, cyclic Jacobi converges to the correct eigenvalues for bigger values of N (Fig. (IIb)). However, as cyclic Jacobi has the advantage of requiring much less time than brute-force, it performs much better for big-size matrices. Regarding the interacting case, we expect the first eigenvalue to be $\lambda_1 = 1.25$ according to [2]. For what it concerns brute-force Jacobi, our reasoning is the same and we can state that it converges to the right value from below when n increases. Instead in this case, the cyclic Jacobi shows a particular behavior: it approaches from above the right value, then it keeps it for some n (or at least we suppose), but soon it loses the convergence. We decide to inspect the dependence of the eigenvalues with respect to n for $\rho = 10, 25, 50$. We present the result in Fig. (2).

We see that the value of λ_1 keeps stable even for higher n if ρ_{max} is big enough. For $n > 500, 580, 630$ respectively however, the value starts fluctuating and becomes less accurate. We cannot justify properly this behavior, but we suppose that it could be due to the fact that there is a limit for the step-size h beyond which we incur in loss of numerical precision.

Moreover, in the non-interactive case, if we see the number of mesh-points N each algorithm needs for the eigenvalues to have four significant decimal digits, we get the values in Tab. (III). This calculus is performed with the following code:

```
if(interacting==0){
double first,second,third;
first = round(eig_v[0]*1e4)/1e4;
second = round(eig_v[1]*1e4)/1e4;
third = round(eig_v[2]*1e4)/1e4;

if(round(first) == 3){
 if(round(second) == 7){
  if(round(third) == 11){
   p_p = n;
   cout <<first <<endl;
   cout <<second <<endl;
   cout <<third <<endl;
   cout <<p_p <<endl;
   }
  }
 }
}
```

This confirms that brute-force Jacobi method has a faster convergence than cyclic Jacobi. Obviously if

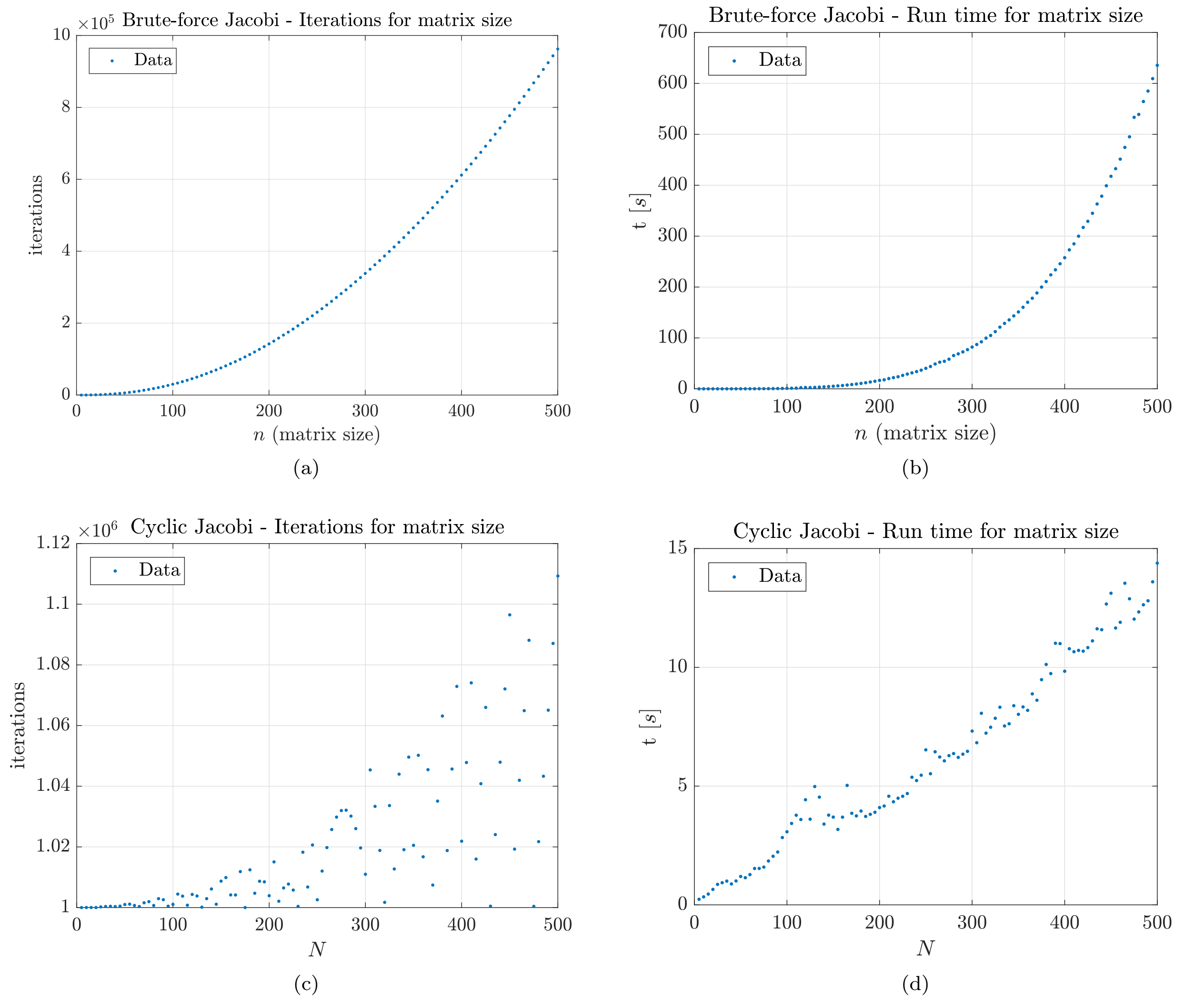

Figure 1: In the plots are compared run-time and iteration depending on mesh-points (matrix size) for both *brute-force* and *cyclic* Jacobi method in the non-interacting case. We set the parameters $\rho_{max} = 25, \rho_{min} = 0$. We see clearly the quadratic dependence of the number of rotations for *brute-force Jacobi* (a), while *cyclic Jacobi* is less stable and shows more fluctuations for both iterations and run-time.

we want to get more precise eigenvalues, we have to increase the number of mesh-points.

As we need to set a finite value of ρ_{max}, it is useful to check the dependence of the eigenvalues on this parameter, while from the above analysis we used a the fixed value $\rho_{max} = 25$. Hence, regarding non-interactive case, we decide to plot the value of the first eigenvalue on $\rho_{max} \in [0, 500]$ for $n = 200$ and $n = 500$ for both the algorithms.

Theoretically, we expect the results to be independent on the choice of ρ. However, as we can see in Fig. (**??**), this isn't our case.

Regarding brute-force Jacobi, for $n = 200$ the value of the eigenvalue is constant until $\rho_{max} = 40$, then there is a diminishing of ~ 0.4 followed by a fast growth. Instead, for $n = 500$, we note that the region of λ_1's constance expands arriving to $\rho \sim 80$.

For what concerns cyclic Jacobi, in the case of $n = 200$, we see a particular pattern, which can be separated into two zones: until $\rho_{max} = 135$ we can distinguish a linear proportional region, then the function assumes a quadratic-like behavior. On the other hand, for $n = 500$, the same pattern appears zoomed throughout x-axis with an increasing of the linear proportional zone. In both cases we see that the most precise eigenvalue is got for $n = \rho_{max}$, but there is a zone, which enlarges with increasing n at

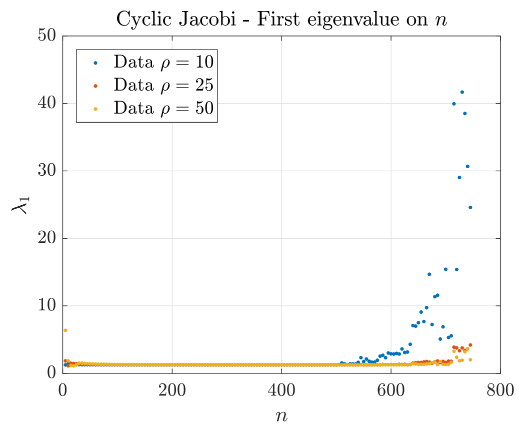

Figure 2

N	200	500	600
Non-interacting case			
λ_1	2.99511	2.99922	2.99946
λ_2	6.9755	6.99609	6.99729
λ_3	10.9401	10.9905	10.9934
Interacting case			
λ_1	1.2497	1.24995	1.24997

(a) brute-force Jacobi

N	200	500	600
Non-interactive case			
λ_1	3.289	3.11439	3.09993
λ_2	7.4095	7.16725	7.1411
λ_3	11.4783	11.2038	11.1715
Interacting case			
λ_1	1.25995	1.25346	1.29271

(b) cyclic Jacobi

Table II: In these tables are shown the lowest eigenvalues for $N = 200, 300, 500$ for the two analized algorithms (*brute-force Jacobi* (a) and *cyclic Jacobi* (b)) and for the non-interactive and interactive case. From the values, we can deduce that the cyclic Jacobi need more mesh-points to converge appropriately.

	brute-force	cyclic
N	71	201
λ_1	2.9607	3.2875
λ_2	6.8004	7.4075
λ_3	10.5042	11.4762

Table III: In the table are compared the minimum mesh-points needed to get 4 decimal leading digits for the two algorithms and their respective three lowest eigenvalues. To reach the same precision, cyclic Jacobi needs more mesh-points.

the beginning where we have a good precision on the eigenvalue. Therefore we can define for the Brute-force Jacobi a confidence interval $\rho_{max} \in [0, 40]$ for $n = 200$ and $\rho_{max} \in [0, 80]$ for $n = 500$. Still for the cyclic we can define a confidence interval $\rho_{max} \in [0, 10]$, but it's really narrow. Indeed, in this case, the recipe to get good eigenvalues is more complicate: we should consider a high number of mesh points (order of 600 to be able to define a greater confidence interval) and keep a low ρ_{max}. However, from Fig. (2), we know that if we fix a ρ_{max}, there is a superior limit to how much we can increase n. Thus, keeping low ρ_{max} (order of 10), to get enough accurate eigenvalues, the matrix size n should vary nearby 500.

B. Physical interpretation of the eigenstates

We want to study the numerical solutions of Schrödinger eq.(6) for the lowest states as function of varying strengths of ω_r. But first let us consider the radial probability density function PDF $|\psi(\rho)|^2$ of the first three states with $\omega_r = 0.01$. From Fig.(4) it is possible to analyze how many nodes ρ_{node} the eigenfunctions ψ have, i.e. the distance from the center where the probability of finding an electron is null. The number of nodes per eigenstate is related to the number of the possible eigenfunctions, e.g. given the n-eigenstate ψ_n the number of nodes is equal to $n - 1$. Where we consider the ground state labeled by ψ_1. The nodes are due to the form of Schrödinger equation and its relation with wave behavior of quantum systems.

As we increase the value of ω_r in (ref formula), we make the harmonic potential hole narrower. On one hand the two electrons are constraint to be closer (5), on the other hand the the Coulomb potential intensifies. Therefore, we expect the energy λ_n of the system will increase as we make stronger the harmonic potential (IV).

eigenvalue	$\omega_r = 5$	$\omega_r = 1$	$\omega_r = 0.5$	$\omega_r = 0.01$
λ_1	17.449	4.0578	2.2301	0.10577
λ_2	37.071	7.9096	4.1344	0.14151
λ_3	56.852	11.82	6.0741	0.17803
$\sigma_x^2 =$	0.0050994	0.026283	0.053544	2.9834

Table IV: The table shows the first three eigenvalues of scaled Schrödinger equation with different pulsations ω_r. In the last row we can notice the variance increases as ω_r gets bigger

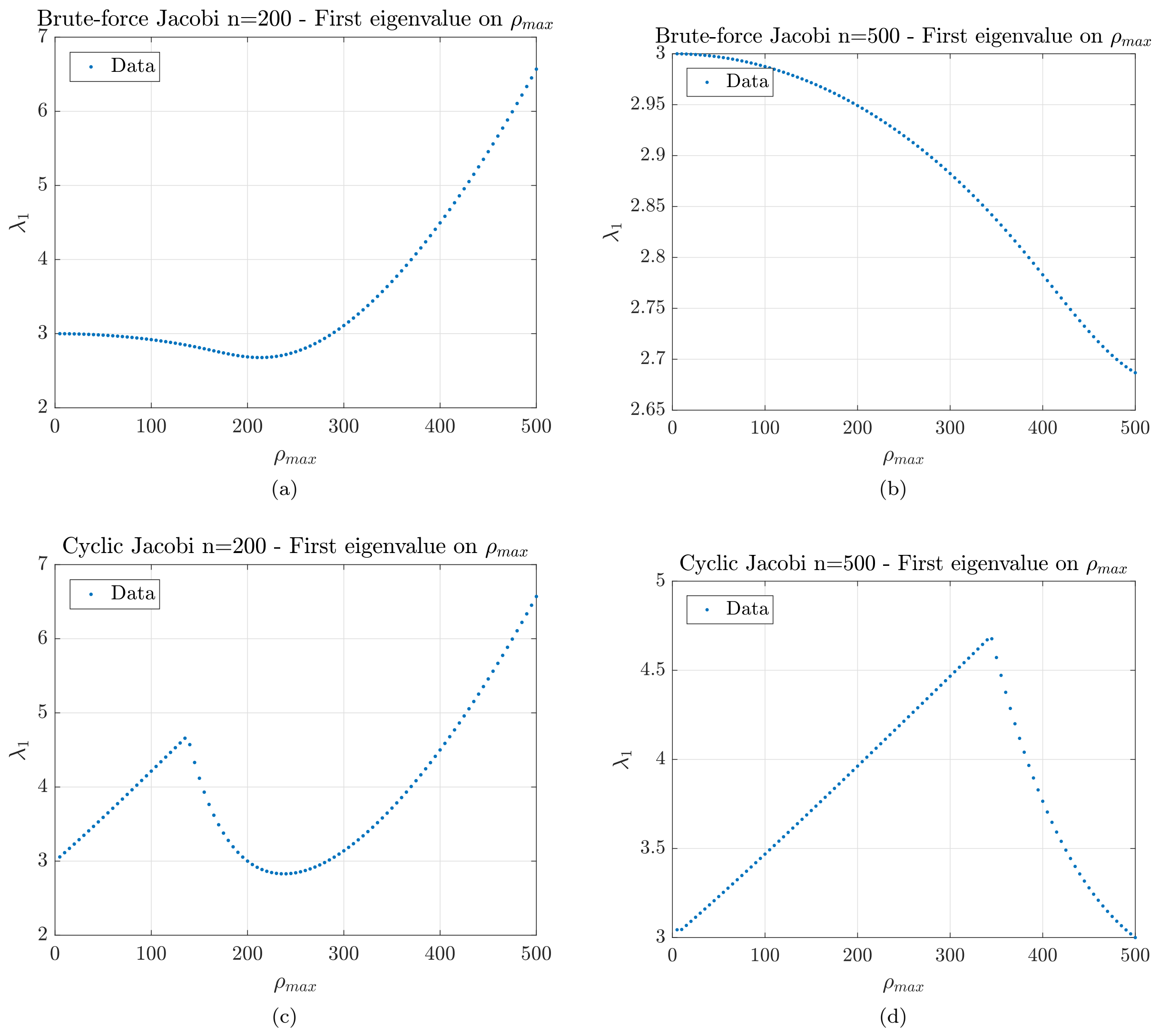

Figure 3

As a consequence the electrons are trapped between the Harmonic potential and the Coulomb repulsion so the uncertainty on their positions

$$\sigma_x = \sqrt{\int_0^\infty (x - \langle x \rangle)^2 \, |\psi_0(x)|^2 \, dx}$$

increases as the ω_r decreases (table IV).

At this point is important to keep in mind the scaling procedure of the two eq. (6) and eq. (3) if we want to compare the interacting case and the non-interacting one. In fact, let us set the value of $\omega_r = 0.01$, 0.5, 1, 5 and keep

$$\alpha_{inter} = \frac{\hbar^2}{m\,\beta\,e^2}$$

for the interacting case, now we need the correspondence scale value for the other eq. (3)

$$\alpha_{osc} = \left(\frac{\hbar^2}{m^2\,\omega^2}\right)^{\frac{1}{4}} \tag{10}$$

We can obtain the denominator of this eq. (10) manipulating the definition of ω_r:

$$m^2\,\omega^2 = \frac{4\,\hbar^2 \omega_r^2}{\alpha^4}$$

Finally using the expression $r = \alpha\,\rho$ with the correspondent α for the two cases, we can compare the probability density functions of the respective ground states Fig. (6). As we can see from the Tab. (V), the mean value of the relative position

$$\langle r \rangle = \int_0^\infty |\psi_0(x)|^2 \, dx$$

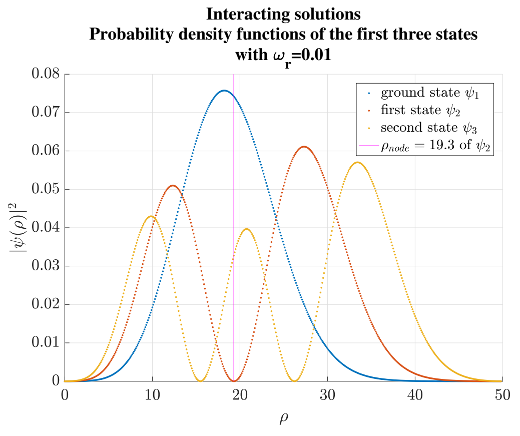

Figure 4: Lowest probability density functions of Schrödinger equation with Coulomb and harmonic potentials. It is also shown the position of the node of ψ_1

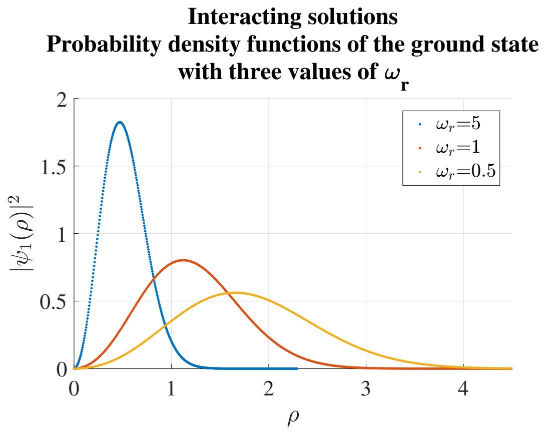

Figure 5: The first three probability density functions with different pulsations. As the harmonic potential diminishes in his strength the functions become flatter

is always bigger in the interacting case than in the non-interacting one. The Coulomb potential makes electrons farther and influences also their uncertainties of the position. Indeed we can see the probability density function of the interacting case is flatter. In addition the difference between the maximum values of the two functions decreases with the value of ω_r.

mean value of x	$\omega_r = 5$	$\omega_r = 1$	$\omega_r = 0.5$	$\omega_r = 0.01$
$\langle r \rangle_{inter}$	0.17274	0.40657	0.5951	6.2039
$\langle r \rangle_{osc}$	0.16989	0.26116	0.36933	2.6116
$\Delta max =$	2.4252	1.125	0.81188	0.12617

Table V: The table shows how the mean value of the probability density functions of the ground state is closer to zero in the non-interacting case. In addition we show the difference between the maximum values reached by the two cases
$\Delta max = |\, max(|\psi_{0inter}|^2) - max(|\psi_{0osc}|^2)\,|$

IV. CONCLUSION

We use two versions of a Jacobi eigenvalue solver (*brute-force* and *cyclic*) to get the first scaled energy eigenvalues λ for both interacting and non-interacting case. By analyzing the algorithms' performances we see that, while brute-force Jacobi works pretty good for relatively small matrices (n), then its run-time starts increasing significantly and cyclic Jacobi proves itself to be more efficient even if it needs more iterations to converge. Considering the output values, we see that for the same matrix-size n we get more accurate values if we use brute-force Jacobi, which converges to the right value from below. Nevertheless the time for it to perform becomes significantly long for $n > 300$. Moreover we show that the precision of brute-force Jacobi increases for larger n for both non-interacting and interacting case. Instead, when we evaluate the interacting case with cycling Jacobi, we demonstrate that it fails. Therefore we analyze the behaviour of the first eigenvalue as function of n for three different values of ρ_{max} $(10, 25, 50)$ and see that after a certain value they become less accurate and diverge. We cannot explain properly this behaviour but we suppose we could blame loss of numerical precision for that. Another point we want to stress is the dependence of the eigenvalue from the value ρ_{max}, that ideally should be set to infinity. We used the non-interacting case to prove that the independence isn't true. However we manage to find a proper confidence interval for brute force Jacobi which is $\rho_{max} \in [0, 40]$ for $n = 200$ and $\rho_{max} \in [0, 80]$ for $n = 500$.

For brute force Jacobi, to find appropriate eigenvalues one should increase as much as possible n, but keep a value of ρ_{max} not too high (a choice made by comparison with the plots we have made should be enough).

With regards to cyclic Jacobi, the search for a proper confidence interval isn't trivial as for the other algorithm. We have to match the dependence of λ_i on n and on ρ_{max} to get the right values. Even-

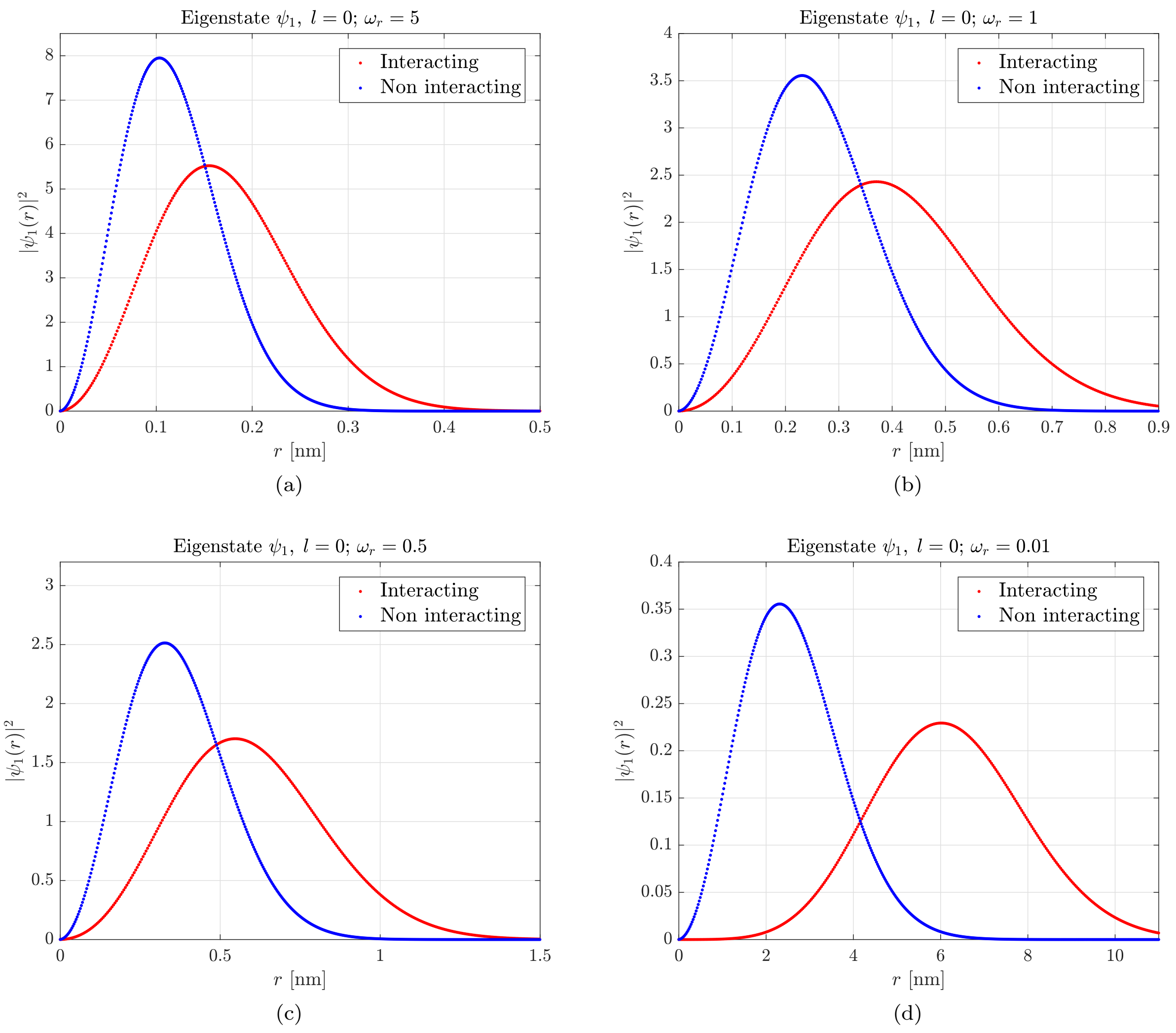

Figure 6: Each plot shows the radial probability density functions of the ground state in the interacting case and in the non-interacting one. Different values of ω_r make the the interacting functions flatter and the electrons farther than in the non-interacting case. In addition the difference between the maximum values of the compared functions decrease a as ω_r gets smaller. LABEL

tually, to get a good precision with this method, one should keep ρ_{max} low (~ 10) and set the matrix size n with a value in a really narrow neighborhood of 500. Even if the precision will always be worse than brute-force Jacobi, cyclic Jacobi allows us to reduce abruptly the run time of our solver.
The solutions to the time-independent Schrödinger's equation of the two electrons interacting shows as the harmonic potential gets bigger the energy of the system increases. In fact making electrons closer yield the Coulomb repulsion to increase and this cause enlargement of energy. So the two particles move faster and the uncertainty of momentum is bigger but they are trapped between the two potentials and σ_x decreases.
From the analysis of the interacting and non-interacting cases we conclude that the Coulomb potential makes the mean value of the relative position farther from the center as ω_r decreases. However, as ω_r increases its value so does δ_{max}.

[1] M. H. Jensen, *Computational physics - Lecture notes Fall 2015*, University of Oslo - Department of Physics, 2015.

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