Project 2 - FYS3150

Abstract

This project is making a eigenvalue and eigenvector solver using the Jacobi method. This can thereby easily solve an matrix equation of the Schroedingers equation. Looking at both non-interacting and interacting case, the non-interacting also checking up on the time for the solver to run.

1. Introduction

In this project the the Schroedinger equation for is solved by using Jacobi's method for eigenvalues an eigenvectors. Measuring time for the Jacobi method is reported and wavefunctions plottet. The Schroedinger equation is solved for a non-interacting case and an interacting case, where the non-interacting case looking at the three lowest energy states with the eigenvalues $\lambda_0, \lambda_1, \lambda_2$. The interacting case is using the change in frequency, and therby the potential to look at the wavefunction.

2. Theory

The equation of interrest in this project, is the radial part of Schroedinger's equation for one electron,

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

the harmonic oscillator potential $V(r) = \left(\frac{1}{2}m\omega^2 r^2\right)$ and E is the energy.

The relation between energies and frequencies are

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

where n is the principal quantum number and l is the orbital momentum of the electron.

Substituting $R(r) = \left(\frac{1}{r}\right)u(r)$ gives

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

Since its the radial part that is of interrest, $r \in [0, \infty)$, that means that the boundry conditions read

$u(0) = 0$ and $u(\infty) = 0$.

This project is only interrested in $l = 0$, and after changing to dimensionless variables, we can rewrite Schroedingers equation as

$$-\frac{d^2}{d\rho^2}u(\rho) + \rho^2 u(\rho)? = \lambda u(\rho) \tag{3}$$

This is the equation to solve for, and for $l = 0$, the first three eigenvalues is $\lambda_0 = 3, \lambda_1 = 7, \lambda_2 = 11$

Solving the equation numerically using the standard expression for the second derivative

$$u'' = \frac{u(\rho + h) - 2u(\rho) + u(\rho - h)}{h^2} + O(h^2) \tag{4}$$

h is the step size and are found by

$h = \frac{\rho_N - \rho_0}{N}$ where $\rho_0 = 0$ and $\rho_N = \rho_{\max}$

Now we can write the Schroeedinger equation as

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

For the diagonal matrix element

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

and the non-diagonal matrix element

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

with all non-diagonal matrix elements equal and an constant, the equation now reads

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

this can be transformed into an matrix eigenvalue problem

$$\begin{bmatrix} d_0 & e_0 & 0 & \cdots & \cdots & \cdots & 0 \\ e_1 & d_1 & e_1 & 0 & \cdots & \cdots & \vdots \\ 0 & e_2 & d_2 & e_2 & 0 & \cdots & \vdots \\ 0 & 0 & \ddots & \ddots & \ddots & 0 & \vdots \\ \vdots & \cdots & 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \cdots & \cdots & 0 & e_{N-1} & d_{N-1} & e_{N-1} \\ 0 & \cdots & \cdots & \cdots & 0 & e_N & d_N \end{bmatrix} \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ \vdots \\ \vdots \\ u_{N-1} \\ u_N \end{bmatrix} = \lambda \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ \vdots \\ \vdots \\ u_{N-1} \\ u_N \end{bmatrix} \quad (6)$$

Inserting the d and e values, can skip thr rows with the enpoints because of the boundary contitions, obtaining

$$\begin{bmatrix} \frac{2}{h^2}+V_1 & -\frac{1}{h^2} & 0 & \cdots & \cdots & \cdots & 0 \\ -\frac{1}{h^2} & \frac{2}{h^2}+V_2 & -\frac{1}{h^2} & 0 & \cdots & \cdots & \vdots \\ 0 & -\frac{1}{h^2} & \frac{2}{h^2}+V_3 & -\frac{1}{h^2} & 0 & \cdots & \vdots \\ 0 & 0 & \ddots & \ddots & \ddots & 0 & \vdots \\ \vdots & \cdots & 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \cdots & \cdots & 0 & -\frac{1}{h^2} & \ddots & -\frac{1}{h^2} \\ 0 & \cdots & \cdots & \cdots & 0 & -\frac{1}{h^2} & \frac{2}{h^2}+V_{N-1} \end{bmatrix} \quad (7)$$

3. Implementation

For solving the eigenvalue problem, the Jacobi method is used. This implies that the matrices used are orthogonal and that an unitary transformation is preserving both the dot product ant the orthoganality.

If the basis is orthogonal,

$v_j^T v_i = \delta_{ij}$

This means that the identity matrix should be the output of this operation

$$S = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix}$$

$$v = S^T S = \begin{bmatrix} c & s \\ -s & c \end{bmatrix}\begin{bmatrix} c & -s \\ s & c \end{bmatrix} = \begin{bmatrix} c^2+s^2 & 0 \\ 0 & c^2+s^2 \end{bmatrix}$$
$$= \begin{bmatrix} \cos^2\theta+\sin^2\theta & 0 \\ 0 & \cos^2\theta+\sin^2\theta \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

To show the unitary preservation, the vectors $v_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ and $v_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ are used. Its easy to see that $v_1 \cdot v_2 = 0$, that means orthogonal and,

$$w_i = Sv_1 = \begin{bmatrix} c \\ -s \end{bmatrix}$$
$$w_i^T w_i = c^2 + s^2 = \cos^2\theta + \sin^2\theta = 1$$

And both dot product and ortogonality is preserved.

The Jacobi method uses a transformation and rotation of the matrix such that the eigenvectors and the eigenvalues are preserved, the following relations are used

$$(a_{kk} - a_{ll})cs + a_{kl}(c^2 - s^2) = 0 \qquad (8)$$

$$\cot 2\theta = \tau = \frac{a_{ll} - a_{kk}}{2a_{kl}} \qquad (9)$$

$$t^2 + 2\tau t - 1 = 0 \qquad (10)$$

$$t = -\tau \pm \sqrt{1+\tau^2} \qquad (11)$$

$$c = \frac{1}{\sqrt{1+t^2}} \qquad (12)$$
$$s = tc \qquad (13)$$

The potential is found for the non-interaction case by $V_i = \rho^2$ and the interacting case by $V_i = \omega_r^2\rho^2 + \frac{1}{\rho}$

The program
`interacting.py`

`non-interacting.py`
interacting.py is used to solve the Jacobi method and producing table for the 3 lowest eigenvalues and
timing, non-interacting.py is used for the interacting case, there are also a file

`test_interacting_noninteracting.py`
wich tests some cases in the interacting/non-interacting modules using pytest.

4. Result

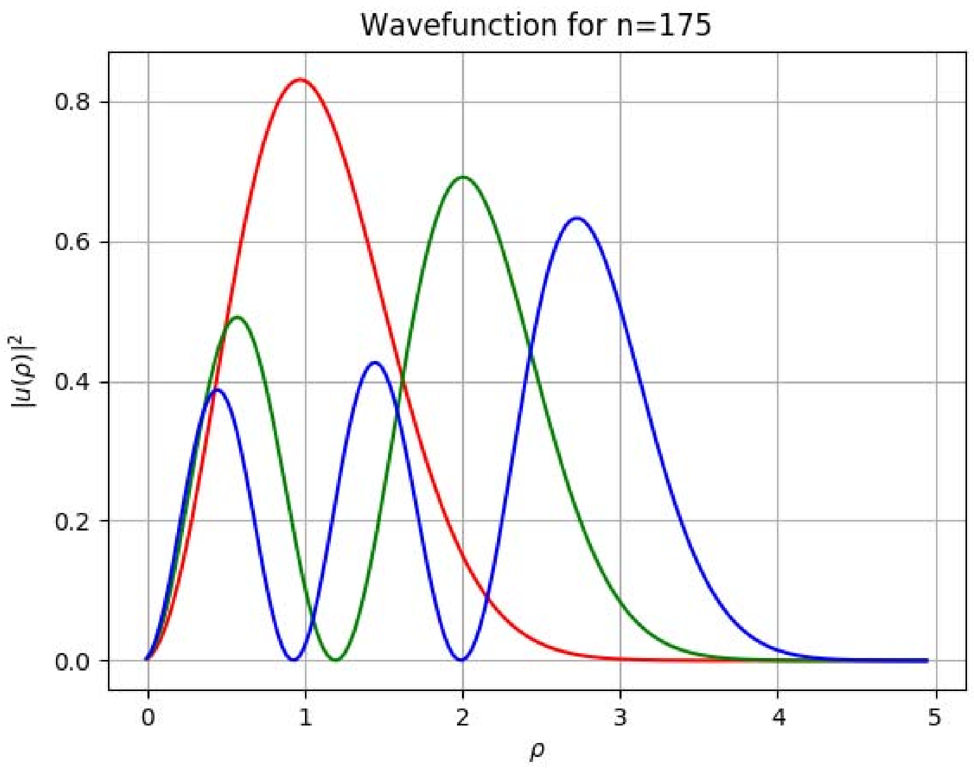
Results from the outprint for values ranging between $n = [50, 175]$ is printet from

commandline into file
`report1.txt`

Table 1 - Eigenvalues, time and iterations for different values of n

n	Eigenvalues	Time [s]	Iterations
50	$[2.99687148, 6.98434175, 10.96193472]$	2.34377440738	4006.0
75	$[2.99861042, 6.99305099, 10.98322511]$	10.5088894709	9292.0
100	$[2.99921854, 6.99609399, 10.99065702]$	33.7149960909	16780.0
125	$[2.99949992, 6.99750154, 10.9940935]$	72.1995331956	26319.0
150	$[2.99965274, 6.9982659, 10.99595931]$	148.953885289	38118.0
175	$[2.99974488, 6.9987267, 10.99708404]$	271.82826845	52081.0

The computation time was very slow, so max n set to 175.
Figure 1 shows the plot of the wavefunction for $n = 175$.



Figur 1 - Wavefunction for the non-interacting case, $n = 175$

For the interacting case, the code from the non-interacting case is reused, except the value of $V_i = V_i = \omega_r^2 \rho^2 + \frac{1}{\rho}$

The $\rho_{\max}$ was changed in order to get the whole wavefunction

Table 2 - $\rho_{\max}$ values for different ω_r

ω_r	0.01	0.5	1	5
$\rho_{\max}$	50	7	5	2

Results from the outprint for values ranging between $n = 175$ is printet from

commandline into file
```
report2.txt
```

Table 3 - Eigenvalue for different frequency in potential

Frequency ω_r	Eigenvalue λ_0
0.01	0.10577306667
0.50	2.22999238471
1	4.05760969451
5	17.4476240377

Figure 2-5 shows the plottet wavefunctions

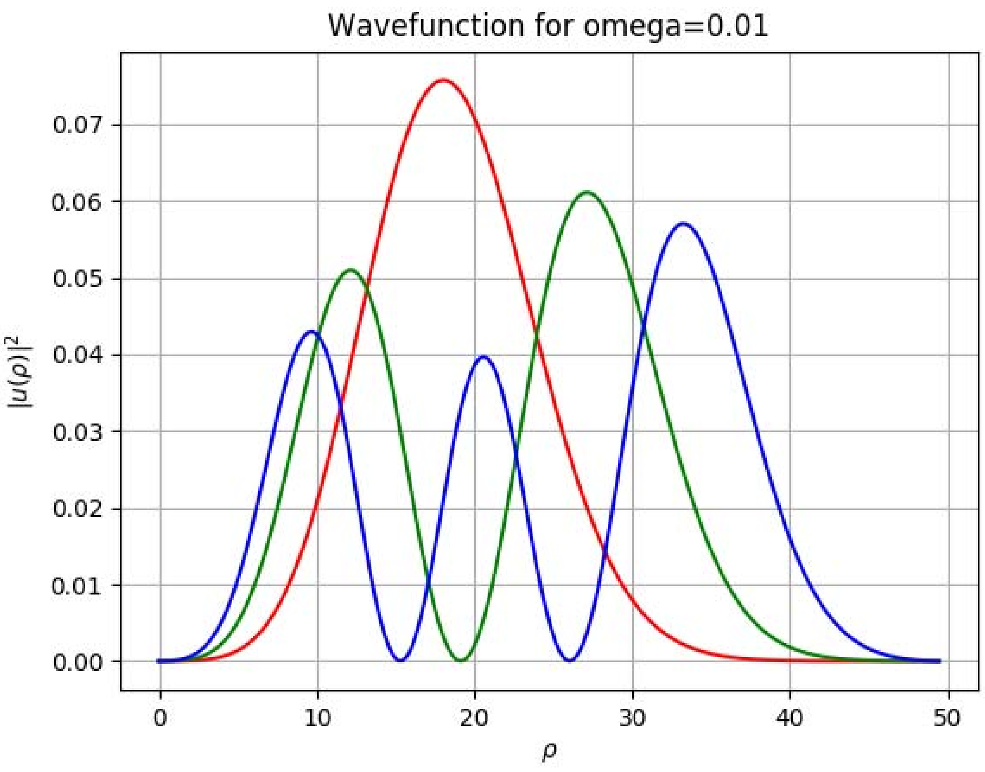


Figure 2 - Wavefunction of the interacting case for $\omega_r = 0.01$

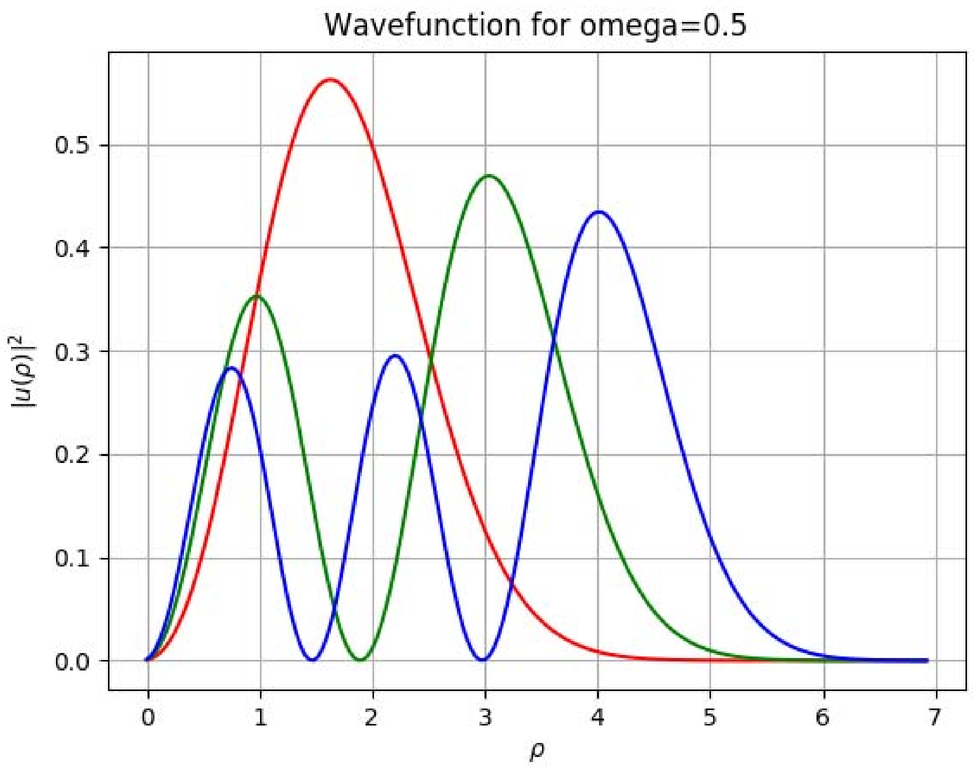


Figure 3 - Wavefunction of the interacting case for $\omega_r = 0.5$

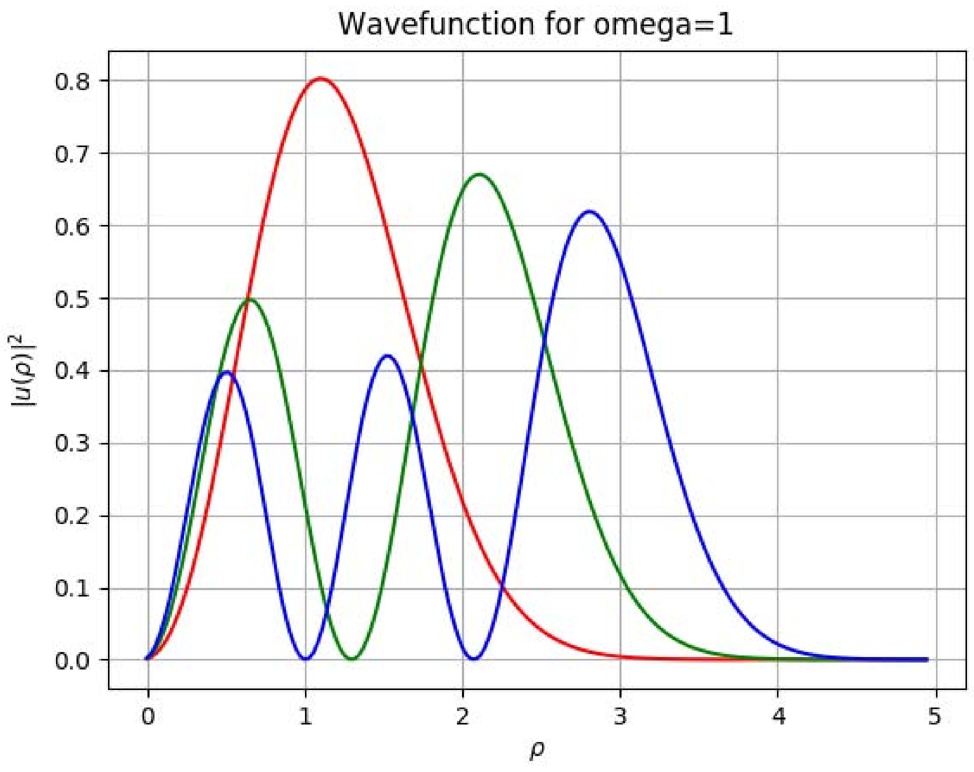


Figure 4 - Wavefunction of the interacting case for $\omega_r = 1$

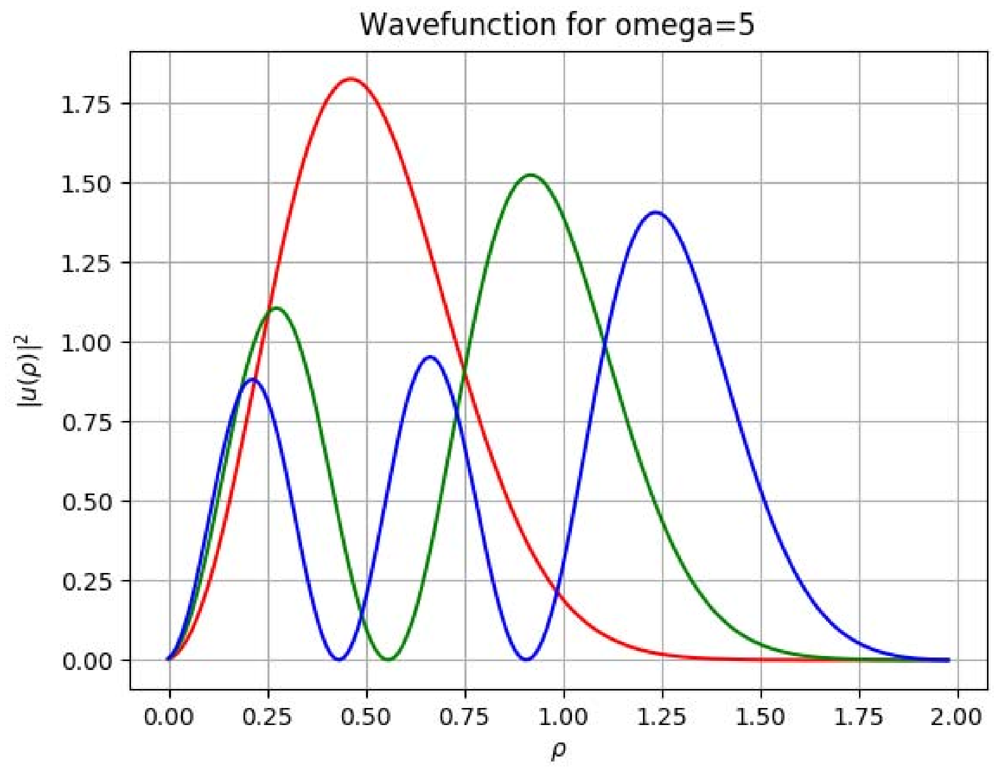


Figure 5 - Wavefunction of the interacting case for $\omega_r = 5$

5. Conclusion

For the task of finding the eigenvalues using the Jacobi method, the Jacobi method easily found the correct egenvalues, but the amount of time it took was high. There probably could be a lot of finetuning to the algorithm and probably some other python libraries solving this task a lot faster. The Jacobi method also uses a lot of time looking through elements that are zero, so implementing a method for discarding this values for computations would probably save a lot of time.

6. References

[1], Project 2,1999-2017, "Computational Physics I

FYS3150/FYS4150":"http://www.uio.no/studier/emner/matnat/fys/FYS3150/indexeng html". Released under CC Attribution-NonCommercial 4.0
[2], https://en.wikipedia.org/wiki/Jacobi_eigenvalue_algorithm