

3D QUANTUM MECHANICAL OSCILLATOR WITH INTERACTION TERM SOLVED WITH JACOBI'S METHOD FYS 4150: PROJECT 2

ASK J. MARKESTAD, THORBJØERN V. LARSEN

Abstract

Jacobis method

INTRODUCTION

THEORY AND ALGORITHMS

$$\begin{pmatrix} 2 & -1 & 0 & \dots & \dots & 0 \\ -1 & 2 & -1 & 0 & \dots & 0 \\ 0 & -1 & 2 & -1 & 0 & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & 0 & -1 & 2 & -1 \\ 0 & 0 & \dots & 0 & -1 & 2 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ \dots \\ \dots \\ u_n \end{pmatrix} = \begin{pmatrix} f_1 h^2 \\ f_2 h^2 \\ f_3 h^2 \\ \dots \\ \dots \\ f_n h^2 \end{pmatrix} \quad (1)$$

A. Algorithms

Source code and accompanying codes can be found at the git hub address:

<https://github.com/ajmarkestad/Fys4150/tree/master/Project2>

```
//general forward algorithm
for (int i=1; i<=n; i++)
{
    b[i]=b[i]-c[i-1]*b[i-1]/b[i-1];
    f[i]=f[i]-c[i-1]*f[i-1]/b[i-1];
}
```

Since our matrix is a tridiagonal symmetric matrix, Jacobi's method is not the best algorithm to find eigenvalues and eigenvectors. While it also works on a general dense matrix, we also wanted to test versus a standard library method that is already implemented in armadillo. This is the divide-and-conquer method which should scale in the general case of a dense matrix as $4n^3$ flops[2]. It is rather easy to implement with the following lines

```
mat B = A; //copy for the ARMADILLO solver
vec eigval;
mat eigvec;
eig_sym(eigval, eigvec, B);
```

It is important to note that both methods are not optimised for the tridiagonal case and are ment for dense symmetric matrices. While this might seem like a strange idea, we can use this to see how Jacobi's method performs compared to other state of the art algorithms.

B. Unit-tests

There exist certain mathematical properties that could be exploited to make sure the program and the algorithms run correctly. Since the transformations that occur in the Jacobi's method are either orthogonal or unitary, one can see that the inner product of a given matrix will stay invariant. Under the orthogonal transformation U one has

$$v^T v = v^T U^T U v = (Uv)^T (Uv) = w^T w \quad (2)$$

while for a unitary transformation W and general complex v

$$v^\dagger v = v^\dagger W^\dagger W v = (Wv)^\dagger (Wv) = w^\dagger w \quad (3)$$

We see that under these transformation the inner product is conserved. We can also check whether orthogonality also is conserved. In the initial basis $\{u_i\}$ the orthogonality relation is $u_j^\dagger u_i = \delta_{ij}$. We transform as earlier

$$\delta_{ij} = u_j^\dagger u_i = u_j^\dagger W^\dagger W u_i = w_j^\dagger w_i \quad (4)$$

which we see has the same property in the transformed system. We can use these identities to construct tests after the Jacobis method calculation, to ensure that machine error in representing numbers not will perturb the results after running a high number of iterations. We implement this as a unittest for a random, symmetric (10x10) matrix as an initial test of the algorithm and in the end of a run with the full blown set of eigenvectors.

```
//Orthogonality test
for i=0 : n
    for j=0 : n
        innerproduct = dot(vector(i),vector(j))
        if ((i==j) && (abs(abs(innerproduct)-1) > pow(10,-12))) result = "bad";
        if ((i!=j) && (abs(innerproduct)>pow(10,-12))) result = "bad";
    }
}
```

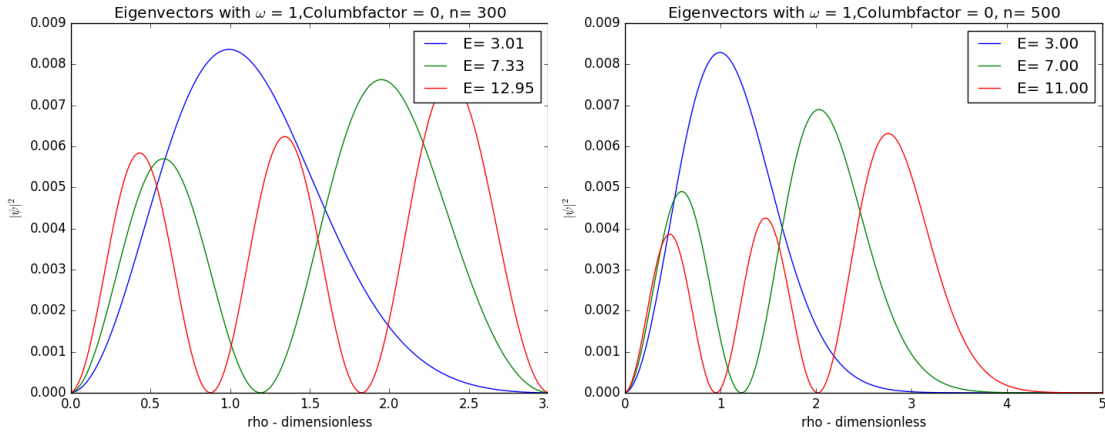
In this setting conserved is a test that the difference is smaller than a given tolerance (in our case $\epsilon = 10^{-12}$). There is also another set of tests that are useful while constructing the programs. These tests rely on simple constructed problems that are solved analytically and compared to the solutions the algorithms give. As we want to find eigenvalues and vectors, we can construct the matrix

$$\begin{pmatrix} 3 & \sqrt{2} \\ 0 & -1 \end{pmatrix}, \quad \lambda_1 = -1, \quad \lambda_2 = 3, \quad v_1 = \begin{pmatrix} -\frac{1}{2\sqrt{2}} \\ 1 \end{pmatrix}, \quad v_2 = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad (5)$$

As a solution the program gives -1 and 3 which corresponds perfectly with the analytic solutions. To make sure the subroutines also runs, a test of the functions that finds the maximum off-diagonal element is also included in the startup tests. We initialise a 3x3 empty matrix \hat{A} with the element $A_{1,2} = 1$ to make sure it finds this element, with the correct column and row. For a verification that negative elements function correctly, we also test that on a similar zero-matrix with the element $A_{2,1} = -20, A_{1,2} = 1$ the value -20 is picked out.

RESULTS

The choice of ρ_{max} and the grid size is slightly tricky as we want to make sure we extend the radius long enough to make sure the wavefunction has time to go sufficiently to zero while keeping the spatial resolution high enough not to affect the eigenvalues. From the differential equation with the interaction case it is obvious that for low ω there might be some problems in extending ρ_{max} as the system naturally broadens from the weak oscillator force. In the first test we wanted to make sure we got the same results as the analytical solutions, and were interested in how far we needed to push the grid-size to get 4 digits accuracy.



(a) Harmonic oscillator with grid size = 300, (b) Harmonic oscillator with grid size = 500, $\rho_{max} = 3$, $\omega = 1$ and Columb Factor = 0. $\rho_{max} = 5$, $\omega = 1$ and Columb Factor = 0.

Table 1: default

Eigenvalue	λ_1	λ_2	λ_3
$\omega = 0.01$	0.599	0.968	1.347
$\omega = 0.5$	3.001	5.710	8.464
$\omega = 1$	4.058	7.910	11.82
$\omega = 5$	8.323	17.03	25.85

From the article (INSERT REFERENCE) we know that the first analytic eigenvalues for $\omega = 1$ and Columb Factor= 0 should be 3, 7 and 11. From an arbitrary chosen simulation of 300x300 in figure 1a it is obvious that $\rho_{max} = 3$ gives a too short size for the higher eigenvectors as these in general are pushed further out. Nevertheless we see that for the lowest eigenvalue we get an almost 3 digits correct. We could try to push this up to a higher ρ_{max} but also keeping the relative size of the grids roughly the same. After going up to $n=500$ and $\rho_{max} = 5$ all the first 4 digits are correct as shown in figure 1b. We suspect that for higher eigenvalues that stretch farther out there is a need for a higher maximal size of the grid together with higher grid sizes for a given precision, and a visual inspection for a general choice of ω and Columb Factor to see that all wavefunctions to sufficiently close to zero is a fair method of choosing ρ_{max} and then let the grid size be roughly $\rho_{max} * 100$.

In the general case with a interaction term we want to see how the strength of the oscillator affects the distribution. We used the recommended values of $\omega = (0.01, 0.5, 1, 5)$. We notices that for a high omega the distribution is pushed towards lower ρ values and for lower values one runs into problems as the distribution is very wide. After choosing appropriate ρ and n the eigenvalues for the energy is seen in table 1. As the strength of the harmonic oscillator potential increases, there is an strong increase in the energy-eigenvalues and from the plots of the distributions the electrons are restricted much closer to each other, which is no surprise.

C. Scaling and flops

To check the performance and scaling with respect to the grid size, an brute force test that runs the algorithm for different n from 50 to 1000 was performed. Then to find the scaling one approximates the total number of flops with time spend expresses this as a relation which is expected to become more precise for large n

$$T \propto n^a \quad (6)$$

$$\log_{10}(T) \approx a \cdot \log_{10}(n) \quad (7)$$

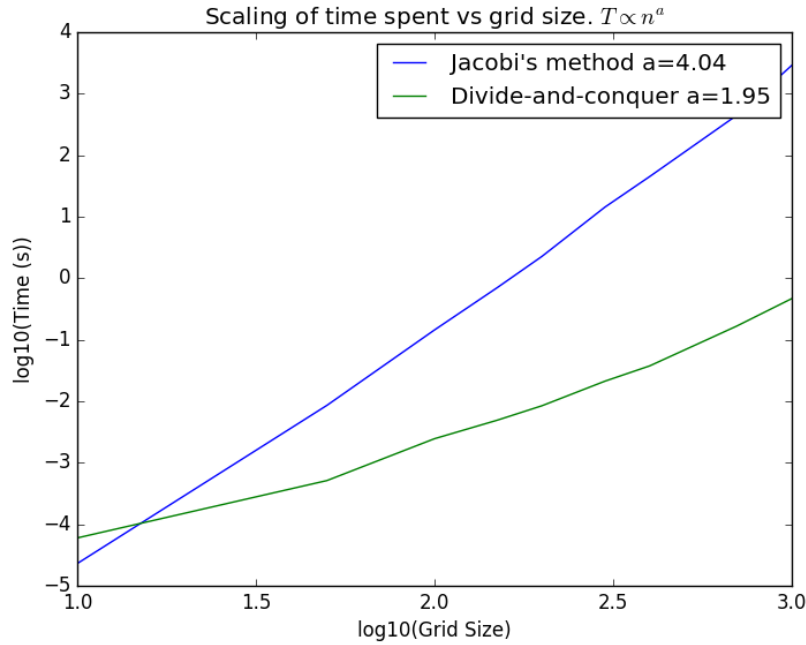


Figure 2: Time spend on finding eigenvalues/vectors for Jacobi's method and the Divide-and-conquer method. This data is from a run on a Macbook Pro 13'

In figure 2 one sees that there is a relation between the grid size and the number of operations. After doing a linear regression on the data with equation 7 one finds that

$$T \approx n^{4.0} \quad (8)$$

This means that the Jacobi's method scales poorly with increasing grid size. Note that these values are expected to be hardware dependent as the number of operations per second varies with the computer, but the values are nevertheless guiding in evaluating the algorithm. The built in method of divide-and-conquer scales much better and is already faster at $n = 50$. The memory requirements for both methods would also be interesting to study how Jacobi's method does with respect to this dimension as well. From the results of this scaling test we used the armadillo method for $n > 200$ to increase the speed of our calculations as the Jacobi's method simply was too slow in this regime (1 hour for 1000x1000 compared to 0.7 s is a no-brainer).

CONCLUSION

Jacobi's method is an easy to implement method for finding eigenvalues and eigenvectors for linear algebra problems with symmetric, square matrices up to 1000x1000. Since the scaling behaviour is poor ($time \approx n^4$) for tridiagonal problems, is it not useful as an algorithm for matrices larger than this size.

REFERENCES

- [1] Morten Hjort-Jensen *Computational Physics Lecture Notes Fall 2015* Department of Physics, University of Oslo 2015 <https://github.com/CompPhysics/ComputationalPhysics/blob/master/doc/Lectures/lectures2015.pdf>
- [2] Wikipedia: Divide-and-conquer method https://en.wikipedia.org/wiki/Divide-and-conquer_eigenvalue_algorithm