3D Quantum Mechanical oscillator with interaction term solved with Jacobi's method
Fys 4150: Project 2

Ask J. Markestad, Thorbjoern V. Larsen

Abstract

Jacobis method

Introduction

Theory and Algorithms

$$\begin{pmatrix} 2 & -1 & 0 & ... & ... & 0 \\ -1 & 2 & -1 & 0 & ... & 0 \\ 0 & -1 & 2 & -1 & 0 & ... \\ ... & ... & ... & ... & ... & ... \\ 0 & ... & 0 & -1 & 2 & -1 \\ 0 & 0 & ... & 0 & -1 & 2 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ ... \\ ... \\ u_n \end{pmatrix} = \begin{pmatrix} f_1 h^2 \\ f_2 h^2 \\ f_3 h^2 \\ ... \\ ... \\ f_n h^2 \end{pmatrix} \tag{1}$$

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

$$v^T v = v^T U^T U v = (Uv)^T (Uv) = w^T w \tag{2}$$

while for a unitary transformation W and general complex v

$$v^\dagger v = v^\dagger W^\dagger W v = (Wv)^\dagger (Wv) = w^\dagger w \tag{3}$$

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

$$\delta_{ij} = u_j^\dagger u_i = u_j^\dagger W^\dagger W u_i = w_j^\dagger w_i \tag{4}$$

which we see has the same property in the transformed system. We can use these identities to construct tests after the Jabobis 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 that 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

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

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

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

$$T \propto n^a \tag{6}$$

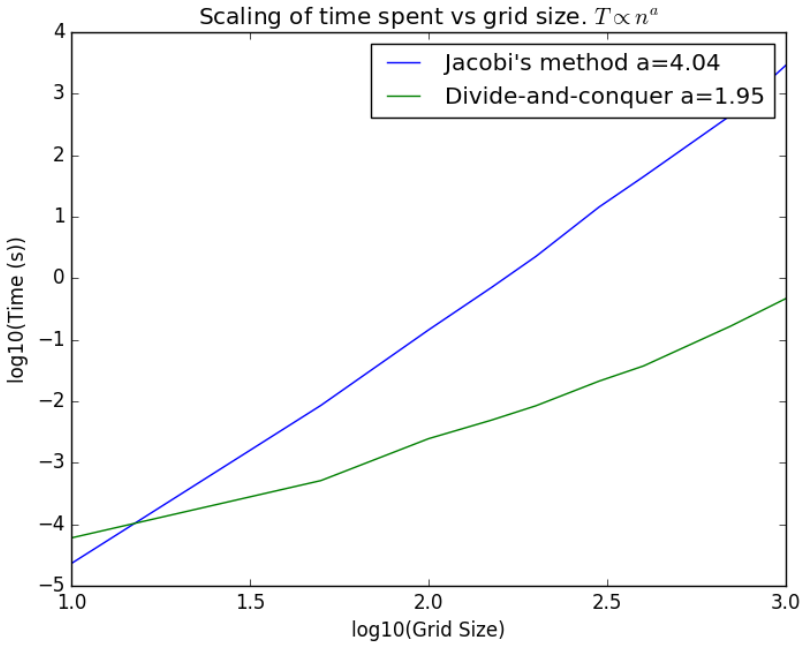

Figure 1: Time spend on finding eigenvalues/vectors for Jacobi's method and the Divide-and-conquer method. This data is from a run on a Macbook Pro 13'

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

In figure 1 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

$$T \approx n^{4.0} \tag{8}$$

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.

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 that 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