Contents

1 Abstract

All files for this project are available at GitHub[1].

2 Introduction

3 Theory

3.1 Orthogonal transformations of vectors

Jacobi's method relies heavily on orthogonal transformations and their properties. Orthogonal transformations are, in $\mathbb{R}^n$, functions on the form

$$T : \vec{x} \mapsto U\vec{x}$$

[1] https://github.com/anjohan/Offentlig/tree/master/FYS3150/Oblig2

where $\vec{x}$ is a vector in $\mathbb{R}^n$ and U is a real, orthogonal $n \times n$ matrix.

If U is orthogonal, $U^T U = I$. This implies:

$$T(\vec{v}_i) \cdot T(\vec{v}_j) = (U\vec{v}_i) \cdot \left(U\vec{v}_j\right) = (U\vec{v}_i)^T \left(U\vec{v}_j\right) = \vec{v}_i^T \overbrace{U^T U}^{I} \vec{v}_j = \vec{v}_i^T \vec{v}_j = \vec{v}_i \cdot \vec{v}_j$$

Orthogonal transformations hence conserve the inner product, and therefore also the orthogonality.

3.2 Orthogonal similarity transformations of matrices

If we have a matrix A with an eigenvector $\vec{x}$ with corresponding eigenvalue λ, then

$$A\vec{x} = \lambda\vec{x}$$

Multiplying with the transpose of an orthogonal matrix S, we get

$$S^T A\vec{x} = \lambda S^T \vec{x}$$

As S is orthogonal, $SS^T = I$, so we can sneak in an identity matrix disguised as SS^T between A and $\vec{x}$ on the left hand side:

$$\begin{aligned} S^T A S S^T \vec{x} &= \lambda S^T \vec{x} \\ \left(S^T A S\right)\left(S^T \vec{x}\right) &= \lambda\left(S^T \vec{x}\right) \end{aligned} \tag{1}$$

This proves that if A is transformed to $S^T AS$, the eigenvalues remain the same, while the new eigenvector is $S^T\vec{x}$.

It is easy to prove that symmetry is conserved by the similarity transformation:

$$\left(S^T A S\right)^T = S^T A^T \left(S^T\right)^T = S^T A^T S = S^T A S$$

if and only if $A^T = A$, i.e. when A is symmetric.

It can also be shown that the Frobenius norm, the square root of the sum of the squares of the matrix elements, is also conserved by the transformation[1].

3.3 Strategy for finding eigenvalues and eigenvectors, using orthogonal similarity transformations

If the transformation matrix S is chosen in a special way such that the largest non-diagonal element of A is transformed to 0, the sum of the squares of the non-diagonal elements decreases from the transformations. This means that if the transformation is applied repeatedly, the matrix is converging towards a diagonal matrix D.

If $D = \text{diag}\{d_1, d_2, \ldots, d_n\}$, the characteristic polynomial is given by

$$\det(\lambda I - D) = (\lambda - d_1)\cdot(\lambda - d_2)\cdot\ldots\cdot(\lambda - d_n)$$

and hence the eigenvalues of D are simply the diagonal elements. The eigenvectors can be chosen as the elements in the standard basis for $\mathbb{R}^n$, i.e. the columns of the identity matrix.

When A is transformed to D, the matrix of eigenvectors, R, is therefore transformed to I. On the other hand, we know from equation (1) on the preceding page that if $\vec{x}$ is an eigenvector of A, then $S^T\vec{x}$ is an eigenvector of the transformed matrix. If A is transformed into D after n transformations with the transformation matrices $S_1, S_2, \ldots, S_n$, then each eigenvector $\vec{x}$ is transformed to $S_n^T S_{n-1}^T \ldots S_1^T \vec{x}$. We must therefore have

$$I = S_n^T S_{n-1}^T \ldots S_1^T R \implies R = S_1 S_2 \ldots S_n I = I S_1 S_2 \ldots S_n \tag{2}$$

The conclusion is that if a suitable transformation matrix can be chosen, we have a method for finding both the eigenvalues and the eigenvectors of any symmetric matrix.

3.4 Choosing the transformation matrix

Jacobi discovered that one could choose a matrix S which is identical to the identity apart from the following elements:

$$s_{kk} = s_{ll} = \cos(\theta), \quad s_{kl} = \sin(\theta), \quad s_{lk} = \sin(\theta)$$

where (k,l) are the indices of the largest non-diagonal element of the matrix to be transformed, and θ is to be determined in such a way that the largest non-diagonal elements are transformed to 0.

With $B = S^T A S$, all elements apart from those in row or column k and l are left unchanged. The other elements are transformed as follows:

$$\begin{aligned}
b_{ik} &= b_{ki} = a_{ik}\cos(\theta) - a_{il}\sin(\theta), \quad i \neq k, i \neq l \\
b_{il} &= b_{li} = a_{il}\cos(\theta) + a_{ik}\sin(\theta), \quad i \neq k, i \neq l \\
b_{kk} &= a_{kk}\cos^2(\theta) - 2a_{kl}\cos(\theta)\sin(\theta) + a_{ll}\sin^2(\theta) \\
b_{ll} &= a_{ll}\cos^2(\theta) + 2a_{kl}\cos(\theta)\sin(\theta) + a_{kk}\sin^2(\theta) \\
b_{kl} &= b_{lk} = (a_{kk} - a_{ll})\cos(\theta)\sin(\theta) + a_{kl}\left(\cos^2(\theta) - \sin^2(\theta)\right)
\end{aligned}$$

Additionally, we know from equation (2) that the new eigenvector matrix is transformed by multiplying by S from the right hand side, which yields the matrix $\tilde{R}$. All elements are left unchanged, apart from

$$\begin{aligned}
\tilde{r}_{ik} &= r_{ik}\cos(\theta) - r_{il}\sin(\theta) \\
\tilde{r}_{il} &= r_{ik}\sin(\theta) + r_{il}\cos(\theta)
\end{aligned}$$

We must now choose θ such that a_{kl}, the largest non-diagonal element, is transformed to 0. This means solving the equation $b_{kl} = 0$:

$$0 = b_{kl} = (a_{kk} - a_{ll})\cos(\theta)\sin(\theta) + a_{kl}\left(\cos^2(\theta) - \sin^2(\theta)\right)$$

This leads to

$$\begin{aligned}(a_{ll} - a_{kk})\cos(\theta)\sin(\theta) &= a_{kl}\left(\cos^2(\theta) - \sin^2(\theta)\right)\\ \frac{a_{ll} - a_{kk}}{a_{kl}} &= \frac{\cos^2(\theta) - \sin^2(\theta)}{\cos(\theta)\sin(\theta)}\end{aligned}$$

Defining the left hand side as 2τ and dividing by $\cos^2(\theta)$ in the numerator and denominator on the right hand side, we get

$$\begin{aligned}2\tau &= \frac{1 - \tan^2(\theta)}{\tan(\theta)}\\ 0 &= \tan^2(\theta) - 2\tau\tan(\theta) - 1\\ \tan(\theta) &= -\tau \pm \sqrt{\tau^2 + 1}\end{aligned}$$

Either sign in front of the square root can in principle be chosen. Numerically, however there is a difference. The point of the algorithm is to get the non-diagonal elements, of which a_{kl} is the largest, as small as possible, which means τ will grow larger for every iteration. When τ is large, $\sqrt{\tau^2 + 1} \approx |\tau|$.

If τ is positive and the + sign is chosen (or vice versa), a large, negative number and a large, positive number are added and give approximately 0. If for example the computer is able to store 8 digits in the mantissa and the result is 6 orders of magnitudes smaller, the result will only have 2 accurate digits.

The conclusion is that if τ is negative, the + sign should be chosen, while if τ is positive, the − sign should be chosen.

$\cos(\theta)$ and $\sin(\theta)$ can then be calculated from the formulas

$$\cos(\theta) = \frac{1}{\sqrt{1 + \tan^2(\theta)}} \qquad \sin(\theta) = \tan(\theta)\cos(\theta)$$

3.5 Implementation of the algorithm

```
    max = largest_nondiagonal_symmetric(A,n,kl);
    do {
        k = kl[0]; l = kl[1];
        a_kk = A[k][k]; a_ll = A[l][l]; a_kl = A[k][l];
        tau = (a_ll-a_kk)/(2*a_kl);
        tangens = -tau - (tau>0 ? -1 : 1)*sqrt(1 + tau*tau);
        cosinus = 1/sqrt(1 + tangens*tangens);
        sinus = tangens*cosinus;

        A[k][k] = a_kk*cosinus*cosinus - 2*a_kl*cosinus*sinus\
                  + a_ll*sinus*sinus;
        A[l][l] = a_ll*cosinus*cosinus + 2*a_kl*cosinus*sinus\
                  + a_kk*sinus*sinus;
        A[k][l] = 0;
        A[l][k] = 0;
        for(i=0; i<n; i++){
            if(i!=k && i!=l){
                a_ik = A[i][k]; a_il = A[i][l];
                A[i][k] = a_ik*cosinus - a_il*sinus;
                A[i][l] = a_il*cosinus + a_ik*sinus;
                A[k][i] = A[i][k]; A[l][i] = A[i][l]; //Symmetric
            }
            r_ik = R[i][k]; r_il = R[i][l];
            R[i][k] = r_ik*cosinus - r_il*sinus;
            R[i][l] = r_ik*sinus + r_il*cosinus;
        }

        iterations++;
        max = largest_nondiagonal_symmetric(A,n,kl);
    } while(iterations <= max_iterations && max > tolerance);
```

4 Results and discussion

4.1 Simulation for one particle

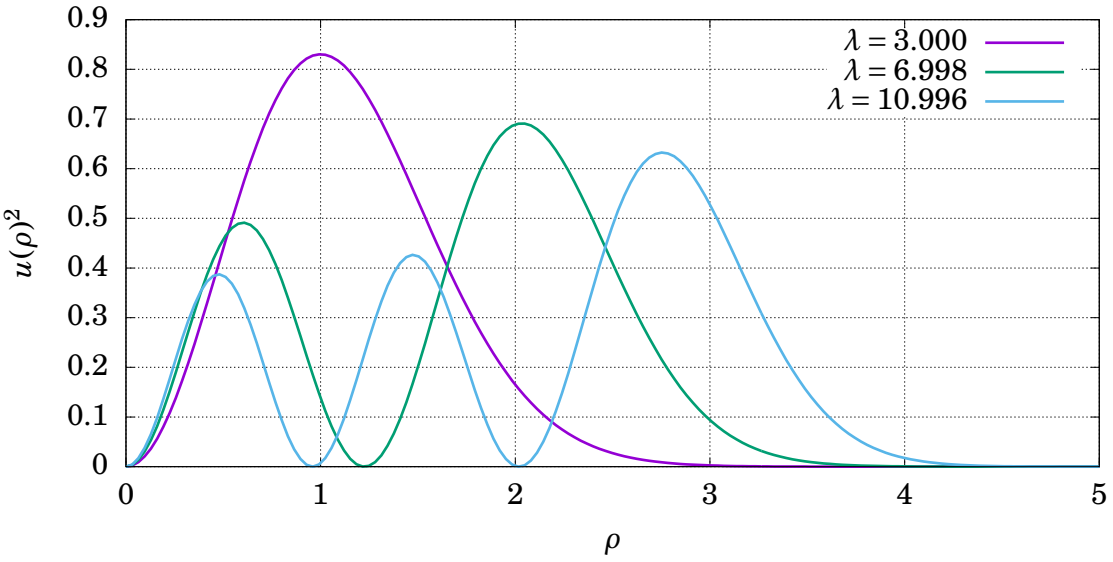

Figure 1: Simulation of one particle in a harmonic oscillator potential for $n = 250$ with $\rho_{\max} = 8$.

n	λ_0	λ_1	λ_2	Number of iterations
50	2.992	6.960	10.902	2863
100	2.998	6.990	10.976	12307
150	2.999	6.996	10.989	28353
200	2.999	6.997	10.994	51095
250	3.000	6.998	10.996	80474

Table 1: The three lowest eigenvalues found by the algorithm for the different number of mesh points n. The analytical values are $\lambda_0 = 3$, $\lambda_1 = 7$ and $\lambda_2 = 11$.

The number of iterations is approximately quadrupled when the number of mesh points is doubled, indicating that the numbers of iterations runs as n^2. This means that the number of iterations is proportional to the number of elements in the matrix. Note that the matrices handled in this project are tridiagonal, and the trends may be different for dense matrices.

4.2 Simulation for two particles

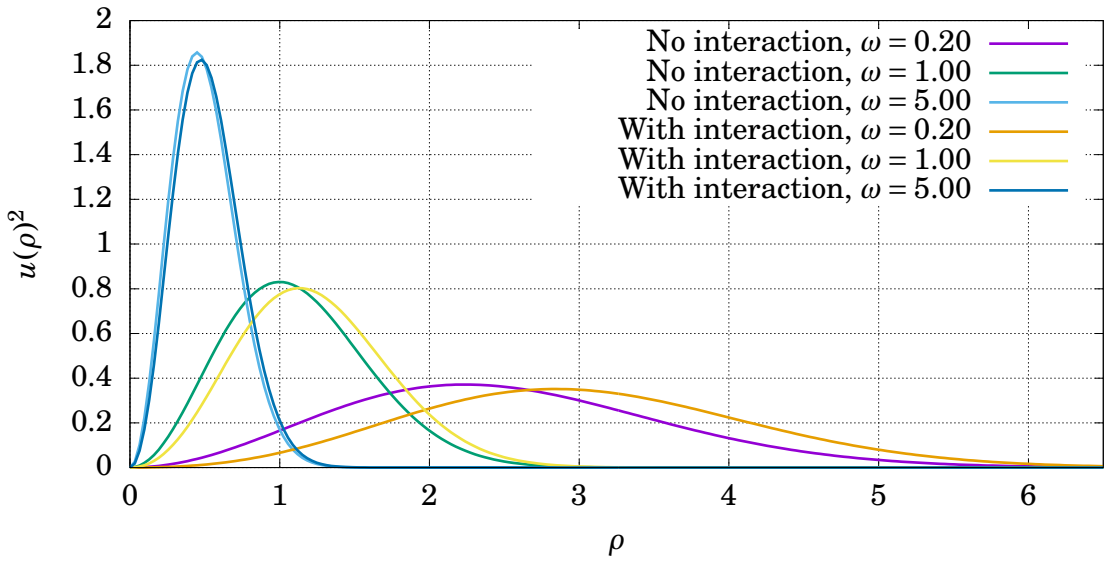

Figure 2: Simulation of two particles, with and without Coulomb interaction, for two different values of ω. A larger ω corresponds to a stronger harmonic oscillator.

5 Tests

The code is found at GitHub[2].

5.1 Ability to find eigenvalues and eigenvectors

The matrix

$$A = \begin{bmatrix} 7 & -2 & 0 \\ -2 & 6 & -2 \\ 0 & -2 & 5 \end{bmatrix}$$

has the pretty eigenvalues 3, 6 and 9, with corresponding eigenvectors

$$\begin{bmatrix} 1 \\ 2 \\ 2 \end{bmatrix}, \quad \begin{bmatrix} -2 \\ 1 \\ 2 \end{bmatrix}, \quad \begin{bmatrix} 2 \\ -2 \\ 1 \end{bmatrix}$$

The implementation of Jacobi's method can be tested on this matrix using

Code snippet 1: Test of eigenvalues.

```
A[0][0] = 7; A[0][1] = -2; A[0][2] = 0;
```

[2] https://github.com/anjohan/Offentlig/blob/master/FYS3150/Oblig2/test.cpp

```
    A[1][0] = -2; A[1][1] = 6; A[1][2] = -2;
    A[2][0] = 0; A[2][1] = -2; A[2][2] = 5;

    iterasjoner = jacobi(A,R,n, 1E-12);
```

where A and R are 3×3 pointer matrices and n is 3. Formatted printing of the resulting A and R yields

```
  9.000  0.000  0.000
  0.000  3.000  0.000
  0.000  0.000  6.000
-----------------------
  0.667  0.333 -0.667
 -0.667  0.667 -0.333
  0.333  0.667  0.667
```

This is the expected result, except that the program returns normalized eigenvectors.

5.2 Ability to find largest non-diagonal elements

A simple test is

Code snippet 2: Test of search for largest non-diagonal element.

```
    A[0][0] = 2;
    A[1][0] = 4; A[1][1] = 12;
    A[2][0] = -2; A[2][1] = -5; A[2][2] = 1;
    A[3][0] = 1; A[3][1] = 0; A[3][2] = 4; A[3][3] = 2;
    A[4][0] = 1; A[4][1] = 0; A[4][2] = 4; A[4][3] = 2; A[4][4] = -7;
    largest_nondiagonal_symmetric(A,5,kl);
    fprintf(file,"k=%d, l=%d",kl[0],kl[1]);
```

which yields

```
k=2, l=1
```

This is correct, as the function searches for the element with the largest absolute value. It is not necessary to specify the upper triangle of the matrix, as the function is specialized for symmetric matrices and therefore only searches the lower triangle.

Bibliography

[1] Morten Hjorth-Jensen. *Computational Physics*. Lecture notes. 2015. URL: https://github.com/CompPhysics/ComputationalPhysics/blob/master/doc/Lectures/lectures2015.pdf.