Contents

1 Abstract

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

2 Introduction

3 Theory

3.1 Orthogonal transformations

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

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

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:

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

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

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

4 Results and discussion

4.1 Simulation for one particle

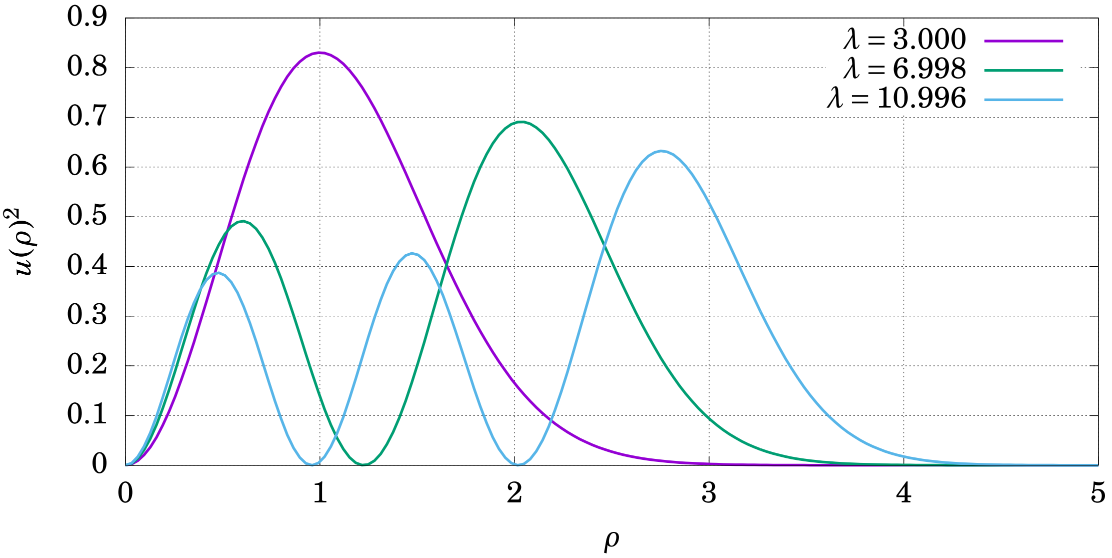

Figure 1: Simulation of one particle in a harmonic oscillator potential for $n = 250$ with $\rho_{\max} = 8$.

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	12 307
150	2.999	6.996	10.989	28 353
200	2.999	6.997	10.994	51 095
250	3.000	6.998	10.996	80 474

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

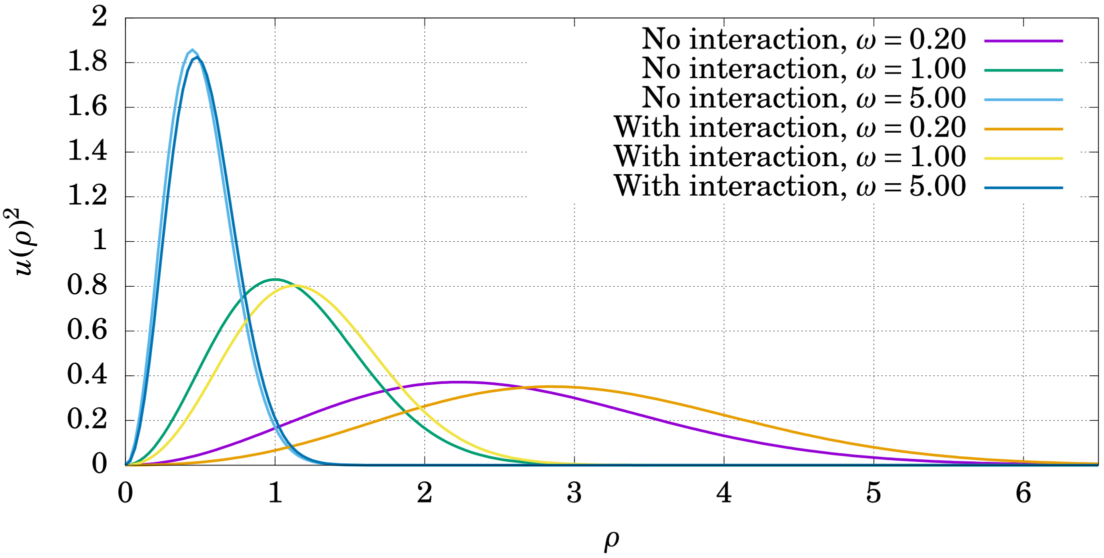

Figure 2: Simulation of two particles, with and without Coulomb interaction, for two different values of ω. A larger ω corresponds to a stronger harmonic oscillator.

5 Tests

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;
    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

```
  6.000  0.000  0.000
  0.000  9.000 -0.000
  0.000 -0.000  3.000
-----------------------
  0.667 -0.667  0.333
  0.333  0.667  0.667
 -0.667 -0.333  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;
    finn_storste_utenfor_diagonal_symmetrisk(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.