

Calculating eigenvalues using Jacobi's rotational algorithm

Anders P. Åsbø
(Dated: October 1, 2019)

The focus of this paper was the specific

CONTENTS

I. Introduction	1
II. Formalism	1
A. The buckling beam problem	1
B. The Jacobi rotational algorithm	1
III. Implementation	2
IV. Analysis	2
V. Conclusion	2
References	2
A. Program files	2
1. project.py	2
2. project_specialized.py	2
3. data_generator.py	2
4. erroranalysis.py	2
5. LUdecomp.py	2

I. INTRODUCTION

The focus of this paper was the implementation, and application of Jacobi's rotational algorithm to find the eigenvalues of Tridiagonal matrices numerically. Reliably finding eigenvalues is a crucial part of many scientific and mathematical disciplines. In this paper I considered the quantum mechanical application of electrons trapped in a harmonic oscillator potential.

II. FORMALISM

A. The buckling beam problem

The pretense for implementing Jacobi's algorithm is the classical problem of a beam of length L fastened in both ends $x_0 = 0$, $x_L = L$. The beam is allowed to be displaced in the y -direction with displacement $u(x)$, while $u(0) = u(L) = 0$. The displacement is driven by a force F at $(L, 0)$ towards the origin. The displacement is then described by

$$\gamma \frac{d^2 u(x)}{dx^2} = -Fu(x),$$

where γ is a constant dependent on the physical properties of the beam[1].

By scaling the differential equation with $\rho = \frac{x}{L}$, such that $\rho \in [0, 1]$, and introducing the parameter $\lambda = FL^2/\gamma$ [1], the differential equation becomes

$$\frac{d^2 u(\rho)}{d\rho^2} = -\lambda u(\rho).$$

Finally, the equation can be discretized with

$$u'' \approx \frac{u(\rho + h) - 2u(\rho) + u(\rho - h)}{h^2},$$

where $h = \frac{\rho_N - \rho_0}{N}$ with N steps[1]. The resulting discretization becomes

$$-\frac{1}{h^2}u_{i+1} + \frac{2}{h^2}u_i - \frac{1}{h^2}u_{i-1} = \lambda u_i,$$

which can be written as the matrix equation

$$A\vec{u} = \lambda\vec{u},$$

where A is an $(N-2) \times (N-2)$ tridiagonal Toeplitz matrix with the diagonal elements $d = 2/h^2$, and upper and lower diagonals with elements $a = -1/h^2$. This matrix happens to have analytical eigenvalues

$$\lambda_j = d + 2a \cos\left(\frac{j\pi}{N+1}\right),$$

where $j = 1, 2, \dots, N$ [1].

B. The Jacobi rotational algorithm

The goal of the Jacobi rotational algorithm is to reduce a matrix A to a diagonal matrix B where the elements along the diagonal are the eigenvalues λ_i of A . This is usually done by finding a matrix S such that

$$B = S^T A S,$$

and $S^T = S^{-1}$ [2]. The Jacobi algorithm achieves this by choosing the elements of S to be equal to the corresponding identity matrix, except for the elements $s_{kk}, s_{ll} = \cos \theta$ and $s_{kl} = \pm \sin \theta$, $s_{lk} = -s_{kl}$ [2], and applying the $B = S^T A S$ transformation repeatedly, until the non-diagonal elements of B are sufficiently close to zero. Doing this we get a system of equations for the various elements of the resulting matrix B

$$b_{ii} = a_{ii}i \neq k, i \neq l,$$

$$b_{ik} = a_{ik}c - a_{il}s, i \neq k, i \neq l,$$

$$b_{il} = a_{il}c + a_{ik}s, i \neq k, i \neq l,$$

$$b_{kk} = a_{kk}c^2 - 2a_{kl}cs - a_{ll}s^2,$$

$$b_{ll} = a_{ll}c^2 - 2a_{kl}cs - a_{kk}s^2,$$

$$b_{kl} = (a_{kk} - a_{ll})cs - a_{kl}(c^2 - s^2),$$

$$b_{lk} = -b_{kl}$$

where $c = \cos \theta$, $s = \sin \theta$, and k, l are chosen such that a_{kl} is the non-diagonal element in A with the largest absolute value.

III. IMPLEMENTATION

IV. ANALYSIS

V. CONCLUSION

-
- [1] Department of Physics, *Project 2 - Computational Physics I FYS3150/FYS4150*, Tech. Rep. (2019).
 - [2] M. Hjorth-Jensen, *Computational Physics Lectures: Eigenvalue problems*, Tech. Rep. (2019).

Appendix A: Program files

All code for this report was written in Python 3.6, and the complete set of program files can be found at <https://github.com/FunkMarvel/CompPhys-Project-2>.

1. project.py

<https://github.com/FunkMarvel/CompPhys-Project-1/blob/master/project.py>

2. project_specialized.py

https://github.com/FunkMarvel/CompPhys-Project-1/blob/master/project_specialized.py

3. data_generator.py

https://github.com/FunkMarvel/CompPhys-Project-1/blob/master/data_generator.py

4. erroranalysis.py

<https://github.com/FunkMarvel/CompPhys-Project-1/blob/master/erroranalysis.py>

5. LUdecomp.py

<https://github.com/FunkMarvel/CompPhys-Project-1/blob/master/LUdecomp.py>