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

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

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

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

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

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 $j = 1, 2, ..., N$[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

$$B = S^T AS,$$

and $S^T = S^{-1}$[2]. The Jacobi algorithm achieves this by choosing the elements of S to be equal to the coresponding 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

$B = S^T AS$ 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

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

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

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

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

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

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

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

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.

To chose an rotational angle θ, the quantities $\tan\theta = t = s/c$ are defined such that

$$\cot 2\theta = \tau = \frac{a_{ll} - a_{kk}}{2a_{kl}},$$

$$t = -\tau \pm \sqrt{1 + \tau^2},$$

$$c = \frac{1}{\sqrt{1 + t^2}},$$

$$s = ct,$$

[2].

C. Eigenvalues of a one-electron Hamiltonian

To test the implimentation of Jacobi's algorithm, as outlined in section II B, a quantum mechanical system consisting of an electron trapped in a radialy symetric harmonic oscillator potential $V(r) = \frac{1}{2}mkr^2$ was chosen as a test case. the corresponding radial equation is

$$-\frac{\hbar^2}{2m}\left(\frac{1}{r^2}\frac{d}{dr}r^2\frac{d}{dr} - \frac{l\,(l+1)}{r^2}\right)R(r) + V(r)R(r) = ER(r),$$

where $\hbar$ is Planck's constant, m is the electron mass, $R(r)$ is the radial wavefunction, and $l = 0, 1, 2, 3, ...$ is the orbital momentum of the electron[1]. By substituting in $R(r) = (1/r)u(r)$, and introducing the scaled variable $\rho = (1/\alpha)r$, where α is some constant, the equation can be reduced to

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

where α is chosen such that $\frac{mk}{\hbar^2}\alpha^4 = 1$, and

$$\lambda = \frac{2m\alpha^2}{\hbar^2}$$

[1].

After discretization as in section II A, the problem results in the system of linear equations

$$-\frac{1}{h^2}u_{i+1} + \left(\frac{2}{h^2} + \rho^2\right)u_i - \frac{1}{h^2}u_{i-1} = \lambda u_i,$$

with the requirement that $u(0) = u(\infty) = 0$[1]. This can be written as a matrix equation where the matrix A is a tridiagonal matrix with $d = \left(\frac{2}{h^2} + \rho^2\right)$ along the diagonal, and $e = -\frac{1}{h^2}$ as the non-diagonal elements. The step-size h is as defined in section II A.

D. Eigenvalues of a two-electron Hamiltonian

The radial equation for a non-interacting electron pair, can be written as the product of two single-electron radial equations

$$\left(-\frac{\hbar^2}{m}\frac{d^2}{dr^2} - \frac{\hbar^2}{4m}\frac{d^2}{dR^2} + \frac{1}{4}kr^2 + kR^2\right)u(r, R) = E^2 u(r, R),$$

where $r = r_1 - r_2$ is the difference between the radial coordinates of the two electrons, and $R = \frac{1}{2}(r_1 + r_2)$ is the coordinate of the center of mass[1].

By seperating the radial wavefunction $u(r, R) = \psi(r)\phi(R)$, and adding in the repulsive Coloumb interaction between the electrons $V(r) = \beta e^2/r$, where $\beta$is a konstant and e is the electron charge, the r-dependent Scroedinger equation becomes

$$-\frac{d^2}{d\rho^2}\psi(\rho) + \omega_r^2\rho^2\psi(\rho) + \frac{1}{\rho} = \lambda\psi(\rho),$$

where $\rho \in [0, \infty)$ is the same dimensionless variable from section II C, the frequency $\omega_r = \frac{1}{4}\frac{mk}{\hbar^2}\alpha^4$, α is chosen such that $\frac{m\alpha\beta e^2}{\hbar^2} = 1$, and $\lambda = \frac{m\alpha^2}{\hbar^2}E$[1].

Doing the same discretization as in section II A and section II C results in the linear equations

$$-\frac{1}{h^2}u_{i+1} + \left(\frac{2}{h^2} + \omega_r^2\rho^2 + \frac{1}{\rho}\right)u_i - \frac{1}{h^2}u_{i-1} = \lambda u_i,$$

and a tridiagonal matrix A with diagonal elements $d = \left(\frac{2}{h^2} + \omega_r^2\rho^2 + \frac{1}{\rho}\right)$ and non-diagonal elements $e = -\frac{1}{h^2}$. The step-size h is defined as in section II A.

This wave equation has analytical eigenvalues for certain ω_r as described in Taut, M.'s paper in Physical Review (1993) [3].

III. IMPLEMENTATION

A. Constructing the matrix and testing agianst the NumPy solver

As a prequel to implementing the jacobi algorithm, "program.py" was written to construct the tridiagonal Toeplitz matrix, and find its eigenvalues using the numpy.linalg.eig() method provided by the NumPy package. "program.py" also includes a test comparing the numerically found eigenvalues with the expected results from the analytical expression in section II B, with a tolerance of 10^{-10}.

B. Implementing Jacobi's rotational algorithm

Jacobi's algorithm was implemented in "jacobi.py". The afformentioned program takes in a value N, and sets up the parameters as well as creating the Toeplitz matrix using the function from "program.py". A mask is created and used when calculating the norm of the non-diagonal elements in the matrix. The jacobi function is called, returning the diagonal elements.

The jacobi function consists of a while-loop that runs until the norm of the non-diagonal elements are less than a tolerance of 10^{-20}. The loops starts by finding a_{kl} with the maximum absolute value, then uses the returned indices to retrive the a_{kk}, and a_{ll}. The parameters τ, t, c, s are all calculated, with t being chosen such that $|\theta| \leq \frac{\pi}{4}$ to ensure minimal difference between A and B[2].

Furthermore, a for-loop is started that runs through A and calculates the non-diagonal elements except for $b_{kl}, b_{lk}, b_{ll}, b_{kk}$. After the for-loop finishes, the remaining matrix elements are calculated, as well as the new norm of the non-diagonal elements, and the while-loop has completed one full loop.

When the norm of the non-diagonal elements is below the set tolerance, the while-loop finishes, and the matrix B is returned by the function.

The find_max function in "jacobi.py" uses two nested for-loops to run over every non-diagonal element in A and check if the current element has a larger absolute value than the previously stored a_{kl}, with the starting value being $a_{kl} = 0$. Once a valid value is found, it is stored and used to evaluate the remaining elements until it is replaced with a new maximum, or the loops end. The new a_{kl} is then returned together with the indices k, l.

Finally, the numerical eigenvalues are extracted, sorted and printed to the terminal.

The program files also includes "unittest.py", which tests the corespondence between the numerical eigenvalues found using the jacobi function on the tridiagonal matrix from section II A, and the analytical eigenvalues, with a tolerance of 10^{-10}. A second unit test in "unittest.py" tests if the find_max function is able to always pick out the element with the largest absolute value in a randomized 5×5-matrix.

C. Implementing the single-electron problem

To solve the eigenvalues of the problem outlined in section II C, "oneelectron.py" was written. The program takes in the number of grid points. Ten it sets the maximum value of ρ, which after trial and error was determined to be 12.5, with the computational resources I had available.

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).

[3] M. Taut, Phys. Rev. A **48**, 3561 (1993).

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. program.py

https://github.com/FunkMarvel/CompPhys-Project-2/blob/master/program.py

2. jacobi.py

https://github.com/FunkMarvel/CompPhys-Project-2/blob/master/jacobi.py

3. oneelectron.py

https://github.com/FunkMarvel/CompPhys-Project-2/blob/master/oneelectron.py

4. twoelectron.py

https://github.com/FunkMarvel/CompPhys-Project-2/blob/master/twoelectron.py

5. unittest.py

https://github.com/FunkMarvel/CompPhys-Project-2/blob/master/unittests.py