Eigenvalue problems, comparison of Jacobi's and Lanczos' algorithms applied to Schroedinger's equation for an electron in a threedimensional harmonic oscillator well

Johannes Sørby Heines

September 30, 2020

Abstract

We observed the behaviour of Jacobi's and Lanczos' algorithms for solving eigenvalue problems by applying them to the wave equation for a single electron in a harmonic oscillator potential. The two algorithms give a similar precision for the same step length h, but the efficiency of Lanczos' algorithm is much higher, allowing it to use smaller values of h and thus give more accurate results.

Introduction

In physics, we often encounter systems whose behaviour depends on their current state. In many cases these can be described or approximated by equations of the form

$$\frac{\mathrm{d}^2 u(x)}{\mathrm{d}x^2} = \lambda u(x).$$

When such equations are discretized they give rise to an eigenvalue problem

$$\mathbf{A}u = \lambda u.$$

There are many different algorithms for solving eigenvalue problems numerically. We explored two methods for cases in which $\mathbf{A}$ is a real symmetrical matrix: Jacobi's and Lanczos' algorithms.

We outline the mathematical workings of each algorithm, describe their numerical implementation in C++ and apply them to the simple case of a single electron in a harmonic oscillator potential.

Theoretical models and technicalities

All code can be found, along with unit tests and benchmarks at `https://github.com/johashei/ComputationalPhysics/tree/master/doc/Projects/2020/Project2/code`.

Numerical solutions to eigenvalue problems make use of similarity transformations:

$$\mathbf{B} = \mathbf{S}^T\mathbf{A}\mathbf{S}, \quad \text{where} \quad \mathbf{S}^T\mathbf{S} = \mathbb{1}.$$

These are useful because they preserve the eigenvalues [3] and the orthogonality of the eigenvectors.

Proof. Consider an orthogonal basis of vectors

$$\mathbf{v}_i = \begin{bmatrix} v_{i1} \\ \vdots \\ v_{in} \end{bmatrix}, \quad \mathbf{v}_j^T\mathbf{v}_i = \delta_{ij}.$$

Let $\mathbf{w}_i = \mathbf{S}^T\mathbf{v}_i\mathbf{S}$. Then

$$\begin{aligned} \mathbf{w}_j^T\mathbf{w}_i &= (\mathbf{S}^T\mathbf{v}_j\mathbf{S})^T(\mathbf{S}^T\mathbf{v}_i\mathbf{S}) \\ &= \mathbf{S}^T\mathbf{v}_j^T\mathbf{S}\mathbf{S}^T\mathbf{v}_i\mathbf{S} \\ &= \mathbf{S}^T\mathbf{v}_j^T\mathbf{v}_i\mathbf{S} \\ &= \mathbf{S}^T\delta_{ij}\mathbf{S} = \delta_{ij}. \end{aligned}$$

□

Jacobi's algorithm

Direct methods determine the eigenvalues of a matrix $\mathbf{A}$ by performing a series of similarity transformations

$$\mathbf{S}_N^T \cdots \mathbf{S}_1^T \mathbf{A} \mathbf{S}_1 \cdots \mathbf{S}_N = \mathbf{D},$$

such that $\mathbf{D}$ is a diagonal matrix. Because similarity transformations preserve the eigenvalues, the diagonal elements of $\mathbf{D}$ are the eigenvalues of $\mathbf{A}$. There is no uniquely defined series of similarity transformations which lead to the matrix $\mathbf{D}$. In Jacobi's algorithm, each transformation performs a rotation of $\mathbf{A}$ along an axis: the elements of $\mathbf{S}_i$ are given by

$$\begin{aligned} s_{kk} &= s_{ll} = \cos\theta, \\ s_{kl} &= -s_{lk} = -\sin\theta, \\ s_{jj} &= 1 \quad j \neq \{k, l\}. \end{aligned}$$

Each iteration finds the off-diagonal element of maximum value a_{kl} of $\mathbf{A}$, and sets θ such that $a_{kl} = 0$ after the rotation. This systematically reduces the Frobenius norm of the off-diagonal elements

$$\text{off}(A) = \sqrt{\sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} a_{ij}^2}.$$

When this value is close enough to zero, the algorithm ends giving approximate eigenvalues of $\mathbf{A}$.

When implementing this algorithm, the program only loops over half the matrix A since it's symmetrical. Once k and l are known it calculates the angle theta by [3]

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

Here the expression is written so as to avoid the calculation $\tau - \sqrt{1+\tau^2}$, which leads to loss of numerical precision for $\tau^2 >> 1$.

The lowest value of $\tan\theta$ is then used to calculate $\cos\theta$ and $\sin\theta$. When the program performs the rotation, each matrix element is only needed to calculate a few of the new ones. These are temporarily and the matrix $\mathbf{A}$ is gradually overwritten with the new values.

Calculating the Frobenius norm of the off-diagonal elements requires many operations, which would slow down the program considerably. To avoid this, we instead set a tolerance on the maximum off-diagonal element a_{kl} to stop the algorithm. We chose $|a_{kl}| < 10^{-8}$.

Lanczos' algorithm

Lanczos' algorithm is an iterative method: it constructs a tridiagonal $m \times m$-matrix $\mathbf{T}$ whose extremal eigenvalues converge to those of the $n \times n$-matrix $\mathbf{A}$ as $m \to n$ [1]. In this limit, $\mathbf{T} = \mathbf{Q}^T \mathbf{A} \mathbf{Q}$. Each iteration of the algorithm computes one column of $\mathbf{Q}$, and the convergence typically happens long before $m = n$ for the lowest and highest eigenvalues [1]. Lanczos' method is therefore useful in cases where n is very large and only the lowest eigenvalues are of interest.

The algorithm can be written as [1]

```
$r_0$ = arbitrary unit vector
$k = 0, \beta_0 = 1, q_0 = 0$
**while** $\beta_k \neq 0$ **do**
    $q_{k+1} = r_k/\beta_k$
    $k = k + 1$
    $\alpha_k = q_k^T \mathbf{A} q_k$
    $r_k = \mathbf{A} q_k - \alpha_k q_k - \beta_{k-1} q_{k-1}$
    $\beta_k = \|r_k\|_2$
**end while**
```

where α and β are the diagonal and off-diagonals of $\mathbf{T}$ respectively.

Note that when implementing this algorithm, the loop is broken after a predetermined number of iterations m, chosen depending on the desired precision and number of eigenvalues. Computation cycles can be saved by only calculating $\mathbf{A} q_k$ once in each loop.

This direct implementation of the algorithm is known to be unstable due to loss of numerical precision [1], but as no such effects were observed in our experiments, we have not treated this problem.

The obtained $m \times m$-matrix $\mathbf{T}$ was diagonalized using Armadillo's `eig_sym()` method.

Single electron in a harmonic oscillator potential

An electron in a harmonic oscillator potential can occupy energy levels given by the time independent radial Schrödinger equation [2]

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

Here $R(r)$ is the radial wave function, $V(r) = kr^2/2$ is the harmonic oscillator potential, and the quantum number l is the electron's orbital angular momentum. This is an eigenvalue problem where the eigenvalues E represent the energy levels.

In order to apply the algorithms above to equation 1, we introduce new variables to obtain a dimensionless equation. We only look at the case where $l = 0$.

By making the substitution $R(r) = u(r)/r$, we get

$$-\frac{\hbar^2}{2m}\frac{\mathrm{d}^2}{\mathrm{d}r^2}u(r) + V(r)u(r) = Eu(r)$$

We then define $\rho = r/\alpha$ where α is a constant with dimension length, giving $V(\rho) = k\alpha^2\rho^2/2$, and thus

$$-\frac{\hbar^2}{2m\alpha^2}\frac{\mathrm{d}^2}{\mathrm{d}\rho^2}u(\rho) + \frac{k}{2}\alpha^2\rho^2u(\rho) = Eu(\rho).$$

Now we can choose α so that $mk\alpha^4/\hbar^2 = 1$ and define $\lambda = 2m\alpha^2E/\hbar^2$. This allows us to rewrite the Schrödinger equation as

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

This equation can then be discretized, giving the eigenvalue equation $\mathbf{A}u = \lambda u$, where $\mathbf{A}$ is a tridiagonal matrix

$$\mathbf{A} = \begin{bmatrix} d_1 & e_1 & & \\ e_1 & d_2 & \ddots & \\ & \ddots & \ddots & e_{N-2} \\ & & e_{N-2} & d_{N-1} \end{bmatrix},$$

with $e_i = -1/h^2$ and $d_i = 2/h^2 + \rho_i^2$, where $h = (\rho_N - \rho_0)/N$ is the step length [2].

Comparison of the algorithms

We ran each algorithm for various values of N and ρ_N, along with the "`eig_gen()`" method from the Armadillo library. The first four eigenvalues are expected from analytical calculations [2] to be $\lambda = 3, 7, 11, 15$ in the limit $N, \rho_N \to \infty$.

While the exact number of iterations required by Jacobi's algorithm depends on the matrix, the number is expected to be of the order of $3N^2$ to $5N^2$ [3]. We ran Jacobi's algorithm up to $N = 500$, with $\rho_N = 10$, and used the `Ctime` library to measure the algorithm's runtime.

Unlike Jacobi's algorithm, our program applies Lanczos' algorithm for a predetermined number of iterations M. In our experiments we chose to use $M = N/5$, as this gave a similar precision to that of `eig_gen` for the four lowest eigenvalues. We also used $\rho_N = 30$ as a better approximation to infinity. We ran the program up to $N = 5000$ and measured it's runtime.

Results

Figure 1 shows a plot of the difference between the first four eigenvalues of $\mathbf{A}$ as obtained by Jacobi's method and the four lowest energy states of the electron, for various N. The number of iterations of Jacobi's algorithm is listed in table 1. The runtime for both the Jacobi's algorithm and Armadillo's "`eig_gen()`" method are also shown.

Results for Lanczos' algorithm are shown in figure 2, and a comparison of the runtimes

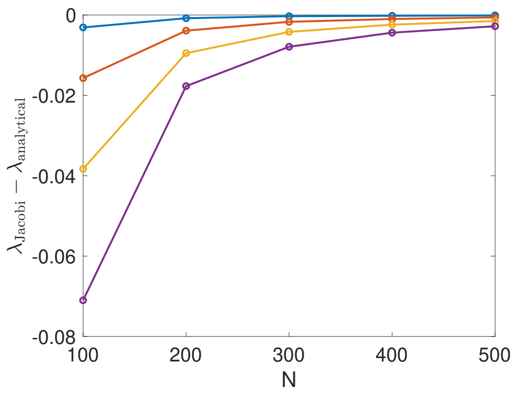

Figure 1: Plot of the difference between the eigenvalues calculated by the Jacobi algorithm and the analytical states for the electron. From top to bottom, $\lambda_{\text{analytical}} = \{3, 7, 11, 15\}$.

Table 1: For a range of N: number of iterations used by Jacobi's algorithm, it's runtime t_{Jacobi}, and that of Armadillo's "`eig_gen()`" method t_{arma}. A fit of the number of iterations to (aN^2) yields $a = 1.6$.

N	iterations	t_{Jacobi}[s]	t_{arma}[s]
100	15 290	0.81	0.0069
200	63 410	12.5	0.10
300	144 732	66	0.25
400	259 167	210	0.64
500	407 112	515	0.92

with those of Armadillo's "`eig_gen`" method are listed in table 2.

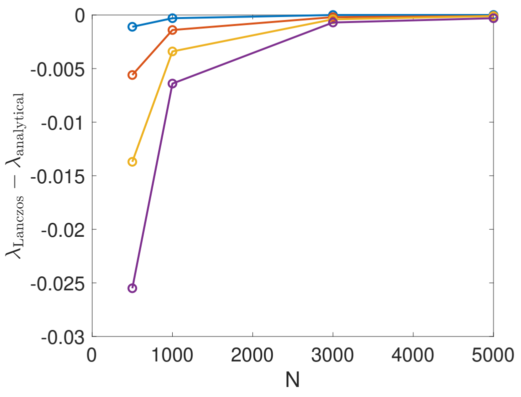

Figure 2: Plot of the difference between the eigenvalues calculated by the Lanczos algorithm with $M = N/5$ iterations and the analytical states for the electron. From top to bottom, $\lambda_{\text{analytical}} = \{3, 7, 11, 15\}$.

Table 2: For various N: runtime of the Lanczos algorithm with $M = N/5$ iterations, and of Armadillo's "`eig_gen()`" method.

N	t_{Lanczos} [s]	t_{arma} [s]
500	0.029823	0.819871
1000	0.208129	3.34472
3000	5.73699	62.5238
5000	48.2992	290.692

Discussion

In all cases Jacobi's algorithm gives the same result as Armadillo's "`eig_gen()`" method up to at least four decimal places. This is expected since it diagonalizes the matrix $\mathbf{A}$ to a tolerance set by us. The number of iterations for Jacobi's algorithm is of the order of $1.6N^2$. This is slightly less than the expected $3N^2$ to $5N^2$, but still very inefficient compared to `eig_gen()`. Our 499×499-matrix required 407 122 iterations and 515 s to diagonalize, whereas Armadillo dit it in 0.9 s. The higher than expected efficiency might come from the fact that the matrix $\mathbf{A}$ starts out tridiagonal.

With $M = N/5$ iterations, the Lanczos algorithm also gives similar results to Armadillo. However, it can be applied to much larger matrices, allowing for much more accurate estimations of the eigenvalues. It is also much faster, especially for high N. This is expected as each iteration only involves one matrix operation, and it only calculates the eigenvalues of a much smaller matrix. The behaviour of Lanczos' algorithm should be compared to quantitative predictions in future work.

Conclusion

We implemented Jacobi's and Lanczos' algorithms and applied them to the eigenvalue problem of a single electron in a harmonic oscillator potential. In both cases we compared the results for the four lowest eigenvalues to the analytical energy levels, and the runtime of the algorithms to that of Armadillo's "`eig_gen()`" method. While all three methods gave results of similar accuracy for the same N, the difference in efficiency meant we could obtain much more accurate values with Lanczos' algorithm by using higher values of N.

The qualitative behaviour of both algorithms is consistent with theory, though we measured the Jacobi algorithm to be slightly more efficient that expected, likely due to the simplicity of the problem we chose to study. A qualitative comparison with theory for Lanczos' algorithm is left to future work.

References

[1] Golub, Gene H., Van Loan, Charles F. (2013) *Matrix Computations*, Fourth edition. Baltimore: The John Hopkins University Press.

[2] Hjort-Jensen, Morten, (2020) *Project 2*, <`http://compphysics.github.io/ComputationalPhysics/doc/Projects/2020/Project2/html/Project2.html`>.

[3] Hjort-Jensen, Morten, (2020) *Computational Physics Lectures: Eigenvalue Problems*, <`http://compphysics.github.io/ComputationalPhysics/doc/pub/eigvalues/html/eigvalues.html`>.