Fys3150 - Project 2 - Eigenvalue problems

Sakarias Frette
Anders Vestengen
William Hirst

September 30, 2020

Abstract

The Jacobi method for solving eigenvalueproblems is an easily understandable method, practical when new to computational eigenvalueproblems. It's long running time however makes it a poor choice when dealing with large matrices. We achieved an accuracy of 4 decimals for a 5×5 matrix when comparing our results to the Armadillo eigsymfunction, and when solving the Schroedinger equation with different potentials, we had an accuracy of at least 2 decimals.

1 Introduction

When solving numerical problems we sometimes have an intuition for what the solution will be. When integrating the path of a ball thrown through the air, we already know what the answer should look like. However, for most numerical problems this is not the case. How can we then check that our solution is correct, or if it even makes sense? The answers is comparing to known values or solutions.

In this report we will show how we can use Jacobi's method to solve a differential equation for a buckling beam problem(or a simple spring g fastened at both ends), then advancing this to solve the Schrödinger equation for a harmonic oscillator. While Jacobi's method is not a fast method, it has other advantages. For example the obvious use of many clearly stated mathematical methods comprising the algorithm gives us several clear ways of testing code to analytical solutions.

2 Method

2.1 Differential equations

The buckling beam problem gives us the following differential equation:

$$\gamma \frac{d^2u(x)}{dx^2} = -Fu(x) \tag{1}$$

with u(x) being the vertical displacement of the beam in the y direction. This equation is defined over a distance $x \in [0, L]$ where L is a known length. F is here some force with known magnitude. This equation can be rewritten as an eigenvalue problem. We first have to make the equation dimensionless, using a variable $\rho = x/L$, where L is a reference length where the boundary condition for the equation is. We then find the new equation:

$$\frac{d^2u(\rho)}{d\rho^2} = -\frac{FL^2}{\gamma}u(\rho)$$

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

which now is an eigenvalue problem. We can discretize this in order to solve this equation numerically. Discretizing the second derivative gives us this equation:

$$u'' = \frac{u(\rho + h) + -2u(\rho) + u(\rho - h)}{h^2} + O(h^2) \tag{3}$$

This ρ will be defined at the interval $\rho \in [\rho_{min}, \rho_{max}] = [0, 1]$, and for a given set of integration points, we have that $h = (\rho_{max} - \rho_{min})/N$. This gives us that $\rho_i = \rho_{min} + i \cdot h$, $i = 1, 2, .., N$. Our discretized equation is then:

$$-\frac{u(\rho_i + h) + -2u(\rho_i) + u(\rho_i - h)}{h^2} = \lambda u(\rho_i)$$

or more compact as

$$-\frac{u_{i+1} + -2u_i + u_{i-1}}{h^2} = \lambda u_i \tag{4}$$

This equation can be rewritten as a matrix equation:

$$\begin{bmatrix} d & a & 0 & 0 & \dots & 0 & 0 \\ a & d & a & 0 & \dots & 0 & 0 \\ 0 & a & d & a & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & \dots & \dots & a & d & a \\ 0 & \dots & \dots & \dots & \dots & d & a \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_{N-2} \\ u_{N-1} \end{bmatrix} = \lambda \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_{N-2} \\ u_{N-1} \end{bmatrix} \tag{5}$$

We define here $a = -1/h^2$ and $d = 2/h^2$.

2.2 Unitary matrices

For a given unitary matrix A, we have that the dot product between two unitary transformed vectors $\vec{w} = A\vec{v}$ and $\vec{u} = A\vec{y}$ is

$$\vec{w} \cdot \vec{u} = (A\vec{v})^T \cdot A\vec{y} = (A^T\vec{v}^T) \cdot A\vec{y} = \vec{v}^T(A^TA)\vec{y} = \vec{v} \cdot \vec{y} \tag{6}$$

This shows that a unitary transform preserves the dot product. Since this also proves linearity we can state that this unitary transform also preserves orthogonality.

2.3 Jacobi method

Numerically speaking, the Jacobi method is an iterative method for diagonalizing matrices until all off-diagonal elements reach a sufficiently small threshold. The resulting matrix should have all of its eigenvalues on the diagonal.
Jacobi's method uses the following theorem; for any symmetrical real nxn matrix A, there exists a matrix S so that $S^TAS = D$, where

$$D = \begin{bmatrix} \lambda_1 & 0 & 0 & \dots & 0 & 0 \\ 0 & \lambda_2 & 0 & \dots & 0 & 0 \\ 0 & 0 & \lambda_3 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & \dots & \dots & \lambda_{n-1} & 0 \\ 0 & \dots & \dots & \dots & 0 & \lambda_n \end{bmatrix} \tag{7}$$

In our case A is known and is the tridiagonal Toeplitz matrix, but S is unknown. Therefore instead of directly finding D, we minimize the difference between A and D through many iterations with different S matrices,

S_i, updating A to converge towards D. D then becomes $D \approx S_1^T S_2^T ... S_n^T A S_1 S_2 ... S_n$.

Since S is unknown, diagonalizing A in one iteration is impossible, instead we minimize each non-diagonal element separately. We do this by multiplying our matrix A with a rotational matrices, S_i and S_i^T . S_i is defined to turn the largest non-diagonal element in A, a_{pq} (and a_{qp}) to zero [1]. The multiplication of $S_i^T A S_i$ is therefore dependent on the value of the largest non-diagonal elements of A. Repeating the method a sufficient amount of times, every non-diagonal element on A will be approximately be zero (dependent on our threshold), and $A \approx D$. When $A \approx D$ for a sufficient n, S_n approximates a matrix with the eigenvectors in it's columns.

2.4 The quantum mechanical aspect

We will extend our Jacobi method to solving Schrodinger equation with spherical coordinates. Assuming a spherical symmetric potential, we can rewrite Schrodinger equation to [2]:

$$-\frac{d^2}{d\rho^2}u(\rho) + V(\rho)u(\rho) = \lambda u(\rho) \tag{8}$$

Where λ is the eigenvalue of the equation i.e. energy level. This can be written to the same matrix equation as (5), with the potential $V(\rho)$ added on the diagonals.

3 Implementation

3.1 The algorithm

The code for this project is found here. This code is using object-oriented classes for the code structure. This allows us to have a tidy implementation/main-function for each potential. It also makes implementation of unit-tests easier. We call our class classtuff. Classtuff contains 5 functions.

The first of which is initialize. Initialize creates our matrix A as a function of our time-step h and the potential $V(\rho)$. Secondly we have Jacobi_aram which returns armadillos solution for our initialized matrix. Thirdly we have offdiag, returning the indices of the largest non-diagonal element.

[1] For the exact definition of S_i and the multiplication of $S_i^T A S_i$, see Git-hub

[2] For full mathematical derivation, see her

Fourthly we have Rotate, which rotates our matrix, setting the largest non-diagonal element to 0 (2.3). Finally we have Jacobi. Jacobi iterates through all of the functions until our threshold is reached, therefor Jacobi contains our algorithm:

Set size, $maxiter = size^3$;
Set $iter = 0$, $ndem = 1$;
while *abs(ndem)* > $\epsilon = 10^{-8}$ *and iter* <= *maxiter* **do**
    Find Offdiagonal max value in A;
    Update maxoff, p, q;
    Rotate A,S;
    Set $ndem = maxoff$;
    Update iter;
end

Using our code we test for three potentials, $V(\rho) = 0$, $V(\rho) = \rho^2$, $V(\rho) = \omega^2\rho^2 + \frac{1}{\rho}$. For $V(\rho) = 0$, we have no information on eigenvalues. Therefore we compare our results to those of armadillos-solve. We then use our S_n matrix to plot the lowest energy eigenvector against the analytical solution for the eigenvector ($u_0(\rho) = \sin(\frac{\pi i}{N})$, i = 1,2,...,N-1).

For $V(\rho) = \rho^2$, the first eigenvalues ($\lambda_1 = 3, \lambda_2 = 7, \lambda_3 = 11, \lambda_4 = 15$) are known. We will therefore vary ρ_{max} (2.1), until the numerical eigenvalues match the analytical.

For $V(\rho) = \omega^2\rho^2 + \frac{1}{\rho}$, the eigenvalues are known [2]. Therefore we will compare our eigenvalues, to those known.

3.2 The Unit Tests

As discussed in the introduction, it is important to test the code, especially in the early stages of the project. We chose to implement a unit-test that test three aspects of the code; if offdiag finds the correct max value, if the S- and S^T-matrix columns are orthonormal and if the algorithm can find the (approximately) correct eigenvalues when compared to armadillos-solve for a simple 3x3 matrix.

[2]For the documentation of the analytical eigenvalues ($\lambda_1 = 3, \lambda_2 = 7, \lambda_3 = 11, \lambda_4 = 15$), click here

4 Results

For the wave function problem described in section 2.1 with ($V(\rho) = 0$), we get the following result for a $5x5$ matrix:

Eigenvalue	Jacobi	Armadillo
1	9.6462	9.6462
2	3.6000e+01	3.6000e+01
3	7.2000e+01	7.2000e+01
4	1.0800e+02	1.0800e+02
5	1.3435e+02	1.3435e+02

Table 1: The table shows 5 eigenvalues for both Jacobis and armadillos solution for the problem with $V(\rho) = 0$.

The results show that up to (at least) 4 decimals our numerical solution matches that of armadillo. When testing with $n = 300$, the Jacobi method is also 361 times slower than the Armadillo "eig sym" function.

With the eigenvalues correctly calculated, we now compare our calculated eigenvector corresponding to the lowest eigenvalue with the analytical solution. The comparison is shown in figure (1).

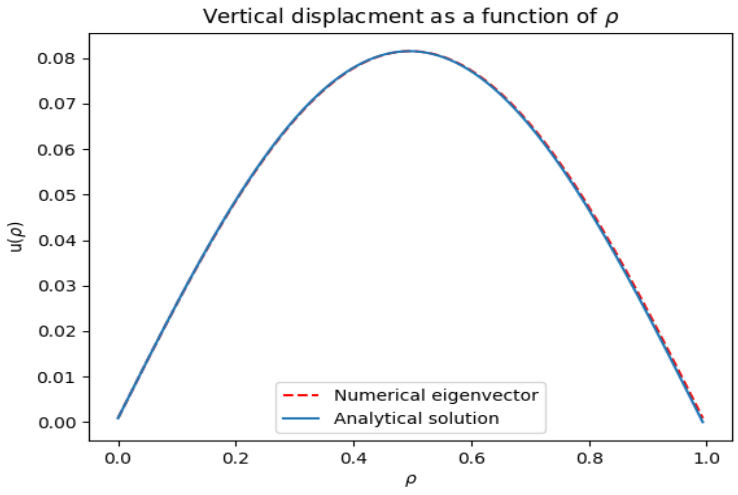

Figure 1: Figure showing the comparison between our numerical and the analytical solution for vertical displacement, plotted with a resolution of n=100

From the figure it is clear that the numerical calculation of the eigenvector for the lowest eigenvalue is a good approximation to the analytical solution. Note that to make the comparison, we had to scale the analytical solution to fit our eigenvector. Because of the nature of eigenvectors, scaling will not affect the result.

Nr	**Analytical**	**Computed with $\rho = 5$**
1	3	2.99991
2	7	6.99957
3	11	10.9991
4	15	15.0039

Table 2: The table shows our computed eigenvalues with a potential $V(\rho) = \rho^2$ against the analytical solutions.

The table above (table 2) shows our numerical values compared to the known. We can see that the accuracy of the numerical eigenvalue decreases for each eigenvalue. For the lowest eigenvalue the result is accurate up till the fourth decimal, while for the fourth eigenvalue it is accurate only up till the 2 decimal.

ω	Numerical Eigenvalue	Analytical Eigenvalue
$\frac{1}{4}$	1.24999 (ρ_{max} = 10)	1.25
$\frac{1}{20}$	0.352646 (ρ_{max} = 14)	0.35

Table 3: This are the eigenvalues for $V(\rho) = \omega^2\rho^2 + \frac{1}{\rho}$, both numerical (ours, with N =450) and analytical.

Table 3 shows the lowest eigenvalue for $V(\rho) = \omega^2\rho^2 + \frac{1}{\rho}$. We chose ω to be $\frac{1}{4}$ and $\frac{1}{20}$. We decided to use this values so that we could compare them to those found in M.Taut's paper on "Two electrons in an external oscillator potential"[3]

[3]Two electrons in an external oscillator potential: Particular analytic solutions of a Coulomb correlation problem

5 Discussion and analysis

5.1 Unit tests

We chose to do three unit tests to make sure our code worked. Firstly we made sure that the offdiagonal function found the actual max value of the matrix. Our approach was to test a matrix with a known max offdiagonal element and check if the function found it. Perhaps a slightly more automatic approach could be to instead use a max function on a random matrix set with zeros on the diagonal, and compare our results with Armadillos max function for matrices. This would give us more information on how accurate that function is with different matrices, but it raises another issue. Unittests take time, and FLOPS, and so by going with that approach, we would end up with more computation time.

We then checked that the computed eigenvalues and the ones from Armadillos eigsym function had a difference less than a certain tolerance. Alternatively we could have calculated the eigenvalues by hand for a known matrix and compare. But, by using the eigsym function we allow for the matrix to change in values, and even test for larger matrices, which analytically would be hard and timeconsuming to calculate.

And lastly, we checked that the eigenvectors of S were orthogonal. We have a unitary matrix, and so according to the Spectral theorem[1] for Hermitian matrices, our eigenvectors should be orthogonal. We check this for a certain tolerance. This could potentially take a lot of computing power when dealing with larges matrices. Another problem with this is that we might get eigenvectors with very small numbers or very large numbers inside, and so numerically they could be difficult to compute correctly, potentially with round off errors and such. Our test could then very well consider two eigenvectors as not being orthogonal, even though they are.

All the functions have one thing in common, they all test the end result of a function. Alternatively we discussed making more unit-tests that tested the code while it was running. In other words, making sure that our code was making progress. For large matrices this would make sense. If we were to test matrices at size of $10^6 \times 10^6$, the code would take a very long time to compute. If the code didn't make progress, or it was diverging from the correct answer, we would want the code to stop immediately.

[3]More on the Spectral Theorem for Hermitian matrices here.

An example of such a test could be to test if the sum of all non-diagonal elements decreases every thousand iteration. Since we are working with relatively small matrices, we chose not to do this.

5.2 The code

With regards to our code there are two main points to discuss. The speed of execution, and the clear mathematical steps for convergence. The overall positive side of our algorithm is the straight forward math that are used to compute the eigenvalues. Without having had any courses or at least some previous knowledge of numerical analysis, one can with some linear algebra experience understand the methods used.The code is also relatively easy to setup for testing, because it has such a strong mathematical foundation. there are many analytical expressions we can use to test our calculations. This also means that the method is very accurate, from table 1, 2, 3 and figure 1 it is obvious that the results are very strong. Figure 1 shows a slight deviation towards the end, but this could be attributed to the small ρ_{max}-value (1) and therefore relatively small h.

However, the downside of all this is the speed. Timing our Jacobi-method versus the adjacent armadillo function reveals that our solution is as much as 477 times slower for a $400x400$ matrix. We believe this to be mostly due to the poor convergence of the Jacobi-method, since off-diagonal values can be zeroed out and then non-zeroed later, we essentially risk nullifying some of the iterations due to the nature of the algorithm.

5.3 Approximation of infinity

When we extend our code to apply to quantum mechanical systems, we make the change to spherical coordinates. In doing so, our boundary conditions become $\Rightarrow u(\rho_{min} = 0) = 0$ and $u(\rho_{max} = \infty) = 0$. Obviously we cannot set ρ_{max} to infinity, therefore we approximate infinity. The biggest issue in doing so, is that when we increase ρ_{max}, we also increase h (if N is not also increased). When h is increased, the accuracy of the calculation decreases. Therefore we must find a balance where ρ is large enough to be realistic and N isn't to large. We found that already for å matrix of size N=450, our code had to run for 4-5 minutes.

To find the balance between N and ρ we used the 4 known eigenvalues for the system with $(\rho) = \rho^2$. We decided that the approximation be successful when ρ was chosen such that our numerical eigenvalues match the analytical to 3 decimals. Table 2 shows that for $\rho_{max} = 5$ and N = 450 all the first 3 numerical eigenvalues matched to at least 3 decimals. But the last eigenvalue only matched the analytical to 2 decimals. Possibly if we could increase N further, we would reach better accuracy, sadly this will take too long.

6 Conclusion

We have found that using Jacobi's method to solve an eigenvalue problem works well. For a simple buckling beam problem described with a 5×5-matrix we achieved an accuracy of 4 decimals when compared to Armadillos eigsym-function. When we introduced a quantum mechanical element, we varied our definition of ρ_{max} to find known eigenvalues. For $V(\rho) = \rho^2$ we found that with $\rho_{max} = 5$ our method calculated correct eigenvalues up to at least 2 decimals. And lastly we found that for $V(\rho) = \rho^2\omega + \frac{1}{\rho}$ and $\omega = \frac{1}{4}$, the eigenvalue was 1.24999 (analytical=1.25) and for $\omega = \frac{1}{20}$ the eigenvalue was 0.352646 (analytical=0.35)

If long runtime is of no concern, this methods gets the job done. The straight forward mathematical description helps develop an easily made algorithm which can be tested in many ways. However, when dealing with large computations, one usually wants faster algorithms and there are clearly more efficient ways to solve eigenvalue problems. The Armadillo eig_sym function are orders of magnitude faster, and should be preferred when solving such problems.