Project 2- Computational Physics

Vladislav Foss
Simen Husby Norrud
Sebastian Koranda

September 2025

Problem 1

Our 2nd order differential function for this problem is given as:

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

We will now be transforming this into a scaled equation with the following "limitation":

$$\hat{x} \equiv x/L \tag{2}$$
$$\hat{x} \in [0, 1] \tag{3}$$
$$\lambda = F\frac{L^2}{\gamma} \tag{4}$$

We will firstly start off with a variable substitution in our equation with the relationship between $\hat{x}$ and x given in Eq.(2).

From calculus we know that a variable substitution is given as $\frac{d}{dx} = \frac{d\hat{x}}{dx}\frac{d}{d\hat{x}}$. With this premise the variable change will give us:

$$\frac{d}{dx} = \frac{1}{L}\frac{d}{d\hat{x}}$$

meaning that:

$$\frac{d^2}{dx^2} = \frac{d}{dx}(\frac{1}{L}\frac{d}{d\hat{x}^2}) = \frac{1}{L^2}\frac{d}{d\hat{x}}$$

Now substituting into our Eq(1), we get:

$$\gamma \frac{1}{L^2} \frac{d^2u(x)}{d\hat{x}^2} = -Fu(\hat{x})$$
$$\frac{d^2u(x)}{d\hat{x}^2} = -\frac{FL^2}{\gamma} u(\hat{x})$$
$$\frac{d^2u(x)}{d\hat{x}^2} = -\lambda u(\hat{x})$$

Thereby we have shown how you can write the original function given in Eq(1) into $\frac{d^2u(x)}{d\hat{x}^2} = -\lambda u(\hat{x})$.

Problem 2

We first computed the analytical eigenvalues and eigenvectors, and then verified our implementation by constructing the tridiagonal matrix and comparing its eigenvalues obtained with Armadillo's `arma::eig_sym` function to the analytical results. Table 1 compares the eigenvalues of the computed and analytical solutions. Table 2 presents the maximum differences between each of the Armadillo and analytic eigenvectors (allowing for sign flips), which are also on the order of 10^{-16}, confirming that Armadillo's solution agrees with our analytical solution.

Mode j	Armadillo λ_j	Analytic λ_j	$\lvert\lambda_j^{\text{Arma}} - \lambda_j^{\text{Analytic}}\rvert$
1	9.7051×10^0	9.7051×10^0	3.5527×10^{-15}
2	3.6898×10^1	3.6898×10^1	2.1316×10^{-14}
3	7.6193×10^1	7.6193×10^1	1.4211×10^{-14}
4	1.1981×10^2	1.1981×10^2	2.8422×10^{-14}
5	1.5910×10^2	1.5910×10^2	0
6	1.8629×10^2	1.8629×10^2	5.6843×10^{-14}

Table 1: Difference of eigenvalues between Armadillo and analytic solution.

Mode j	Max difference
1	2.22045×10^{-16}
2	3.33067×10^{-16}
3	5.82867×10^{-16}
4	3.88578×10^{-16}
5	7.77156×10^{-16}
6	5.27356×10^{-16}

Table 2: Maximum difference between Armadillo and analytic eigenvectors.

Problem 3

We implemented the function `double max_offdiag_symmetric(const arma::mat& A, int& k, int& l)` which, given a symmetric matrix A, loops over the upper triangular part (since for symmetric matrices the lower part is identical) and finds the element with the largest absolute value. The function also updates the indices k and l to point to the location of this element and returns its value.

We test the function with the matrix:

$$A = \begin{bmatrix} 1.0 & 0.0 & 0.0 & 0.5 \\ 0.0 & 1.0 & -0.7 & 0.0 \\ 0.0 & -0.7 & 1.0 & 0.0 \\ 0.5 & 0.0 & 0.0 & 1.0 \end{bmatrix}$$

And correctly returns -0.7, at row 2, column 3.

Problem 4

We want to solve the eigenvalue problem $A\vec{v} = \lambda\vec{v}$, where A is going to be a tri-diagonal matrix.

If we reorder the equation we get:

$$\mathbf{A}\vec{x} = \lambda\vec{x} \tag{5}$$

$$(\mathbf{A} - \lambda\mathbf{I})x = 0 \tag{6}$$

The equation $\det(\mathbf{A} - \lambda\mathbf{I})$ will then be a polynomial of degree N. The problem arises when N gets very large and thereby becomes very inefficient to compute. Therefore we introduce **Jacobi's rotation Method**.

The motivation for Jacobi's rotation method is that if $\mathbf{A}$ can be transformed into a diagonal form, its diagonal entries are the eigenvalues, and the corresponding eigenvectors are given by the columns of the transformation matrix. To achieve this we apply an orthogonal rotation matrix $\mathbf{R}$ such that

$$\mathbf{R}^T\mathbf{R} = \mathbf{I}, \qquad \mathbf{A}' = \mathbf{R}^T\mathbf{A}\mathbf{R}.$$

Orthogonal transformations preserve eigenvalues and keep the matrix symmetric. At each step, $\mathbf{R}$ is chosen to eliminate the largest off-diagonal element in $\mathbf{A}$. By repeating this process, the matrix becomes increasingly diagonal, until all off-diagonal terms are below a given tolerance. The diagonal elements then approximate the eigenvalues, and the accumulated product of rotations $\mathbf{R}$ contains the eigenvectors.

In our code we have a tolerance of $tol = 1e^{-15}$. The function `max_offdiag_symmetric` scans through the upper triangular part of the symmetric matrix $\mathbf{A}$.

It returns the indices (k, l) of the largest off-diagonal element in absolute value, as well as its magnitude. This step identifies the element that contributes most to the "non-diagonality" of the matrix and is therefore the best candidate to eliminate in the next rotation.

The function `jacobi_rotate` eliminates the chosen off-diagonal element A_{kl}. Given the indices (k, l) from the previous step, it computes the rotation parameters

$$\tau = \frac{A_{ll} - A_{kk}}{2A_{kl}}, \qquad t = -\tau \pm \sqrt{1+\tau^2}, \qquad c_\theta = \frac{1}{\sqrt{1+t^2}}, \qquad s = c_\theta t_\theta,$$

and applies the orthogonal similarity transformation

$$\mathbf{A}' = \mathbf{R}_{kl}^T\mathbf{A}\mathbf{R}_{kl}.$$

This sets $A'_{kl} = A'_{lk} = 0$, updates the diagonal entries A_{kk} and A_{ll}, and mixes the remaining elements in rows/columns k and l. At the same time, the eigenvector matrix $\mathbf{R}$ is updated with the same rotation so that its columns converge to the eigenvectors.

Problem 5

Scaling behaviour

For the tridiagonal matrix of A, most elements will be zero and Jacobi's method only needs to eliminate the non-zero elements off the diagonal. In this case, the number of required similarity transformations scales somewhat like $\mathcal{O}(N^2)$.

If $\mathbf{A}$ is instead a dense symmetric matrix, then there are $\frac{N(N-1)}{2}$ off-diagonal elements that must be reduced. Each Jacobi rotation eliminates only one element at a time, while simultaneously altering an entire row and column. As a result, the number of required transformations scales more poorly, on the order of $\mathcal{O}(N^3)$, and since each rotation requires $\mathcal{O}(N)$ work, the total computational complexity of Jacobi's method for a dense matrix is $\mathcal{O}(N^4)$.

For a dense symmetric matrix $\mathbf{A}$, the number of required Jacobi rotations still scales approximately like

$$\mathcal{O}(N^2).$$

This is because one sweep involves applying a rotation for each of the

$$\frac{N(N-1)}{2} = \mathcal{O}(N^2)$$

off-diagonal elements. In practice, convergence is usually reached in only a few sweeps, so the total number of similarity transformations grows quadratically with N, similar to the tridiagonal case.

However, there is a crucial difference in terms of computational cost. Each Jacobi rotation in a dense matrix requires updating an entire row and column of $\mathbf{A}$, which is an

$$\mathcal{O}(N)$$

operation. Thus, while the number of transformations is $\mathcal{O}(N^2)$, the *total computational cost* is

$$\mathcal{O}(N^2) \times \mathcal{O}(N) = \mathcal{O}(N^3).$$

This scaling matches that of more advanced algorithms (such as the QR method), but with a much larger constant factor, which makes Jacobi inefficient for large dense matrices.

Table 3 and Figure 1 compares the number of Jacobi rotations required for convergence between the sparse tridiagonal case and a dense random symmetric matrix of the same size. The results illustrate that the number of rotations is very similar in both cases, since Jacobi is not sparsity-preserving: each rotation introduces additional nonzero elements, and the matrix quickly behaves like a dense one.

N	Jacobi rotations (Sparse)	Jacobi rotations (Dense)
6	42	58
10	168	179
20	758	795
40	3288	3265
100	20778	20620
200	83733	83025

Table 3: Number of Jacobi rotations required for convergence for sparse (tridiagonal) vs dense symmetric matrices.

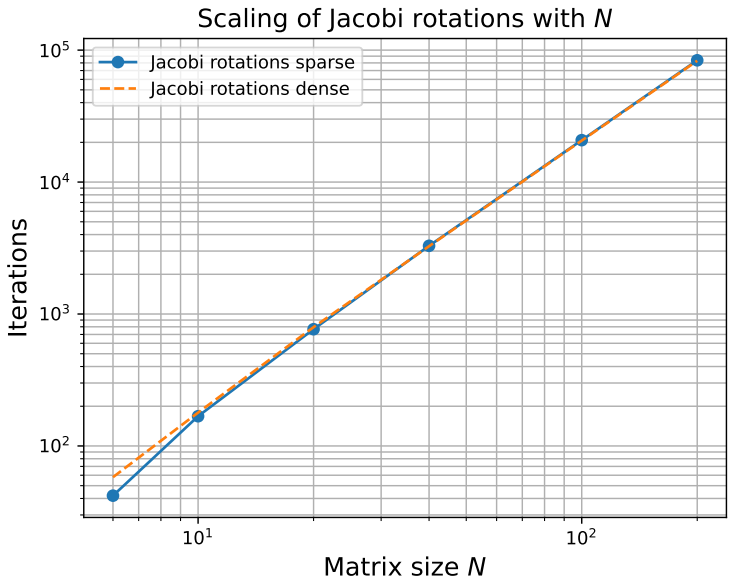

Figure 1: Number of Jacobi rotations for sparse and dense matrix

Problem 6

Figure 2 shows the three lowest eigenvectors obtained using our Jacobi algorithm for a discretization with n = 10 steps, together with the corresponding analytical eigenvectors (including the boundary points). We see that the numerical and analytical eigenvectors overlap almost perfectly, confirming the accuracy of our implementation for small n.

Figure 3 presents the same comparison for n= 100 steps. Again, the

numerical eigenvectors follow the analytical ones closely, and the higher resolution gives smoother curves. This demonstrates that our method converges to the analytical solutions as n increases.

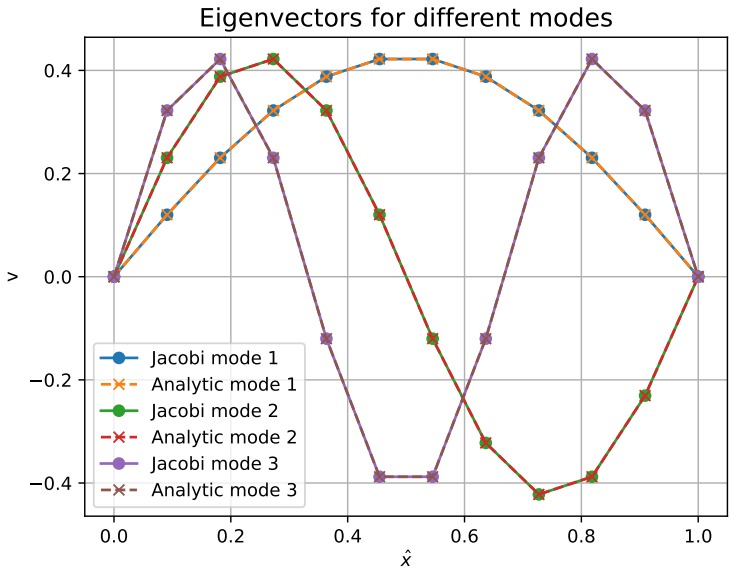

Figure 2: Eigenvectors corresponding to the three lowest eigenvalues for N = 10. Analytic and computed values.

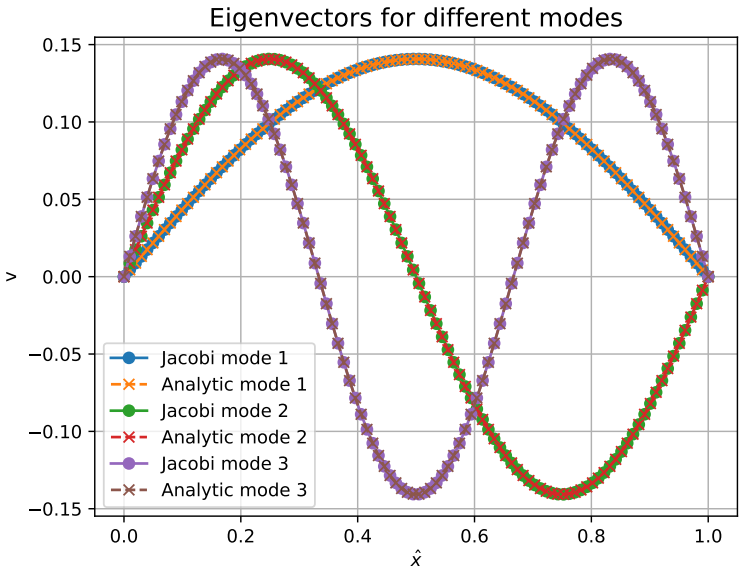

Figure 3: Eigenvectors corresponding to the three lowest eigenvalues for N = 100. Analytic and computed values.

A Source code

GitHub repository with code: `https://github.uio.no/sebasmk/fys4150.git`