Project 2

Alessio Canclini, Filip von der Lippe
(Dated: October 8, 2022)

Github repository: https://github.com/Fslippe/FYS4150/tree/main/project2

INTRODUCTION

The main purpose of this project is to familiarize ourselves with the topics of equation scaling, code testing and numerical solutions to eigenvalue problems. The eigenvalue problem we will be working with is a special case of a one-dimensional buckling beam.
Problem 1 demonstrates how the initial second order differential equation is scaled. Problems 2 through 4 focus on writing and testing the Jacobi rotation algorithm which will be used to solve the eigenvalue problem. The code can be found in the Github repository linked above. Problem 5 looks at the algorithm's scaling behavior, specifically how the the number of similarity transformations is related to the size N of the input matrix. Finally in problem 6 the constructed code is used to solve the eigenvalue problem and make a plot of the three eigenvectors corresponding to the three lowest eigenvalues, visualizing the solutions to the differential equation for the buckling beam.

We are working with:

- A horizontal beam of length L.
- Let $u(x)$ be the vertical displacement of the beam at horizontal position x, with $x \in [0, L]$.
- A force F is applied at the endpoint ($x = L$), directed into the beam, i.e. towards $x = 0$.
- The beam is fastened with pin endpoints, meaning that $u(0) = 0$ and $u(L) = 0$, but the endpoints are allowed to rotate ($u'(x) \neq 0$).

Second order differential equation describing the buckling beam situation:

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

Throughout this project we will be working with the scaled equation:

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

Where $\hat{x} \equiv x/L$ is a dimensionless variable, $\hat{x} \in [0, 1]$ and $\lambda = \frac{FL^2}{\gamma}$.

PROBLEM 1

Using the definition $\hat{x} \equiv x/L$ to show that Eq. 1 can be written as Eq. 2.
First applying the chain rule on the differential operator $\frac{d}{dx}$. This gives us:

$$\frac{d^2}{dx^2} = \frac{d^2}{d\hat{x}^2}\frac{d\hat{x}^2}{dx^2}$$

Then using that $\left(\frac{\hat{x}}{x}\right)^2 = \frac{1}{L^2}$.

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

Putting this into Eq. 1 we have:

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

Then dividing both sides by γ and multiplying by L^2.

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

Inserting $x = \hat{x}L$ we can rename $u(\hat{x}L)$ to $\hat{u}(\hat{x})$ and again rename this $u(\hat{x})$.

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

This finally gives us Eq. 2:

$$\frac{d^2u(\hat{x})}{d\hat{x}^2} = -\lambda u(\hat{x})$$

Where $\lambda = \frac{FL^2}{\gamma}$.

PROBLEM 2

Before implementing the Jacobi rotation algorithm we write a short program that creates a tridiagonal matrix $\mathbf{A}$ of size N. The eigenvalue equation $\mathbf{A}\vec{v} = \lambda\vec{v}$ is the solved using Armadillo's `arma::eig_sym`. This program is the tested for $N = 6$ and the Armadillo eigenvalues are compared to the analytical solution. Having confirmed that these helper functions work we can move on to the next step of the project.

PROBLEM 3

a) To implement the Jacobi rotation algorithm we need a function that can identify the larges (in absolute value) off-diagonal element of a matrix. We therefore write a suitable `max_offdiag_symetric` function in C++ (see "functions.cpp" in GitHub repo).

b) To ensure that this section of the code works we write a test code that runs the `max_offdiag_symetric` function for the matrix:

$$A = \begin{bmatrix} 1 & 0 & 0 & 0.5 \\ 0 & 1 & -0.7 & 0 \\ 0 & -0.7 & 1 & 0 \\ 0.5 & 0 & 0 & 1 \end{bmatrix}$$

Returning the expected result of 0.7 as the largest (in absolute value) off-diagonal element of a matrix confirms that the function works as it should.

PROBLEM 4

a) Having working functions that execute individual steps of the total algorithm we can now write e full code implementation of Jacobi's rotation algorithm to solve the eigenvalue equtaion $\mathbf{A}\vec{v} = \lambda\vec{v}$. We first write a `jacobi_rotate` function that executes one single Jacobi rotation. This and the previous `max_offdiag_symetric` function are then used to create the final `jacobi_eigensolver`.

b) Letting $\mathbf{A} = 6 \times 6$ we run the `jacobi_eigensolver` function and run it through an assertion test to confirm that eigenvalues and eigenvectors produced by `jacobi_eigensolver` match the aanalytical results for $N = 6$.

PROBLEM 5

We will now look at how many similarity transformations are needed before a result where all non-diagonal matrix elements are close to zero is reached.

Running the program for different choices of N gives us the following scaling data in figure 1. We observe that the straight line in our log-log plot in figure 1 has a slope of 2. This leads us to the conclusion that the number of required transformations T are square proportional to the matrix size N. So $T = N^2$.

Figure 1 demonstrates that the program shows close to the same scaling behavior for both tridiagonal and dense matrices. This is as expected, since the tridiagonal matrix is no longer tridiagonal after the first Jacobi rotation of the algorithm. Therefore the tridiagonal matrix is treated just like a dense matrix after a few rotations in this case.

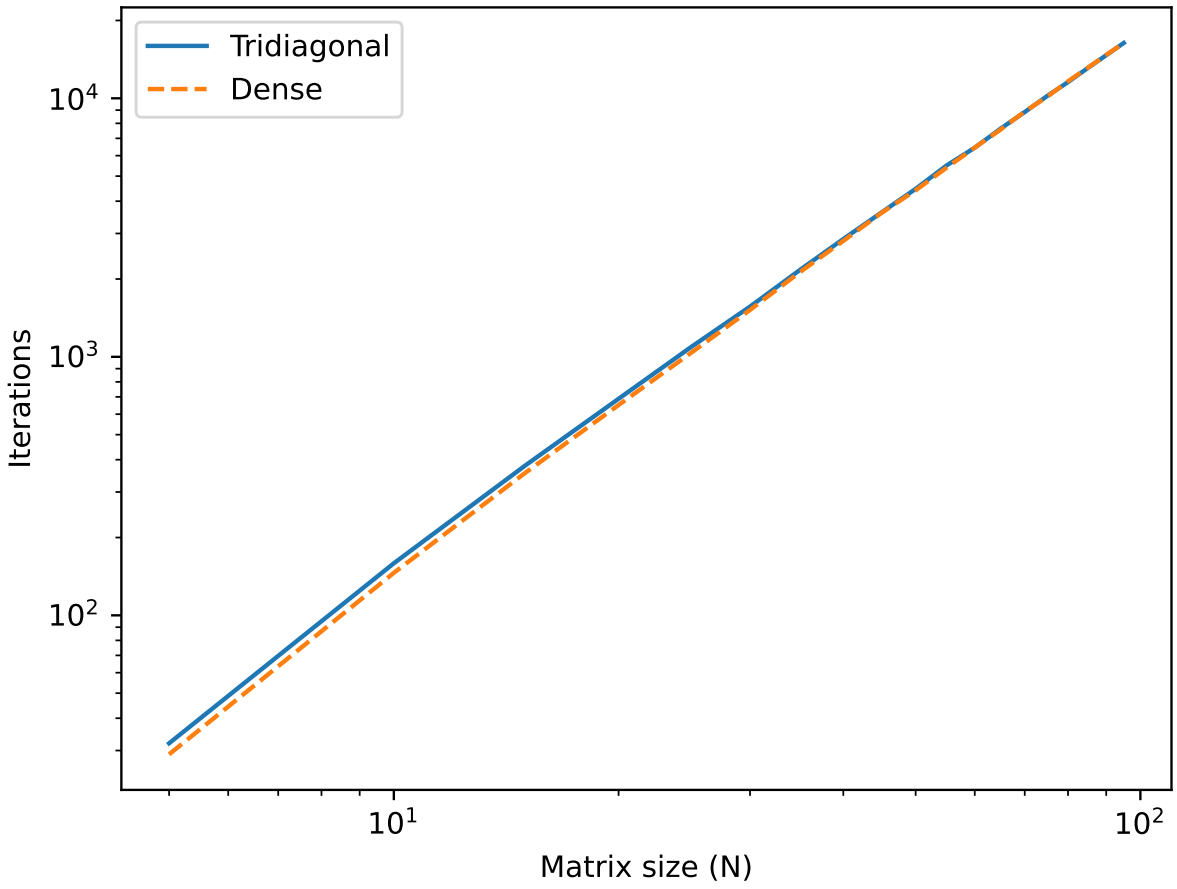

Figure 1: The plot shows transformation scaling data for different matrix sizes on a log-log scale. The blue line represents the Jacobi rotation algorithms' behavior for a tridiagonal matrix, whereas the orange line represents the Jacobi rotation algorithms' behavior for a dense matrix.

PROBLEM 6

a) Solving the equation $\mathbf{A}\tilde{\mathbf{v}} = \lambda\tilde{\mathbf{v}}$ for a discretization of $\hat{x}$ with steps $n = 10$, steps gives us the three eigenvectors in figure 2

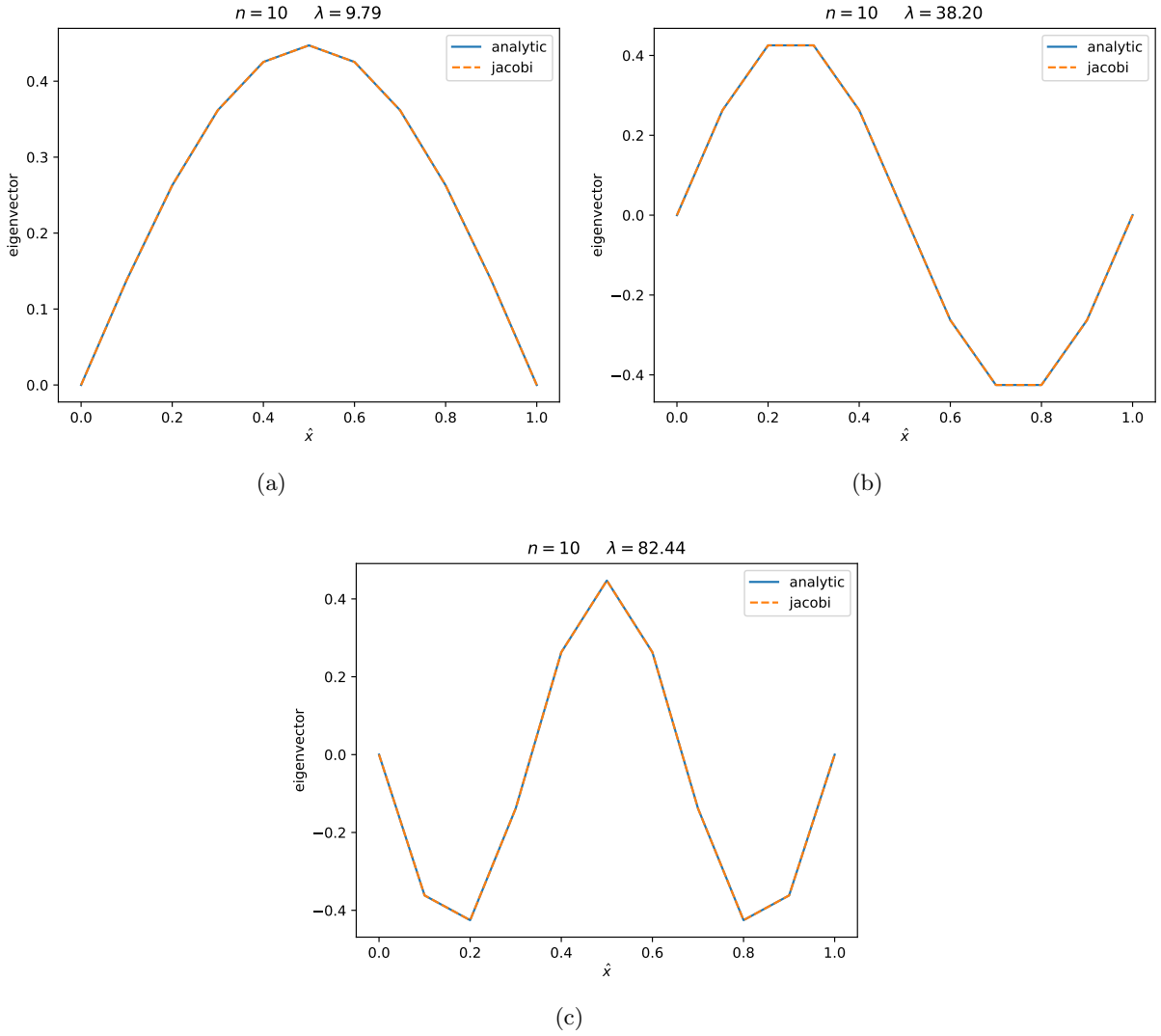

Figure 2: Eigenvectors corresponding to the three lowest eigenvalues for $n = 10$. Plots show the vector v_i elements against the corresponding positions

b) Figure 3 shows the same plots for discretization of with $n = 100$ steps.

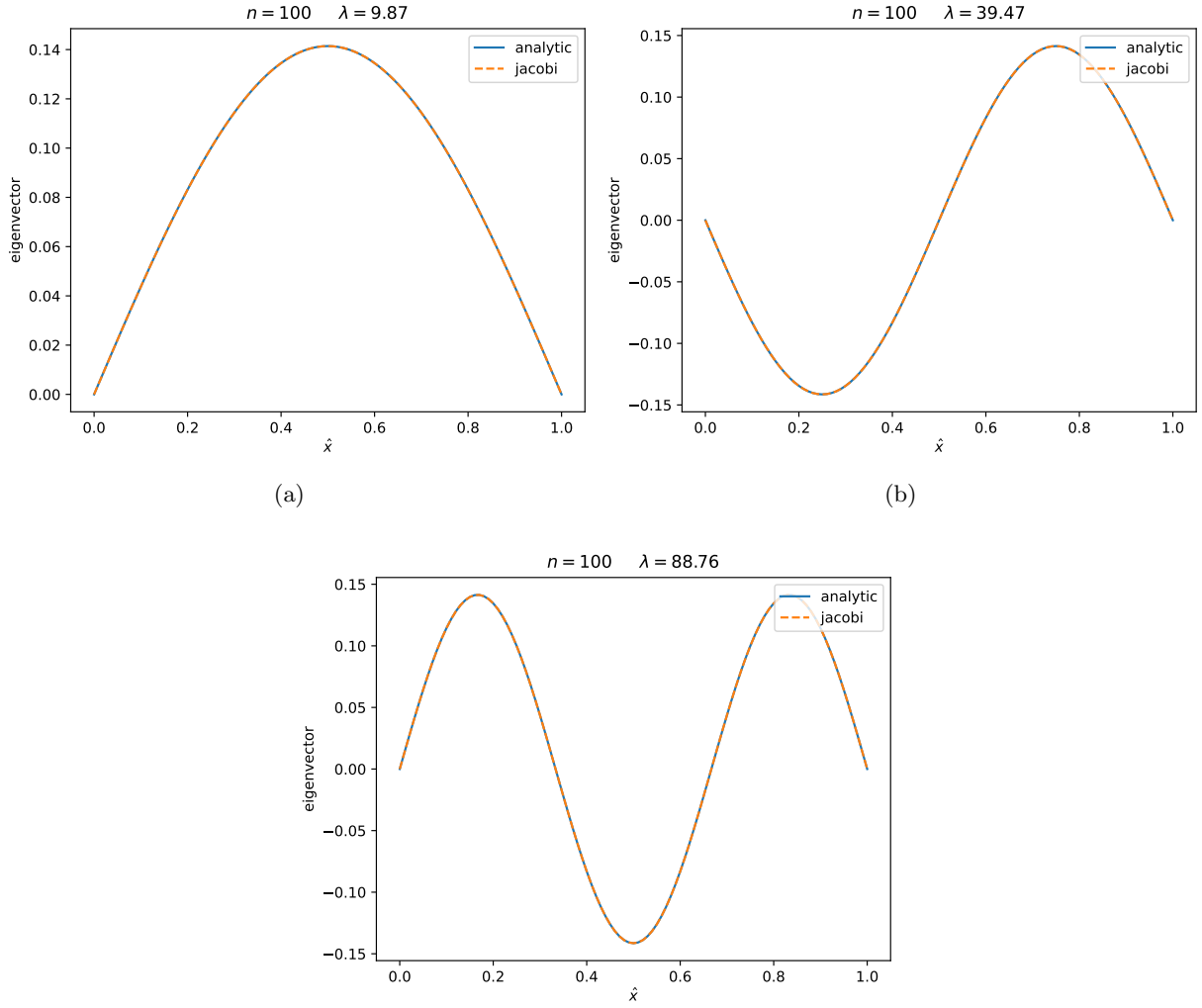

Figure 3: Eigenvectors corresponding to the three lowest eigenvalues for $n = 100$.