

Solutions to Project 2

Lars Bogner
(Dated: October 8, 2025)

Code hosted at <https://github.com/larsbog/FYS4150>

PROBLEM 1

Using the rescaling of the positions $\hat{x} \equiv \frac{x}{L}$ we define the following relations:

$$u_x \equiv u, \quad u_{\hat{x}} \equiv \hat{u}, \quad \text{with} \quad u(x) = \hat{u}(\hat{x}) = u(L\hat{x}). \quad (1)$$

Now we can rewrite the equation as

$$\begin{aligned} \gamma \frac{d^2 u(x)}{dx^2} &= -Fu(x) \\ \gamma \frac{d^2 u(L\hat{x})}{d(L\hat{x})^2} &= -Fu(L\hat{x}). \end{aligned} \quad (2)$$

In this step we simply replaced x by the equivalent expression $L\hat{x}$. Now we can use the chain rule to rewrite the left hand side:

$$\gamma \frac{\partial^2 u(L\hat{x})}{\partial \hat{x}^2} \underbrace{\frac{\partial^2 \hat{x}}{\partial (L\hat{x})^2}}_{=L^{-2}} = -Fu(L\hat{x}). \quad (3)$$

Rearranging the factors gives the final result:

$$\frac{d^2 \hat{u}(\hat{x})}{d\hat{x}^2} = -\frac{FL^2}{\gamma} \hat{u}(\hat{x}) \equiv -\lambda \hat{u}(\hat{x}). \quad (4)$$

PROBLEM 2

The code is implemented in `src/project2/arma_eigenproblem_checker.cpp`. The reusable functions for the tridiagonal matrix, the analytical solution and a wrapper around the Armadillo eigenvalue solver are implemented in `./src/helpers.cpp`.

PROBLEM 3

Subproblem (a)

The function with the signature `double max_offdiag_symmetric(const arma::mat& A, int& k, int& l)` finds the largest off-diagonal element in the symmetric matrix A and returns its value. The algorithm works like described in Algorithm 1. The code is implemented in `./src/helpers.cpp`.

Subproblem (b)

The code correctly identifies the largest off-diagonal element as 0.7 at position (1,2). The function call is tested in `src/project2/off_diagonal_finder.cpp`.

Algorithm 1 Finding the largest off-diagonal element in a symmetric matrix

```

function MAX_OFFDIAG_SYMMETRIC( $A, k, l$ )
   $n \leftarrow$  number of rows of  $A$ 
   $\max \leftarrow 0$ 
  for  $i = 0$  to  $n - 1$  do
    for  $j = i + 1$  to  $n - 1$  do
      if  $|A_{ij}| > \max$  then
         $\max \leftarrow |A_{ij}|$ 
         $k \leftarrow i$ 
         $l \leftarrow j$ 
  return  $\max$ 

```

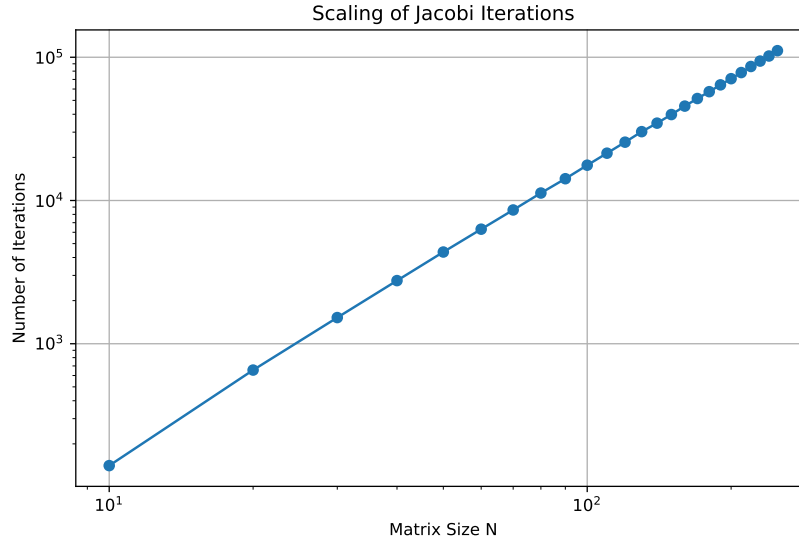


FIG. 1. Scaling of the number of iterations with the matrix size N .

PROBLEM 4

The Jacobi rotation algorithm is implemented as described in the lecture notes. The implementation is found in `src/helpers.cpp` as the function `void jacobi_eigensolver(const arma::mat& A, double eps, arma::vec& eigenvalues, arma::mat& eigenvectors, const int maxiter, int& iterations, bool& converged)`. It uses the `jacobi_rotate` function to perform the actual rotation. The algorithm is tested in `src/project2/arma_eigenproblem_checker.cpp`. It modifies the eigenvalues and eigenvectors passed by reference. The function also returns the number of iterations used and whether the algorithm converged within the maximum number of iterations by modifying the variables passed by reference.

PROBLEM 5

Subproblem (a)

The scaling of the number of iterations is tested in `src/project2/scaling_tester.cpp`. The results are shown in [Figure 1](#). On the double logarithmic scale, the scaling is approximately linear, indicating a polynomial relation between the number of iterations and the matrix size N .

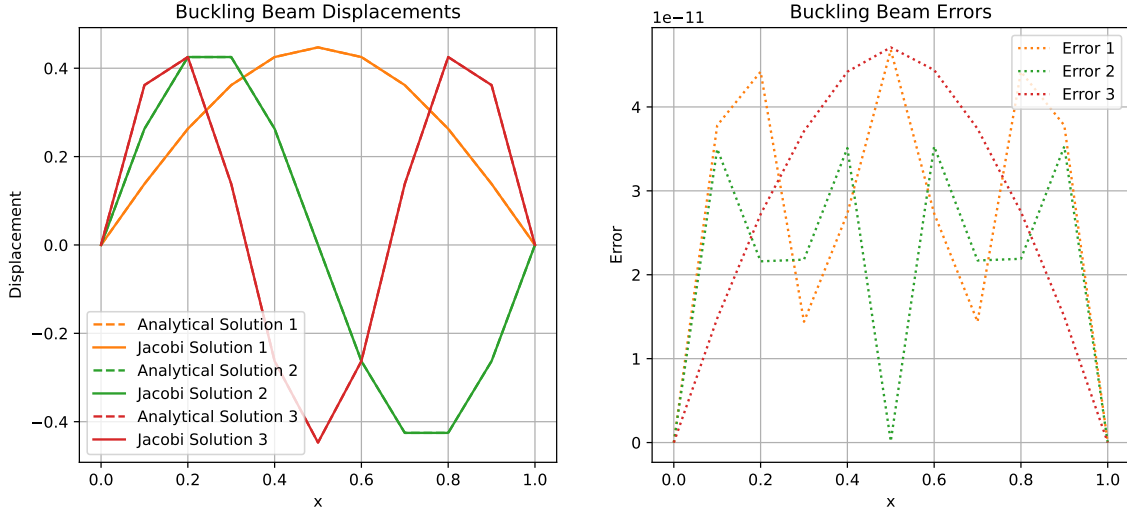


FIG. 2. Comparison of the numerical and analytical solution of the buckling beam problem for $N = 10$.

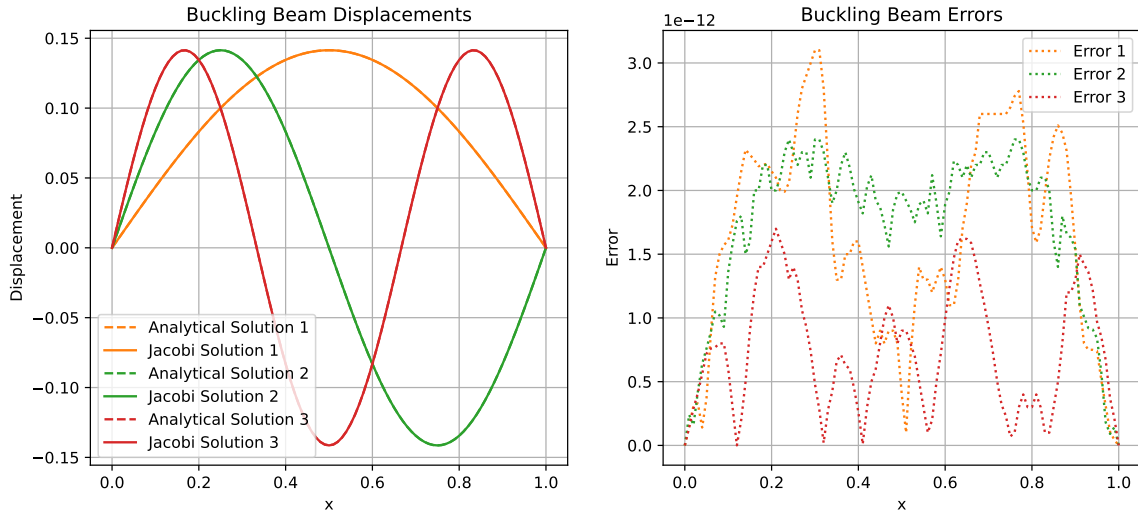


FIG. 3. Comparison of the numerical and analytical solution of the buckling beam problem for $N = 100$.

Subproblem (b)

The scaling is also tested for a dense matrix, i.e. a matrix without the tridiagonal structure. The results show nearly no dependence on the density of the matrix. This is to be expected, since the Jacobi algorithm creates new non-zero elements in each iteration, quickly filling up the matrix before converging to the solution.

PROBLEM 6

The numerical and analytical solution of the buckling beam problem is compared in [Figure 2](#) for $N = 10$ and in [Figure 3](#) for $N = 100$. The numerical solution is in very good agreement with the analytical solution, even for the small value of $N = 10$. For $N = 100$, the solutions are visually indistinguishable. Nevertheless, the absolute error between the solutions is displayed on the right-hand side of the figures. The code to generate the plots is found in `src/project2/python/buckling_beam_plotter.py`.