FYS4150 - Project 2

Halvor Melkild
(Dated: September 28, 2022)

GitHub repo: https://github.com/halvorme/FYS4150

In this project we will study a static buckling beam. The system can be modeled as a second order differential equation. To solve the problem numerically, we will reframe it as an eigenvalue problem.

The system we study is a beam of length L, with fixed endpoints. A force F is applied in the direction of the beam at one endpoint. The buckling of the beam is modeled by the function $u(x)$, which tells the displacement of the beam at the point $x \in [0, L]$. the function $u(x)$ is a solution of

$$\gamma \frac{\mathrm{d}^2 u(x)}{\mathrm{d}x^2} = -F u(x), \tag{1}$$

where γ is a material dependent constant.

PROBLEM 1

Before we try to to solve Equation 1, we want to make the variable x dimensionless. The natural length scale of the system is the length of the beam, L. We therefore introduce the dimensionless variable

$$\hat{x} = \frac{x}{L}, \qquad \hat{x} \in [0, 1]. \tag{2}$$

By doing a change of variable we find that Equation 1 becomes

$$\frac{\gamma}{L^2} \frac{\mathrm{d}^2 u(\hat{x})}{\mathrm{d}\hat{x}^2} = -F u(\hat{x}). \tag{3}$$

Whn we define the dimensionless variable $\lambda = FL^2/\gamma$, this equation can be rewritten as

$$\frac{\mathrm{d}^2 u(\hat{x})}{\mathrm{d}\hat{x}^2} = -\lambda u(\hat{x}). \tag{4}$$

PROBLEM 2

The problem is solved in the function `problem2()` in the file `tasks.cpp`. When the full program for the project is run, the results from this problem will be written to terminal.

As the machine precision is limited, an exact comparision of the analytic and numeric solutions will never be equal. We therefore do only an approximate comparison of the solutions, using Armadillos in-built function `approx_equal()`. We have set the tolerance level for the differece in each element to 10^{-9}.

PROBLEM 3

The algorithm for finding the entry with the maximal absolute value is found in the file `utils.cpp`. The test is implemented in the function `problem3` and the results are written to the terminal.

PROBLEM 4

The algorithm for Jacobi's method is implemented in the functions `jacobi_eigensolver()` and `jacobi_rotate()` in `algo.cpp`. The test is run by the function `problem4()` in `tasks.cpp`.

It is possible to speed up the algorithm we have used, by using the fact that A is symmetric. One can work with only the upper triangle of the matrix and not bother with updating the lower half.

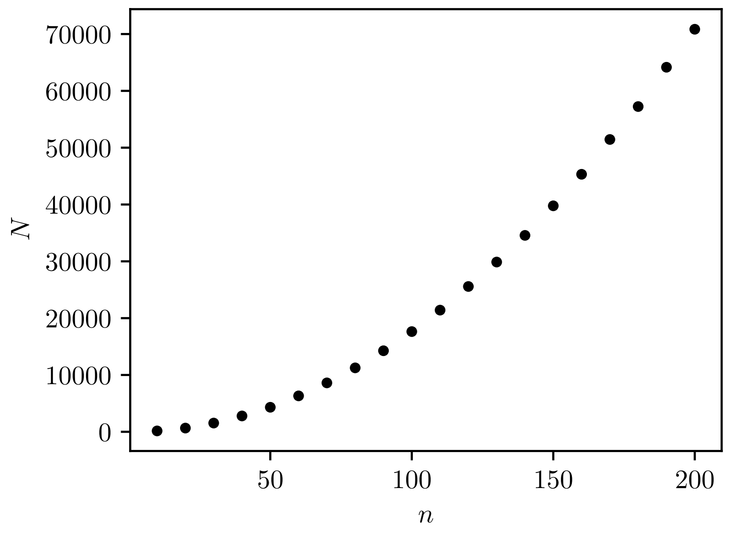

FIG. 1. The plot shows the exact solution to the Poisson equation, with the source term $f(x) = 100\mathrm{e}^{-10x}$, and boundary conditions $u(0) = u(1) = 0$.

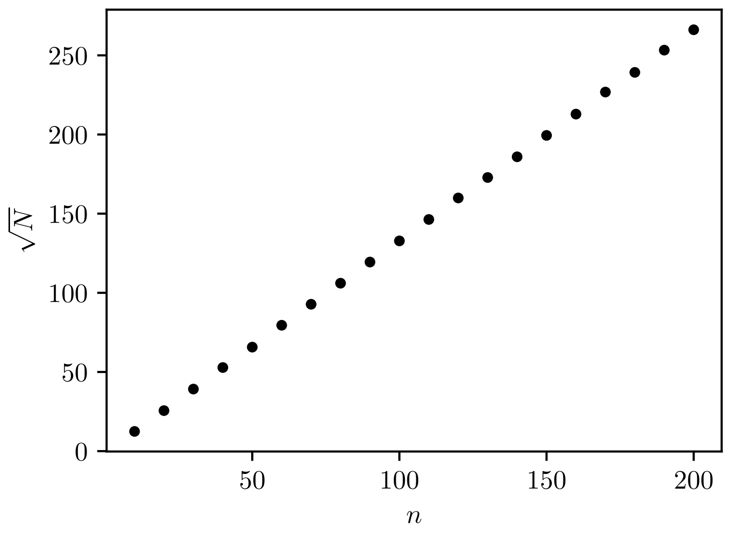

FIG. 2. The plot shows the exact solution to the Poisson equation, with the source term $f(x) = 100\mathrm{e}^{-10x}$, and boundary conditions $u(0) = u(1) = 0$.

PROBLEM 5

In Figure 1 we present the number of iterations needed for Jacobi's method to converge (N), as a function of the size of the tridiagonal matrix A. n denotes the number of rows in A. By plotting the square root of N in Figure 2, we see that the relation seems more or less linear. Our conclusion is that the number of iterations grows as

$$N \sim \mathcal{O}(n^2) \tag{5}$$

for a tridiagonal matrix.

The fact that matrix entries that are already zero might become non-zero when we rotate away another entry,

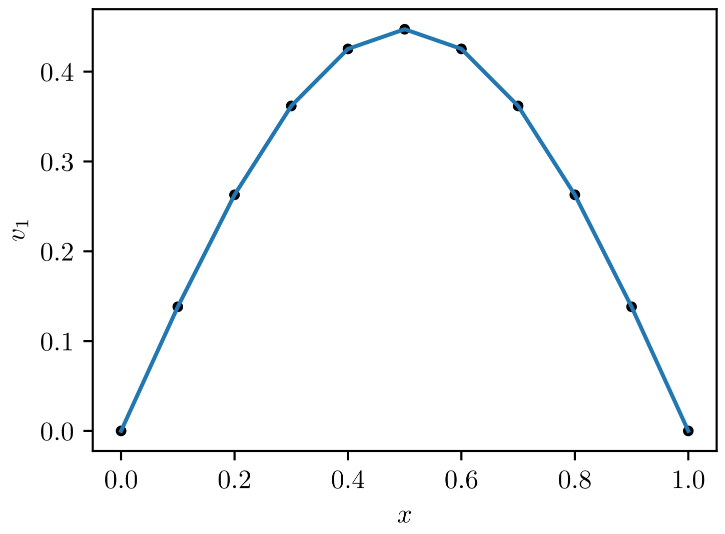

FIG. 3. First mode

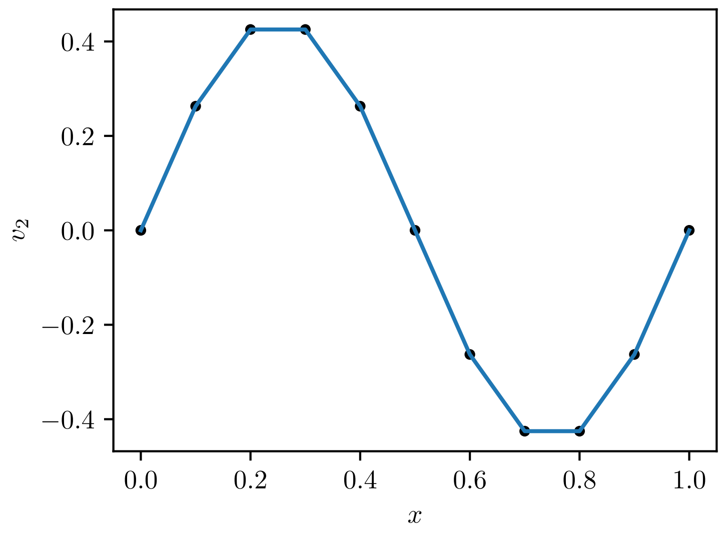

FIG. 4. Second mode

means that even though A is initially tridiagonal we have to work with the whole matrix. We would therefore expect that the algorithm has similar efficiency for a dense symmetric matrices.

PROBLEM 6

For $n = 10$ the eigenvectors corresponding to the three lowest eigenvalues are shown in Figures 3-5. For $n = 100$ they are shown in Figures 6-8. The blue lines indicate the analytic solutions and the points indicate the numerical ones.

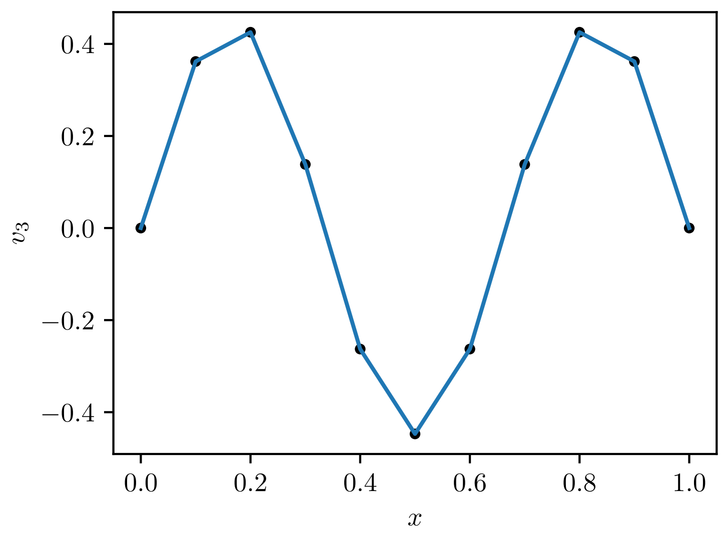

FIG. 5. Third mode

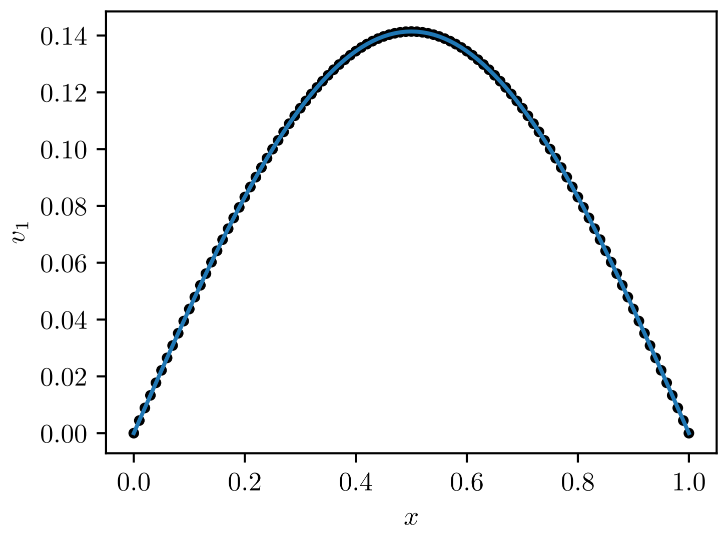

FIG. 6. First mode

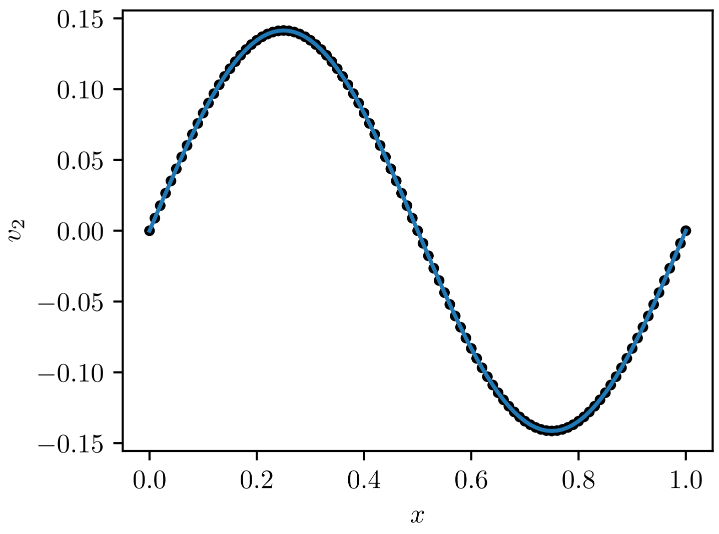

FIG. 7. Second mode

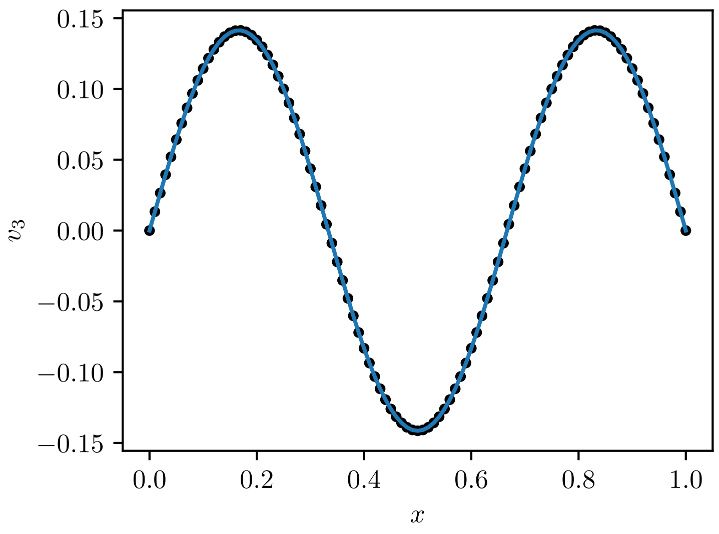

FIG. 8. Third mode