# FYS3150 Project 2

Carl Fredrik Nordbø Knutsen & Didrik Sten Ingebrigtsen
(Dated: September 27, 2021)

*https://github.com/carlfre/FYS3150-Project-2*

## PROBLEM 1

Firstly, we want to show that given $\hat{x} \equiv x/L$ and $\lambda = \frac{FL^2}{\gamma}$, this equation

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

can be written as

$$\frac{d^2 u(\hat{x})}{d\hat{x}^2} = -\lambda u(\hat{x}) \tag{1}$$

Let us start by replacing $x$ with $\hat{x}$. Since $\frac{d}{dx} = \frac{d\hat{x}}{dx}\frac{d}{d\ \hat{x}} = \frac{1}{L}\frac{d}{d\hat{x}}$, we get

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

If we now insert $F = \frac{\lambda\gamma}{L^2}$, we see that our final expression becomes

$$\begin{aligned} \gamma \frac{1}{L^2}\frac{d^2 u(\hat{x})}{d\hat{x}^2} &= -\frac{\lambda\gamma}{L^2}u(x) \\ \frac{d^2 u(\hat{x})}{d\hat{x}^2} &= -\lambda u(\hat{x}), \end{aligned}$$

which is the same as (1), as we wanted to show.

## PROBLEM 2

Next, we want to show an important property of orthogonal transformations. Let us assume that we have a set of vectors $v_i$ that is orthonormal, in other words $v_i^T \cdot v_j = \delta_{ij}$, and that $U$ is an orthogonal matrix, i.e. that $U^T = U^{-1}$. Now, we want to show that the set of vectors $w_i = Uv_i$ is also orthornormal. Saying that it is orthonormal is the same as saying that $w_i^T \cdot w_j = \delta_{ij}$. We insert our definition of $w_i$ into this, and get

$$(Uv_i)^T \cdot (Uv_j) = v_i^T U^T U v_j = v_i^T U^{-1} U v_j = v_i^T \cdot v_j$$

This, we know is equal to $\delta_{ij}$, and thus we have proved the property.

## PROBLEM 3

The code used to test this is in include/test.hpp and src/test.cpp, and can be run by running make test. When done, it is evident that the analytical and Armadillo's eig_sym give the same results, at least to a couple of decimals. In this test, we also used our Jacobi's rotation implementation implemented as a part of problem 5, and compared the results it gives.

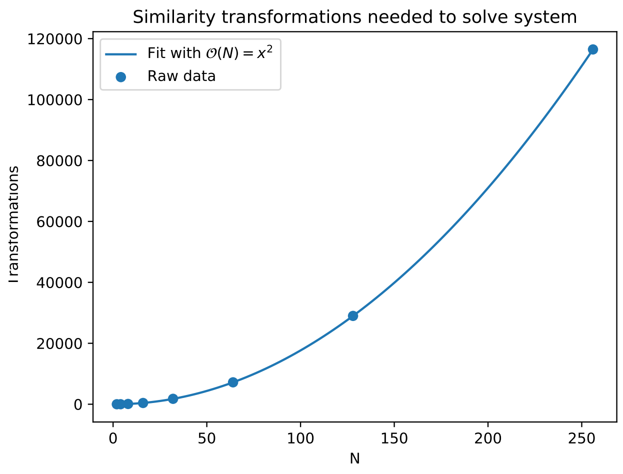

FIG. 1. The number of similarity transformations required for different sizes of matrices, and a second order polynomial fit to the empirical data.

## PROBLEM 4

### a)

The max_offdiag_symmetric function can be found in include/matrix_operations.hpp and src/matrix_operations.cpp.

### b)

Just like in problem 3, the code used to test this is in include/test.hpp and src/test.cpp, and can be run by running make test. However, we used assertions to verify that everything works as it should, instead of printing it out and manually interpreting the results. If we had done everything again, we would have used this approach on problem 3 as well.

## PROBLEM 5

The code used to implement Jacobi's rotation algorithm is in the function jacobi_rotate in include/matrix_operations.hpp and src/matrix_operations.cpp. The tests discussed in part b of this problem was adressed above here, in problem 3.

## PROBLEM 6

### a)

To find the number of transformations required, we use our previously implemented jacobi_rotate function, and write the number of iterations used to file. We then use in src/plot.py to plot it, and the results can be seen in figure 1. If we fit a polynomial of degree 2 to the data, we can see it fits perfectly.

### b)

The Jacobi's rotation algorithm doesn't take advantage of the fact that the matrices in our problem are tridiagonal. After a couple of iterations on a large matrix, it will no longer be tridiagonal. This makes it reasonable to expect that it won't perform that much worse on a dense matrix than on a tridiagonal matrix like here.

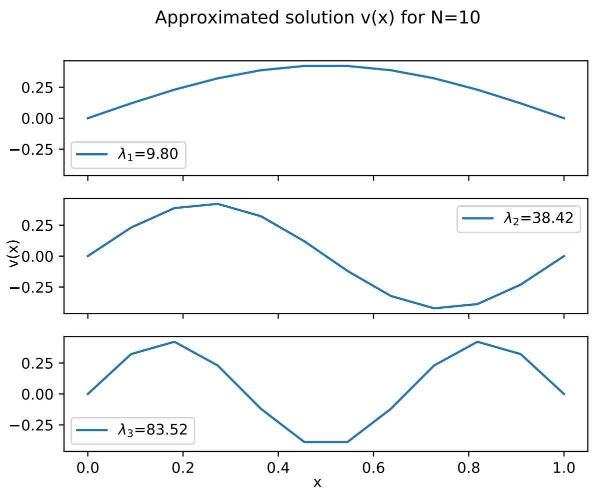

FIG. 2. Solutions for $N = 10$. The three different subplots show the normalised eigenvectors for different eigenvalues. This can be interpreted as the deflection of the beam under different ratios of force, quadratic length, and material properties.

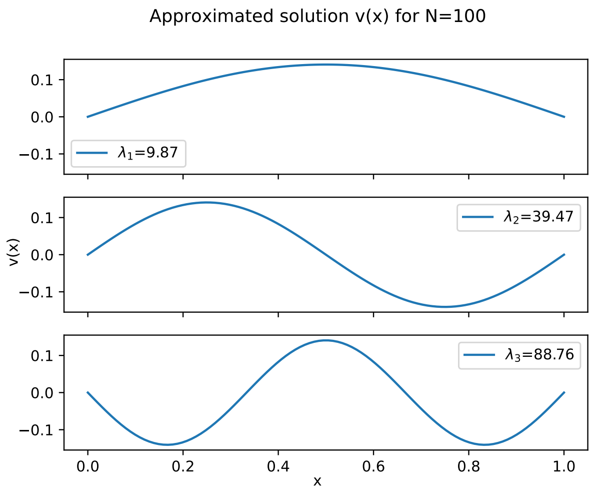

FIG. 3. Solutions for $N = 100$. The three different subplots show the normalised eigenvectors for different eigenvalues. This can be interpreted as the deflection of the beam under different ratios of force, quadratic length, and material properties.

## PROBLEM 7

To solve this problem, we use all the components implemented, and assemble it in main.cpp. We also have some new plotting code in src/plot.py for this problem. Together, they make fig 2 and 3. Notice the difference between the third subplots in the two plots. This just comes from the fact that if $v$ is an eigenvector, then $-v$ is also one.