Project 2 FYS4150

Vetle Vikenes, Johan Mylius Kroken & Nanna Bryne

(Dated: September 27, 2022)

The code is available on GitHub at `https://github.com/Vikenes/FYS4150/tree/main/project2`.

INTRODUCTION

To describe a one-dimensional buckling beam, we have the second order differential equation

$$\gamma \frac{\mathrm{d}^2 u}{\mathrm{d}x^2} = -F u(x), \quad x \in [0, L] \tag{1}$$

where L is the length of the horizontal beam, F is the applied force at the end point $x = L$. $u(x)$ is the vertical displacement of the beam at the horizontal position x and γ is a constant determined by the properties of the material. We consider the endpoints to be pinned down, so that $u(0) = u(L) = 0$. The endpoints are allowed to rotate, so $u'(x) \neq 0$.

PROBLEM 1

We want to recast equation (1) into a dimensionless equation, and we begin by defining the dimensionless length $\hat{x} \equiv x/L$, so that $\hat{x} \in [0,\, 1]$. The second derivative with respect to x is then

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

Equation (1) can therefore be written as

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

where we defined $\lambda \equiv F L^2/\gamma$. Inserting for $x = L\hat{x}$ yields

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

Dividing by L on both sides, we arrive at the scaled version of equation equation (1)

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

PROBLEM 2

Before implementing the Jacobi algorithm, we create a test that checks whether we set up our tridiagonal matrix A of size $N \times N$ correctly. A is defined by the signature $(a,\, d,\, a)$, where $a = -1/h^2$ and $d = 2/h^2$ with h being the step size $h \equiv (\hat{x}_{\max} - \hat{x}_{\min})/n$, where $n = N - 1$.

To test we write a short program in C++ that first sets up the tridiagonal A for $N = 6$, then uses the Armadillo library to solve the equation

$$A\mathbf{v} = \lambda \mathbf{v} \tag{3}$$

where $\mathbf{v}$ and λ are the eigenvectors and eigenvalues of A, respectively. Finally, we compare the eigenvectors and eigenvalues from Armadillo with the analytical result for $N = 6$. The analytical solution to equation (3) for a matrix of size $N \times N$ is given by

$$\lambda^{(i)} = d + 2a\cos\left(\frac{i\pi}{N+1}\right), \quad i = 1, \ldots, N \tag{4}$$

$$\mathbf{v}^{(i)} = \left[\sin\left(\frac{i\pi}{N+1}\right), \ldots, \sin\left(\frac{Ni\pi}{N+1}\right)\right]^T, \quad i = 1, \ldots, N \tag{5}$$

The resulting eigenvectors of the two methods will be scaled before comparison.

PROBLEM 3

The Jacobi algorithm relies on identifying the off-diagonal elements of a matrix with the largest absolute value, which is what we will consider now.

a)

In C++, we write a function that finds the off-diagonal element with the largest absolute value of an $N \times N$ matrix A. It takes the reference to an Armadillo matrix as input, as well as references to two integers, k and l. Algorithm 1 describes how the finds the desired off-diagonal element, and its corresponding indices, k and l. The algorithm specializes to a symmetric matrix, so we consider the upper-right off-diagonal triangle of the matrix only.

Algorithm 1 Find max off-diagonal element

```
Require: $A$ symmetric
  $k = 0$
  $l = 1$
  $max = |A_{kl}|$
  for $i \neq j$ in upper-right triangle of $A$ do
    if $|A_{ij}| > max$ then
      $max = |A_{ij}|$
      $k = i$
      $l = j$
  return $max$
```

b)

We test algoritm 1 using the matrix

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

For the test to pass, the program should return 0.7 as maximum value. The corresponding indices should then be row $k = 1$ and column $l = 2$, since we consider the upper-right triangle. Due to fine coding, the test passed.

PROBLEM 4

a)

We write a code that solves equation (3) using Jacobi's rotation algorithm. We begin by finding the largest off-diagonal element of A using algorithm 1. Using that result, we apply Jacobi's rotation algorithm, presented in Algorithm 2. This process is repeated until Algorithm 1 returns an off-diagonal element $max < \epsilon$, where $\epsilon = 10^{-8}$ is the chosen tolerance.

Algorithm 2 Jacobi rotation

```
if $A^m_{kl} = 0$ then
    $c = 1$
    $s = 0$
    $t = 0$
else
    $\tau = (A^m_{ll} - A^m_{kk})/(2A^m_{kl})$
    if $\tau > 0$ then
        $t = 1/(\tau + \sqrt{1+\tau^2})$
    else
        $t = -1/(-\tau + \sqrt{1+\tau^2})$
    $c = 1/(\sqrt{1+t^2})$
    $s = ct$
$A^{m+1}_{kk} = c^2 A^m_{kk} - 2csA^m_{kl} + s^2 A^m_{ll}$
$A^{m+1}_{ll} = c^2 A^m_{ll} + 2csA^m_{kl} + s^2 A^m_{kk}$
$A^{m+1}_{kl} = 0$
$A^{m+1}_{lk} = 0$
for $i = 0, 1, 2, \ldots, N-1$ do
    if $i \neq k \wedge i \neq l$ then
        $A^{m+1}_{ik} = cA^m_{ik} - sA_{il}$
        $A^{m+1}_{ki} = A^{m+1}_{ik}$
        $A^{m+1}_{il} = cA^m_{il} + sA^m_{ik}$
        $A^{m+1}_{li} = A^{m+1}_{il}$
    $R^{m+1}_{ik} = cR_{ik} - sR_{il}$
    $R^{m+1}_{il} = cR_{il} + sR_{ik}$
```

b)

To test the rotation algorithm, we compare its resulting eigenvalues and eigenvectors with the analytical solutions in equations (4) and (5), respectively. $N = 6$ is our choice of matrix size for comparison, with $(-1/h^2,\ 2/h^2,\ -1/h^2)$ as the signature for the triangular matrix. It worked brilliantly, as expected.

PROBLEM 5

Consider our symmetric, tridiagonal matrix $A \in \mathbb{R}^{N\times N}$ with signature (a, d, a). Let M denote the number of transformations needed for a Jacobi rotation algorithm to converge. That is, M represents the number of iterations in the Jacobi rotation algorithm needed for the transformed matrix A' to be similar enough[1] to a diagonal matrix.

For $N = 2, 3, \ldots, 100$, we run the Jacobi eigensolver and save the corresponding integer M. We do this for both the tridiagonal matrix A and an arbitrary symmetric square (dense) matrix A^* of the same size as A. The result is plotted in Figure 1. From the graph in the figure we deduce that the number of transformations scales approximately exponential with matrix size N for the tridiagonal matrix A.

Similar behaviour is seen for matrix A^* which is not a tridiagonal matrix. We would perhaps expect a tridiagonal matrix like A to converge with fewer transformations, since most of its entries are already zero. However, by running the Jacobi rotation algorithm on a tridiagonal matrix you might end up replacing zero (or close to zero) values with non-zero value in the attempt of rotating once. This mean you might have to put the same element to zero more than once and thus do more transformation that we initially would think. This is emphasized in figure Figure 1 where the M-dependency of A and A^* are approximately the same.

[1] Choosing the number $\varepsilon = 10^{-8}$ to be *close enough* to zero.

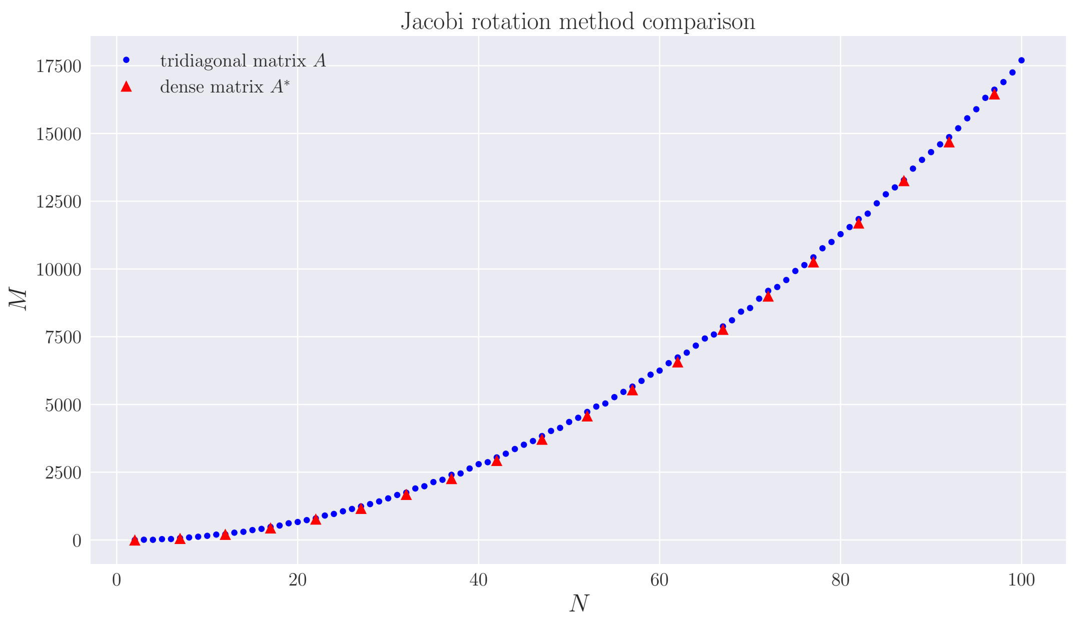

FIG. 1. The number of transformations M as function of the matrix size N.

PROBLEM 6

a)

For $n = 10$ steps, i.e. $n+1 = 11$ points $\hat{x}_i$ and $A \in \mathbb{R}^{(n-1)\times(n-1)} = \mathbb{R}^{9\times 9}$, we solve the eigenvalue problem $A\mathbf{v} = \lambda\mathbf{v}$ using the Jacobi eigensolver. In addition, we solve the same problem with the analytic expressions for $\lambda^{(i)}$ and $\mathbf{v}^{(i)}$, found using equations 4 and 5 respectively. This yields two versions of the normalised vectors $\mathbf{v}^{(i)}$, and for some i's these are counter-oriented. When we encounter this situation, the issue is solved by forcing the result from the Jacobi algorithm $\mathbf{v}^{(i)} \to -\mathbf{v}^{(i)}$.

The three resulting eigenvectors $\mathbf{v}^{(1)}$, $\mathbf{v}^{(2)}$ and $\mathbf{v}^{(3)}$ corresponding to eigenvalues $\lambda^{(1)}$, $\lambda^{(2)}$ and $\lambda^{(3)}$ s.t. $\lambda^{(i)} < \lambda^{(j)}$ for $i < j$, are plotted in Figure 2. The vectors are extended with the boundary points, i.e. $v_0^{(i)} = v_0 = 0$ and $v_n^{(i)} = v_n = 0$ for all i.

b)

We do exactly the same as in **a)**,
using $n = 100$ today!
So now you can see,
as presented in 3,
the results in a very good way!

ACKNOWLEDGMENTS

We want to express our most sincere gratitudes to our code for passing all the tests we have presented to it. This work could not have completed without the funding of Group teacher salaries, and its subsequent investment in Ringnes[2].

[2] Org. nr. 989 668 137 - RINGNES NORGE AS

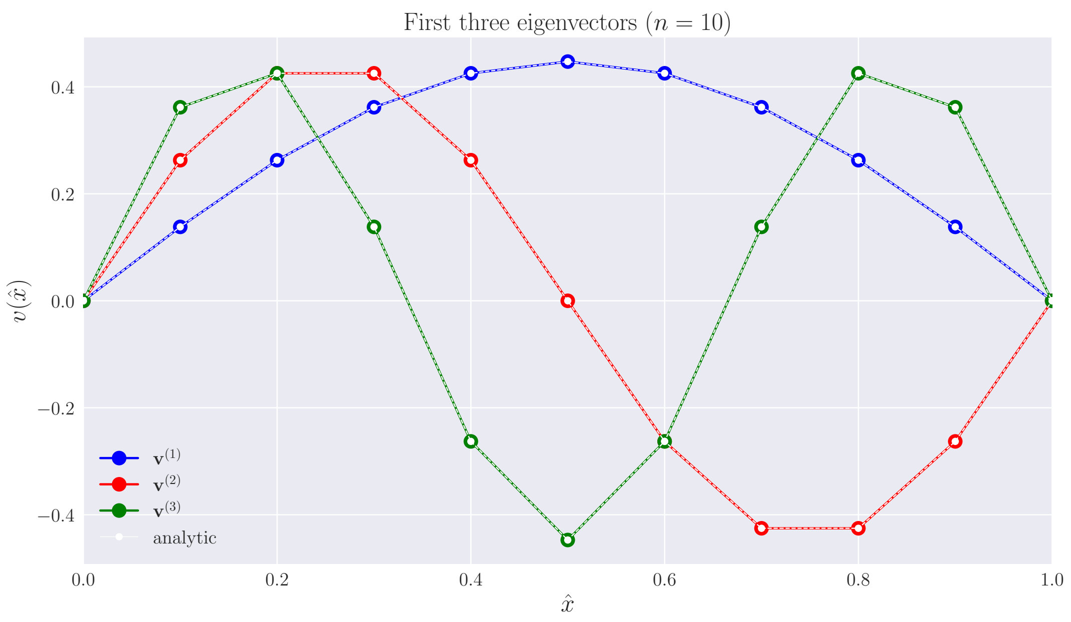

FIG. 2. The first three eigenvectors $\mathbf{v}^{(i)}$ respectively corresponding to the three lowest eigenvalues $\lambda^{(i)}$ computed with the Jacobi eigensolver using $n = 10$ discretisation steps. The white overplotted graphs are the predictions from the analytic expression.

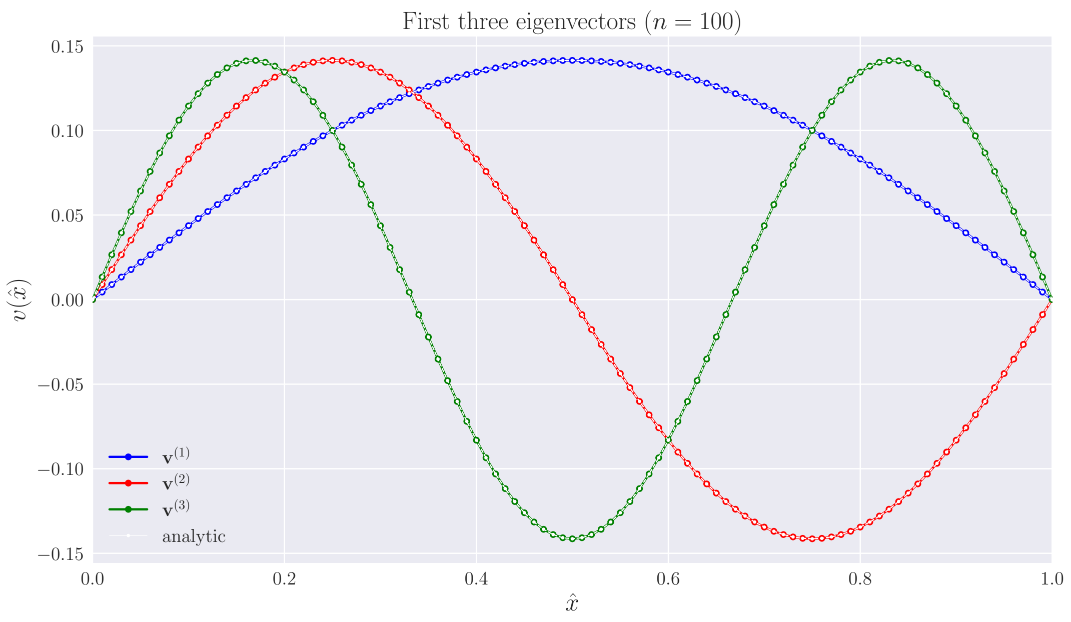

FIG. 3. The first three eigenvectors $\mathbf{v}^{(i)}$ respectively corresponding to the three lowest eigenvalues $\lambda^{(i)}$ computed with the Jacobi eigensolver using $n = 100$ discretisation steps. The white overplotted graphs are the predictions from the analytic expression.