Project 2

Henrik Haug
Axl H Kleven
Live Ljungqvist Storborg
(Dated: September 22, 2025)

Github Repository Link

INTRODUCTION

This report will be studying the situation of a one-dimensional buckling beam. The beam is of length L, and is attached at the ends. It experiences a compressing force F, and if F is large enough, the configuration will be unstable, causing a vertical displacement u(x) for small perturbations. See FIG 1

Our main focus for this report will be to numerically calculate the change in shape of the beam under pressure. Thus, our understanding of this system has many applications in both physics and real world engineering. It is also helpful to further our understanding of numerical calculations and numerical solutions of eigenvalue problems.

The second order differential equation describing our system is given by equation (1).

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

Where γ is a constant defined by the material, u(x) is the vertical displacement of the beam at horizontal position x, F is the force applied at the endpoint, and x $\in$ [0, L], where L is the length of the beam.
We will be using the scaled version of equation (1), shown in equation (2). The scaling of equations is useful to increase precision (and sometimes to even be able to do the calculations themselves) as it allows us to avoid values that are too large / small when calculating numerically.

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

Where λ is a constant given by $\lambda = \frac{FL^2}{\gamma}$. And $\hat{x} = \frac{x}{L}$ which also means that $\hat{x} \in [0, 1]$.

This differential equation will be solved numerically by applying Jacobi's rotation method. When implemented numerically, Jacobi's rotation algorithm is an iterative method well suited for parallelisation, although not very efficient for large matrices. The method is reasonably intuitive, as well as being suited for real, symmetric matrices. This is perfect for us, as we want to increase our understanding of the algorithms as well as only working with smaller, real, and symmetric matrices.

As mentioned, this report will heavily rely on numerical calculations. The differential equation we have shown in equation (2) is written for a continuous $\hat{x}$. This is not possible to work with, and thus our first step will be to discretize the equation. This will allow us to write the problem as a linear algebra eigenvalue problem. To solve this problem we will implement a solution using Jacobi's rotation algorithm, and explore the results and limitations of this solution.

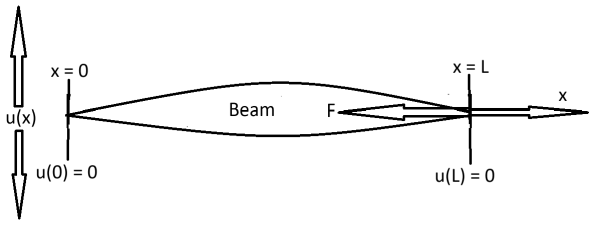

FIG. 1. At the end of the beam, x = L, a force F is applied. This compresses the beam, causing it to bend into new shapes. u(x) denotes this horizontal displacement. Because the beam is held at x = 0, and x = L it can't change shapes here (u(0, L) = 0).

METHOD & THEORY

Discretization

We discretize $\hat{x}$ by dividing the range of $\hat{x}$: [0, 1], into n parts. This will create a discretized $\hat{x}$ of n+1 points $\hat{x}_0, \hat{x}_1, ..., \hat{x}_n$. This creates a stepsize h

$$h = \frac{\hat{x}_{\max} - \hat{x}_{\min}}{n} = \frac{1-0}{n} = \frac{1}{n}$$

Meaning we write our discretized points $\hat{x}_i = \hat{x}_0 + ih$, for i = 0, 1, ..., n.

Further using v_i to denote our approximate solution to u_i, we can rewrite equation (2) using the definition of the double derivative:

$$-\left(\frac{v_{i+1} - 2v_i + v_{i-1}}{h^2}\right) = \lambda v_i$$

$$\frac{-v_{i+1} + 2v_i - v_{i-1}}{h^2} = \lambda v_i$$

For our bondary conditions, $v_0 = v_n = 0$, this can be written as the linear algebra eigenvalue problem

$$\mathbf{A}\vec{v} = \lambda\vec{v}$$

A will here be a tridiagonal matrix on the form (a, d, a) and of dimensions N x N, where N = n - 1. Meaning $\vec{v}$ will be on the form $\vec{v} = [v_1, v_2, ..., v_{n-1}]$. Here a = -$\frac{1}{h^2}$ and d = $\frac{2}{h^2}$

Analytical solutions

We need a test for our numerical results, and choose to test them against the analytical solutions of the eigenvalue problem of a given triadiagonal matrix A(a, d, a) of size N x N. The analytical solutions are shown in equations (3, 4).

$$\lambda^{(j)} = d + 2a\cos\frac{j\pi}{N+1} \qquad \text{for j = 1, ..., N} \tag{3}$$

$$\vec{v}^{(j)} = \left[\sin\frac{j\pi}{N+1}, \sin\frac{2j\pi}{N+1}, ..., \sin\frac{Nj\pi}{N+1}\right]^T \quad \text{for j = 1, ..., N} \tag{4}$$

Jacobi's rotation method

The main idea behind Jacobi's rotation method is to apply the spectral theorem as defined in equation(5)

$$\mathbf{S}^T\mathbf{A}\mathbf{S} = \mathbf{D} \tag{5}$$

to find the matrix S by iteratively applying similarity transformations, also called rotations, to A. And eventually ending up with the diagonal matrix D [1].

$$\mathbf{A} \to \mathbf{S}_1^T\mathbf{A}\mathbf{S}_1 \to \mathbf{S}_2^T\mathbf{S}_1^T\mathbf{A}\mathbf{S}_1\mathbf{S}_2 \to ...$$

These rotations will cause the resulting matrix to be a little more diagonal-like, until eventually the matrix will be sufficiently close to being diagonal, and then, the product of all our applied transformations will be our approximation to S.

The transformations can more easily be written

$$\mathbf{A}^{m+1} = \mathbf{S}_m^T\mathbf{A}^m\mathbf{S}_m$$

And for practical reasons, keeping track of all the applied transformations can easily be done by using a matrix R.

$$\mathbf{R}^{m+1} = \mathbf{R}^m S_m$$

So by using $\mathbf{R}^1 = \mathbf{I}$, our final $\mathbf{R}^M$ will be our approximation to S.
Lets have a closer look at each rotation matrix $\mathbf{S}_m$. For illustration, we'll be using a 10 x 10 matrix with $\cos\theta = c_\theta$ and $\sin\theta = s_\theta$

$$\mathbf{S}_m = \begin{bmatrix} 1 & & & & & & & & & \\ & 1 & & & & & & & & \\ & & 1 & & & & & & & \\ & & & c_\theta & & & s_\theta & & & \\ & & & & 1 & & & & & \\ & & & & & 1 & & & & \\ & & & -s_\theta & & & c_\theta & & & \\ & & & & & & & 1 & & \\ & & & & & & & & 1 & \\ & & & & & & & & & 1 \end{bmatrix}$$

With all empty elements being zero. $\mathbf{S}_m$ represents a clockwise rotation of an angle θ in the (x_4, x_7) plane.

Jacobi's rotation algorithm

This is the general numerical implementation of Jacobi's rotation method. And goes, according to Kvellestad [1] pages 73 - 75, as:

Step 1:
Choose a tolerance ϵ, and let $\mathbf{A}^1 = \mathbf{A}$, and $\mathbf{R}^1 = \mathbf{I}$.

Step 2:
Find the indices (k, l) of the maximum (absolute value) off-diagonal element in $\mathbf{A}^1$

Step 3:
While $|a_{kl}^m| > \epsilon$:

Step 3.1:Compute

$$\tau = \frac{a_{ll}^m - a_{kk}^m}{2a_{kl}^m}$$

Step 3.2: Compute

$$t_\theta = -\tau \pm \sqrt{1 + \tau^2}$$

And choosing the solution which gives the smallest t_θ:

$$\text{if } \tau > 0 : t_\theta = \frac{1}{\tau + \sqrt{1 + \tau^2}}$$

$$\text{if } \tau < 0 : t_\theta = \frac{1}{\tau + \sqrt{1 - \tau^2}}$$

Compute

$$c_\theta = \frac{1}{\sqrt{1 + t_\theta}}$$

$$s_\theta = c_\theta t_\theta$$

[1] See Kvellestad [1] for more on similarity transformations.

Step 3.3: Transform $\mathbf{A}^m \rightarrow \mathbf{A}^{m+1}$ by updating $\mathbf{A}$ according to the similarity transformation caused by S. Starting with the indexes of k and l:

$$a_{kk}^{m+1} = a_{kk}^m c_\theta^2 - 2a_{kl}^m c_\theta s_\theta + a_{ll}^m s_\theta^2$$

$$a_{ll}^{m+1} = a_{ll}^m c_\theta^2 + 2a_{kl}^m c_\theta s_\theta + a_{kk}^m s_\theta^2$$

$$a_{kl}^{m+1} = 0$$

$$a_{lk}^{m+1} = 0$$

Step 3.4: And then continuing by updating all other elements along these rows and columns.
For all i where i $\neq$ k and i $\neq$ l:

$$a_{ik}^{m+1} = a_{ik}^m c_\theta - a_{il}^m s_\theta$$

$$a_{ki}^{m+1} = a_{ik}^{m1}$$

$$a_{il}^{m+1} = a_{il}^m c_\theta + a_{ik}^m s_\theta$$

$$a_{li}^{m+1} = a_{il}^{m+1}$$

Updating the overall rotation matrix $\mathbf{R}^m \rightarrow \mathbf{R}^{m+1}$

$$r_{ik}^{m+1} = r_{ik}^m c_\theta - r_{il}^m s_\theta$$

$$r_{il}^{m+1} = r_{il}^m c_\theta + r_{ik}^m s_\theta$$

Step 3.5: Again finding the indexes (k, l) of the maximum off-diagonal element in $\mathbf{A}$, which is now $\mathbf{A}^{m+1}$. The loop returns to step 3.

Step 4: Read out the eigenvalues from the final matrix $\mathbf{A}^{m+1}$. And the eigenvectors from the final matrix $\mathbf{R}^{m+1}$

Similarity transformation

By running Jacobi´s rotation method as explained above, we can count how many similarity transformations take place by implementing an iterator in the algorithm. This way, we get direct access to the link between matrix size and the amount of similarity transformations that take place.

Discretization method

By using the method for discretization of $\hat{x}$ described above, we produced the figures (2) and (3).
We have implemented a sorting algorithm to extract the eigenvectors corresponding to the three smallest eigenvalues. These eigenvectors are shown in figure (2) and (3) together with the analytical solution.

Tools

We have used the C++ library armadillo for our implementation of vectors and matrices used in the Jacobi Rotation algorithm, as well as for testing our eigenvalues / vectors. We have also applied the Python library matplotlib to produce all illustrations in this report.

RESULTS & DISCUSSION

In appendix (I) we have shown that using $\hat{x} \equiv \frac{x}{L}$ We can write equation (1) as equation (2).

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

$$\downarrow$$

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

We set up a script for a tridiagonal matrix $\mathbf{A}$ for N = 6 and solve the equation $\mathbf{A}\vec{v} = \lambda\vec{v}$ using Armadillo. We compute the analytical eigenvalues λ and eigenvectors $\vec{v}$ and subtract them from armadillos to compare. The results are shown in table (I).

$$\Delta\lambda = 10^{-14} \begin{bmatrix} -0.53291 \\ 2.1316 \\ 2.8422 \\ 2.8422 \\ -2.8422 \\ -5.6843 \end{bmatrix}$$

$$\Delta\vec{v} = 10^{-16} \begin{bmatrix} 0.83267 & -0.55511 & 2.2204 & 2.2204 & 1.1102 & -1.6653 \\ 2.2204 & 2.2204 & 0 & 0 & -5.5511 & 1.6653 \\ 0 & 3.0531 & 0 & -4.9960 & 3.0531 & 0 \\ 2.2204 & 2.2204 & -3.3307 & 2.7756 & -2.7756 & 2.2204 \\ 6.1062 & -3.3307 & -5.2736 & 3.0531 & 1.1102 & -1.6653 \\ 7.7716 & -4.9960 & 4.4409 & -2.2204 & -8.8818 & -3.0531 \end{bmatrix}$$

TABLE I. Results from subtracting the analytical eigenvalues λ and eigenvectors $\vec{v}$ from armadillos eigenvalues and eigenvectors.

We can see that all the values are 0, or approximately 0, meaning the analytical eigenvalues and vectors agree very well with the numerical eigenvalues and vectors calculated using the armadillo package.

Further we design a script that can identify the largest (absolute value) off-diagonal element in a matrix. This is necessary for the implementation of the Jacobi rotation algorithm. We test the function by using

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

The script returns the value 0.7, with indices (2, 1), meaning the index of -0.7. The script appears to fulfill its purpose [2].

The implementation of the Jacobi rotation algorithm was done according to the the algorithm shown in the methods section. We test the result of this algorithm by checking that the eigenvectors and eigenvalues correspond with the analytical results for N = 6. Yet again we subtract the results of the algorithm from the analytical values. The results are shown in table (II).

$$\Delta\lambda = 10^{-14}\begin{bmatrix} -3.9080 \\ -6.3949 \\ 1.4211 \\ -9.9476 \\ -227.37 \\ -170.53 \end{bmatrix}$$

$$\Delta\vec{v} = 10^{-13}\begin{bmatrix} -472.59 & 222.61 & 0.91149 & -0.90705 & 261.55 & -399.82 \\ -407.81 & 398.92 & -0.40218 & -0.40135 & -327.96 & 498.56 \\ -262.26 & 498.97 & -0.72831 & 0.72553 & 588.90 & -221.88 \\ 181.50 & 498.97 & 0.72276 & 0.72553 & -407.40 & -221.89 \\ 589.32 & 398.90 & 0.39718 & -0.39690 & 473.49 & 498.56 \\ 327.05 & 222.60 & -0.90483 & -0.91260 & -180.78 & -399.81 \end{bmatrix}$$

TABLE II. Results from subtracting the analytical eigenvalues λ and eigenvectors $\vec{v}$ from the eigenvalues and eigenvectors calculated numerically using Jacobi's rotation algorithm.

When we count the amount of similarity transformations that take place when running Jacobi´s rotation method on symmetric tridiagonal matrices of different sizes, we get the results shown in table (III).

By studying these results, we can clearly see that there is a correspondence of a matrix of size $N \times N$ needing an N^2 amount of similarity transformations until all non-diagonal elements are 0. This quadratic scaling can be expressed as the cost $O(N^2)$

If however, A were a dense matrix, we would still expect to see a quadratic scaling of cost $O(N^2)$. The difference in scaling between a symmetric tridiagonal matrix and a dense matrix is that the dense matrix will take longer, as it has more elements to converge, but the dense matrix will all in all follow the same trend as the symmetric tridiagonal matrix.

TABLE III. Table containing comparison of symmetric matrices with size N x N and amount of similarity transformations

Matrix Size	Similarity transformations
5	32
10	154
15	366

[2] Program will not be sent back to source for the time being

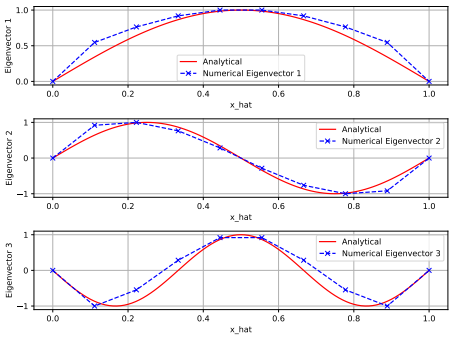

FIG. 2.
The figure shows the comparison of the analytical solution (in red) and the numerical solution (in blue) at N=10,.

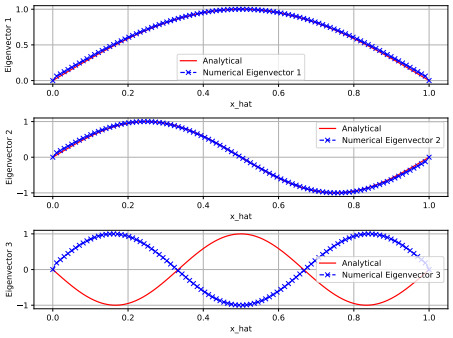

FIG. 3.
The figure shows the comparison of the analytical solution (in red) and the numerical solution (in blue) at N=100.

At last, we present the numerical and analytical solutions for each eigenvector corresponding to the three smallest λ in figure (2) and (3).

The reason we are seeing the opposite graph for eigenvector 3 in figure (3) is because of different signs. This result is still valid because of the eigenvector property where a scaled eigenvector is equivalent to the eigenvector itself because of the relationship between the elements in the eigenvector. In this case we see a scaling of -1 in the solution of the third numerical eigenvector compared to the analytical solution.

During the calculations of the presentation of figure (2) and (3), we realized that out eigenvectors were of the wrong size. Our eigenvectors were of size N instead of N+2, leaving no room for adding boundary points, resulting in us having to overwrite the first and last point of our eigenvectors. This has little effect in figure (3), but has more effect in figure (2), because by removing two points, we remove 20% of our data. This is also visible, as the graphs are in figure (2) are more rough approximation to the analytical graph.

[1] Anders Kvellestad. Lecture notes fys3150 – computational physics, fall 2024. *UiO*, 1(1):66, 2024.

I. APPENDIX

In the introduction we presented equations (1 and 2) denoting the second order differential equation describing our system. We will here prove that equation (2) is the scaled version of equation (1) using $\hat{x} \equiv \frac{x}{L}$.
To change the variable of differentiation, we start by doing the derivatives on the left side using the chain rule.

$$\frac{d^2u(x)}{dx^2}$$

$$\frac{d}{dx}\left(\frac{du(x)}{dx}\right)$$

Where the first derivative in the parentheses is given by:

$$\frac{du(x)}{dx} = \frac{du(x)}{d\hat{x}}\frac{d\hat{x}}{dx} = \frac{du(x)}{d\hat{x}}\frac{d\frac{x}{L}}{dx} = \frac{1}{L}\frac{du(x)}{d\hat{x}}$$

And the second derivative:

$$\frac{d^2u(x)}{dx^2} = \frac{d}{dx}\left(\frac{1}{L}\frac{du(x)}{d\hat{x}}\right) = \frac{1}{L}\frac{d^2u(x)}{dxd\hat{x}} = \frac{1}{L}\frac{d^2u(x)}{d(\hat{x}L)d\hat{x}}$$

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

Which we can insert into the original equation (1):

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

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

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

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

Which is equal to equation (2). We can do this last step where $u(L\hat{x})$ on both sides go to $u(\hat{x})$, because L is just a scaling constant inside the function.