Project 2

Adrian Gundersen, Casper Johnsen & Victor B. Johansen
(Dated: September 23, 2025)

https://github.uio.no/adriangg/FYS3150/

INTRODUCTION

For this project, we will be looking at the well-known buckling-beam problem. This is a one-dimensional problem yielding an ordinary differential equation. We imagine a beam of length L that is subject to a force F. Our goal is then to look at how the force impacts the displacement of the beam. This will be done by implementing linear algebra in various manners.

PROBLEM 1

Firstly, we want to show that a differential equation can be written as an eigenvalue problem. To do so we will be scaling the units of the equation. The problem involves a horizontal beam where we let $u(x)$ be the horizontal displacement, and $x \in [0, L]$ be a point along our beam. The problem is described by the following equation:

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

where γ is a material property constant, and F is the force applied on the endpoint ($x = L$) pointing towards the origin ($x_0 = 0$). Additionally, the endpoints are fixed at $u(x = 0) = u(x = L) = 0$. They can, however, be rotated, meaning that we do not require $u'(x) = 0$ in the endpoints.

The first step is to define a dimensionless variable $\hat{x} \equiv \frac{x}{L} \in [0, 1]$. We then also need to correlate our u-function to our new variable. However, as we will not be using our previous u-function, we will be denoting our new function by the same letter u, even though it works on $\hat{x}$. If we wanted to be specific $u_{\hat{x}}(\hat{x}) = u_x(x)$. But, for simplicity, $u_{\hat{x}}(\hat{x}) \equiv u(\hat{x})$.

Using $\hat{x} = \frac{x}{L}$ and the chain rule we find the derivatives yield:

$$\begin{aligned} \frac{d}{dx} &= \frac{d}{d\hat{x}}\frac{d\hat{x}}{dx} = \frac{1}{L}\frac{d}{dx} \\ \frac{d^2}{dx^2} &= \frac{1}{L^2}\frac{d}{d\hat{x}^2} \end{aligned} \tag{2}$$

Now we can rework Equation 2 into:

$$\frac{d^2u(\hat{x})}{d\hat{x}^2} = -\lambda u(\hat{x}) \tag{3}$$

where $\lambda = \frac{FL^2}{\gamma}$. Now we clearly see that we have an eigenvalue problem with eigenvalue λ.

PROBLEM 2

In the coming problems we are going to implement the Jacobi Algorithm to find the numerical solution. First we will however lay the groundwork. We let $\mathbf{A} \in \mathbb{R}^{N\times N}$ be a tridiagonal matrix with diagonal $\mathbf{d} \in \mathbb{R}^N$ and sub- and superdiagonal $\mathbf{a} \in \mathbb{R}^{N-1}$. We want to use this matrix and find the eigenvalues $\lambda^{(j)}$ and eigenvectors $\mathbf{v}^{(j)}$.

$$\mathbf{Av} = \lambda\mathbf{v} \tag{4}$$

We know that the analytical eigenvalue-eigenvector pair for the problem is given by:

$$\lambda^{(j)} = d + 2a\cos\left(\frac{j\pi}{N+1}\right), \quad \mathbf{v}^{(j)} = \left(\sin\tfrac{j\pi}{N+1}, \sin\tfrac{2j\pi}{N+1}, \ldots, \sin\tfrac{Nj\pi}{N+1}\right), \quad j = 1, \ldots, N \tag{5}$$

We want to try this on a specific matrix $\mathbf{A}$ where the diagonal $\mathbf{d} = (d, d, \ldots, d)$ and the sub- and superdiagonal $\mathbf{a} = (a, a, \ldots, a)$ and $d = \frac{2}{h^2}$ and $a = \frac{-1}{h^2}$. Here $h = \frac{1}{n}$ denotes the step size, with $n = N + 1$. We also let $N = 6$ for our test.

Now we use armadillo's [1] function `arma::eig_sym` to calculate eigenvectors and eigenvalues. Then we want to see that the two methods are consistent. To do so we normalize the vectors by dividing by the norm.

TABLE I. Differences in eigenvalues and eigenvectors for $j = 1, \ldots, 6$. The difference in eigenvectors is calculated by $\|\mathbf{v}_{\mathbf{analytical}} - \mathbf{v}_{\text{numerical}}\|$.

j	Difference in eigenvalue	Norm of difference in eigenvector
1	$3.55 \cdot 10^{-15}$	$3.48 \cdot 10^{-16}$
2	$2.13 \cdot 10^{-14}$	$5.69 \cdot 10^{-16}$
3	$1.42 \cdot 10^{-14}$	$7.60 \cdot 10^{-16}$
4	$2.84 \cdot 10^{-14}$	$6.11 \cdot 10^{-16}$
5	0	$1.08 \cdot 10^{-15}$
6	$5.68 \cdot 10^{-14}$	$6.23 \cdot 10^{-16}$

From Table I we see that the difference is down of order 10^{-14}.

PROBLEM 3

Now we want to make a short function that takes in a general symmetric matrix and outputs the maximum absolute value of the off-diagonals and the indices (k, l). To see that it works we try it on a simple matrix:

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

Knowing that the matrix is symmetric we need only to look at the upper right triangle. As the armadillo library stores matrices in column-major order, we iterate over the columns first and then the rows for cache-efficiency. Running our program we get that 0.7 is the max value at index (1,2) which is as expected knowing that indexing starts at 0.

PROBLEM 4

We now implement the max-off-diagonal Jacobi rotation algorithm as written in the lecture notes [2]. The idea is to solve the eigenvalues of a symmetric matrix $\mathbf{A}$ by rotating the off diagonal elements toward zero. The rotation matrix $\mathbf{S}(p, q, \theta)$ also creates the matrix $\mathbf{R}$ such that its vectors are the eigenvectors of $\mathbf{A}$.

$$\mathbf{A}^{(m+1)} = \mathbf{S}^\top \mathbf{A}^{(m)} \mathbf{S}$$

The rotations are accumulated in $\mathbf{R}$ via $\mathbf{R} \leftarrow \mathbf{RS}$, so after s steps

$$\mathbf{A}^{(s)} = \mathbf{R}^\top \mathbf{A}^{(0)} \mathbf{R} \approx \mathbf{\Lambda} \text{ (diagonal)}.$$

Hence,

$$\mathbf{A}^{(0)} \approx \mathbf{R}\,\mathbf{\Lambda}\,\mathbf{R}^\top, \quad \text{with the columns of } \mathbf{R} \text{ being the orthonormal eigenvectors.}$$

Algorithm 1 Jacobi rotation algorithm: a & r are matrix elements, m is iteration, $c_\theta = \cos\theta$, $s_\theta = \sin\theta$ and $t_\theta = \tan\theta$.

Step 1
Choose some tolerance ϵ
let $\mathbf{A}^{(1)} = \mathbf{A}$
let $\mathbf{R}^{(1)} = \mathbf{I}$

Step 2
Declare a variable $|a_{kl}^{(m)}|$
Step 3
while $|a_{kl}^{(m)}| > \epsilon$ **do**
Let $|a_{kl}^{(m)}|$ be the maximum absolute value of the off-diagonal in $\mathbf{A}$. Keep this along with indices (k, l)

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

If $\tau > 0$

$$\frac{1}{\tau + \sqrt{1+\tau^2}}$$

else

$$\frac{1}{\tau - \sqrt{1+\tau^2}}$$

compute cos and sin

$$c_\theta = \frac{1}{\sqrt{1+t_\theta^2}} \qquad s_\theta = c_\theta t_\theta$$

Transformation to time = m+1. c_θ^2, s_θ^2 and $c_\theta s_\theta$ are calculated before m+1 to save FLOPs, matrix elements saved as temps to save memory pulls.

$$\begin{aligned}
a_{kk}^{(m+1)} &= a_{kk}^{(m)}\, c_\theta^2 - 2\, a_{kl}^{(m)}\, c_\theta s_\theta + a_{ll}^{(m)}\, s_\theta^2, \\
a_{ll}^{(m+1)} &= a_{ll}^{(m)}\, c_\theta^2 + 2\, a_{kl}^{(m)}\, c_\theta s_\theta + a_{kk}^{(m)}\, s_\theta^2, \\
a_{kl}^{(m+1)} &= 0, \\
a_{lk}^{(m+1)} &= 0.
\end{aligned}$$

For all i where $i \neq k$ and $i \neq l$:

$$\begin{aligned}
a_{ik}^{(m+1)} &= a_{ik}^{(m)}\, c_\theta - a_{il}^{(m)}\, s_\theta, \\
a_{ki}^{(m+1)} &= a_{ik}^{(m+1)}, \\
a_{il}^{(m+1)} &= a_{il}^{(m)}\, c_\theta + a_{ik}^{(m)}\, s_\theta, \\
a_{li}^{(m+1)} &= a_{il}^{(m+1)}.
\end{aligned}$$

Update the overall rotation matrix, $\mathbf{R}^{(m)} \to \mathbf{R}^{(m+1)} = \mathbf{R}^{(m)}\mathbf{S}_m$, by updating elements.

For all i :

$$r_{ik}^{(m+1)} = r_{ik}^{(m)}\, c_\theta - r_{il}^{(m)}\, s_\theta \tag{6}$$

$$r_{il}^{(m+1)} = r_{il}^{(m)}\, c_\theta + r_{ik}^{(m)}\, s_\theta \tag{7}$$

To use our implementation of the Jacobi rotation algorithm 1 we need to run it multiple times. Every time we run it the highest off-diagonal elements l and k decreases. But the off-diagonal elements do not necessarily reach zero. For that reason we implement a factor ϵ, which is a small number (In our case $\epsilon = 10^{-8}$). Then we loop the Jacobi rotation algorithm until the maximum off-diagonal element is less than ϵ. If we choose a small ϵ, the loop may run for a long time. For that reason we also set a ceiling on the allowed iterations. We will later look at how required iterations scale with N.

TABLE II. Eigenvalues using Jacobi rotation algorithm vs. the analytical eigenvalues for $N = 6$.

j	$\lambda^{(j)}$ (Jacobi)	$\lambda^{(j)}$ Analytical
1	9.7051	9.7051
2	36.8980	36.8980
3	76.1930	76.1930
4	119.8100	119.8100
5	159.1000	159.1000
6	186.2900	186.2900

TABLE III. Eigenvectors using the Jacobi rotation algorithm vs. the analytical eigenvectors for $N = 6$. Each column $v^{(j)}$ belongs to the eigenvalue $\lambda^{(j)}$. The rightmost column is the sum of all differences in absolute values for each k, $\sum_{j=1}^{6}\left(|v_{k,\text{an}}^{(j)}| - |v_k^{(j)}|\right)$.

$v_k^{(j)}$	Numerical $\mathbf{v}^{(j)}$ (Jacobi)						Analytical $\mathbf{v}^{(j)}$						$\sum_{j=1}^{6}\left(\lvert v_{k,\text{an}}^{(j)}\rvert - \lvert v_k^{(j)}\rvert\right)$
	$v^{(1)}$	$v^{(2)}$	$v^{(3)}$	$v^{(4)}$	$v^{(5)}$	$v^{(6)}$	$v^{(1)}$	$v^{(2)}$	$v^{(3)}$	$v^{(4)}$	$v^{(5)}$	$v^{(6)}$	
$k = 1$	0.2319	-0.4179	-0.5211	-0.5211	0.4179	0.2319	0.2319	0.4179	0.5211	0.5211	0.4179	0.2319	0.0000
$k = 2$	-0.4179	0.5211	0.2319	-0.2319	0.5211	0.4179	-0.4179	-0.5211	-0.2319	0.2319	0.5211	0.4179	0.0000
$k = 3$	0.5211	-0.2319	0.4179	0.4179	0.2319	0.5211	0.5211	0.2319	-0.4179	-0.4179	0.2319	0.5211	0.0000
$k = 4$	-0.5211	-0.2319	-0.4179	0.4179	-0.2319	0.5211	-0.5211	0.2319	0.4179	-0.4179	-0.2319	0.5211	0.0000
$k = 5$	0.4179	0.5211	-0.2319	-0.2319	-0.5211	0.4179	0.4179	-0.5211	0.2319	0.2319	-0.5211	0.4179	0.0000
$k = 6$	-0.2319	-0.4179	0.5211	-0.5211	-0.4179	0.2319	-0.2319	0.4179	-0.5211	0.5211	-0.4179	0.2319	0.0000

As we can see in Table III the Jacobi rotation algorithm produces no noticeable difference from the analytical eigenvectors for $N = 6$. It is important to notice that we normalize the eigenvectors to have the same scaling. We see that even as we sum up all the errors it is zero up to the fourth decimal. For higher precision there will however always be discrepancies depending on our tolerance ϵ and round-off errors. For low enough ϵ the precision of your machine will also play part as seen in Project 1.

We see the same for the eigenvalues in Table II. There is no difference up to four decimals between the numerical method and the analytical solution.

PROBLEM 5

For Problem 5 we want to see for different values of N, how many transformations we need to perform before all off-diagonal elements are below our tolerance ϵ. We let $N = \{5, 10, 15, \ldots, 100\}$ and run our Jacobi algorithm. We do this for both our tridiagonal matrix describing the problem and a random dense, symmetric matrix. We then compare the number of required iterations.

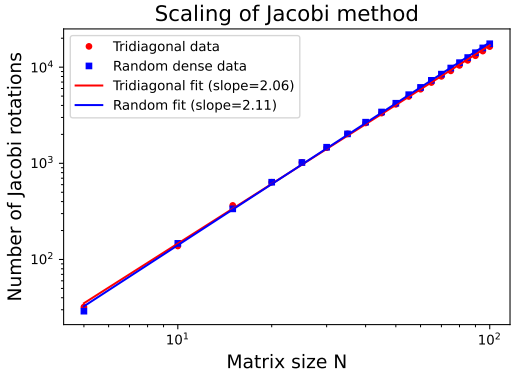

FIG. 1. A log-log-plot describing the number of Jacobi rotations for both a tridiagonal matrix and a random dense matrix to have off-diagonal elements below our tolerance ϵ. The x-axis denotes the matrix size N. The y-axis denotes the number of Jacobi rotations.

Plotting the number of iterations against matrix size N in a loglog-plot in Figure 1, gives us a seemingly linear relation. This is equivalent to the number of iterations scaling polynomially with size N. By plotting logarithmically and doing a linear regression, we get the relation between number of rotations N and the slope for the loglog-plot. We then know $\mathcal{O}(N^{\text{slope}})$.

From Figure 1, we see that the tridiagonal matrix initialized by the buckling-beam problem generally requires less iterations than the random dense matrix. For the tridiagonal matrix the scaling is $\mathcal{O}(N^{2.062})$ and for the dense matrix $\mathcal{O}(N^{2.110})$.

PROBLEM 6

In this problem we are solving Equation 4 by the Jacobi rotation method for $N = 9$ and $N = 99$. This means that the total step size is $n = 10$ and $n = 100$ (as $n = N + 1$). We then look at the three eigenvectors $\mathbf{v}$ with lowest eigenvalues and plot them against $\hat{x}$. In addition, we compare them to the analytical solution.

Since we are looking at the interior points of our domain,, we also add our fixed boundary points to the numerical eigenvector. We know that we have Dirichlet boundary conditions, so that the vector element at $\hat{x} = 0, L$ yields a 0-element in the vector.

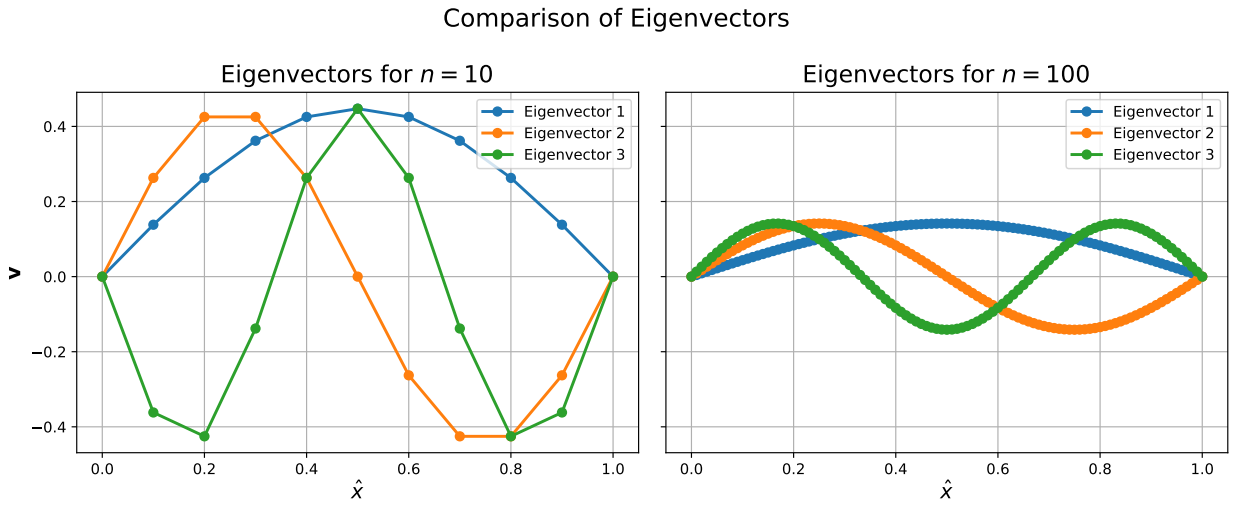

FIG. 2. Comparison of the three eigenvectors with the lowest eigenvalue. $n = N + 1$, of the $N \times N$ matrix. (Vector element $\hat{v}_i$ against position $\hat{x}_i$)

In Figure 2 we can see the right and left plots are quite similar in shape. They are both sine-modes with the same number of nodes and sometimes flipped sign. However, a flipped sign is the same solution due to linear dependence. The only difference for generating the data in the two plots is the value of n.

Another side note we noticed was the number of nodes of the sine-modes we plotted increased with increasing related eigenvalues. The second lowest eigenvalue gave us two modes. And by testing we found that the fourth gave degree four and so on.

What should be noticed is that the magnitude of the eigenvectors are different. This is due to the normalization of the state. We require that each state $\|\mathbf{v}^{(j)}\| = 1$. This is analogous to $\sum_i (v_i^{(j)})^2 = 1$. So as we get finer discretizations, the amplitude changes for the points.

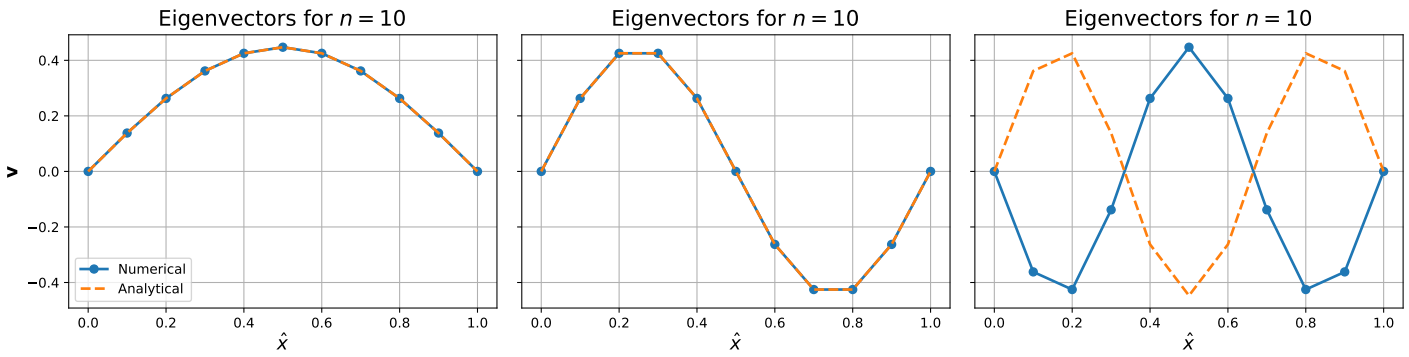

FIG. 3. Comparison of the three eigenvectors with lowest eigenvalue against the analytical solution, with $n = 10$. (Vector element $\hat{v}_i$ against position $\hat{x}_i$)

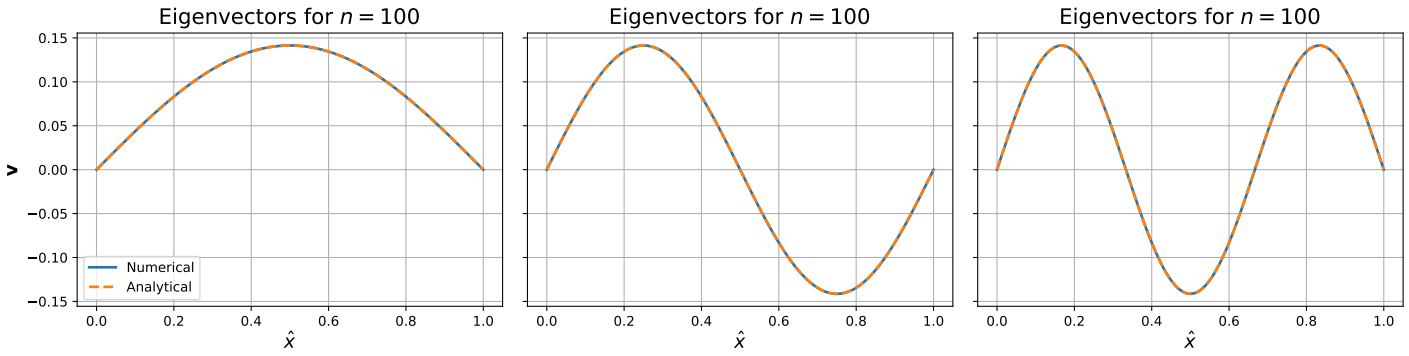

FIG. 4. Comparison of the three eigenvectors with lowest eigenvalue against the analytical solution, with $n = 100$. The x-axis depicts the indexing we give the eigenvalues. The y-axis shows the respective eigenvectors.

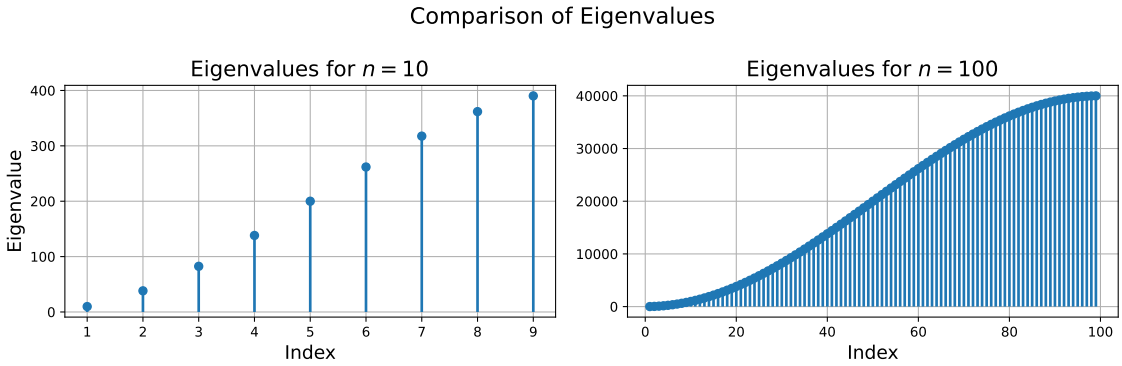

FIG. 5. Eigenvalues for the $N \times N$ matrix $\mathbf{A}$ with $n = 10$ and $n = 100$. The x-axis depicts the indexing we give the eigenvalues. The y-axis shows the respective eigenvalues.

We know that the Jacobi rotation method randomly picks the smallest value without caring about sign while calculating the eigenvectors. In Figure 3 the two first vectors have correct sign compared to the analytical solution. In Figure 4 all eigenvectors have correct sign.

We wanted to observe what happens with the eigenvalue as the steps increase. By inspecting Figure 5 closely, the shape is roughly conserved. By increasing n the shape truly takes form.

DECLARATION OF USE OF GENERATIVE AI

In this scientific work, generative artificial intelligence (AI) has been used. All data and personal information have been processed in accordance with the University of Oslo's regulations, and we, as the authors of the document, take full responsibility for its content, claims, and references. An overview of the use of generative AI is provided below.

Summary

- **Tool(s) used:** OpenAI ChatGPT (GPT-5), https://chatgpt.com/
- **Use:**
  - Generating boilerplate code like plotting with Matplotlib.
  - Generating Makefiles.
  - Formatting README.md.
  - Checking language for clarity and grammar and general proof reading.
  - Generating brief code documentation/comments for better readability in adherence to conventions.
  - Creating tables in proper tex-format.
  - Brainstorming ideas for optimizing code for efficiency.

---

[1] C. Sanderson and R. Curtin, Journal of Open Source Software **1**, 26 (2016).
[2] A. Kveim and collaborators, "Fys3150 lecture notes," https://github.com/anderkve/FYS3150/tree/master/lecture_notes, accessed: 2025-09-10.