Reflective Report

Design and Creative Technologies

Torrens University, Australia

Student: Luis Guilherme de Barros Andrade Faria - A00187785
Subject Code: MFA 502
Subject Name: Mathematical Foundations of Artificial Intelligence
Assessment No.: 2A
Title of Assessment: Reflective Report, Set 1, Problem 2
Lecturer: Dr. James Vakilian
Date: Oct 2025

Table of Contents

1. Introduction and Overview

The second component of *EigenAI* focused on computing eigenvalues and eigenvectors for a 2×2 matrix without using external libraries.

After finishing the determinant recursion logic, I wanted to extend the same clarity and interactivity to another essential linear-algebra concept. I structured this challenge to produce correct eigenpairs and teach the underlying steps interactively through *EigenAI's* Streamlit interface.

The EigenAI architecture remained consistent with Set 1 Problem 1:

- Frontend / Presentation layer: Streamlit UI for matrix input, progress animations, and result display.
- Logic layer: Pure-Python solver (`eigen_solver.py`) implementing characteristic-polynomial computation and vector derivation.
- Integration: `set1Problem2.py` connects both layers, displaying tutor-style explanations during execution.

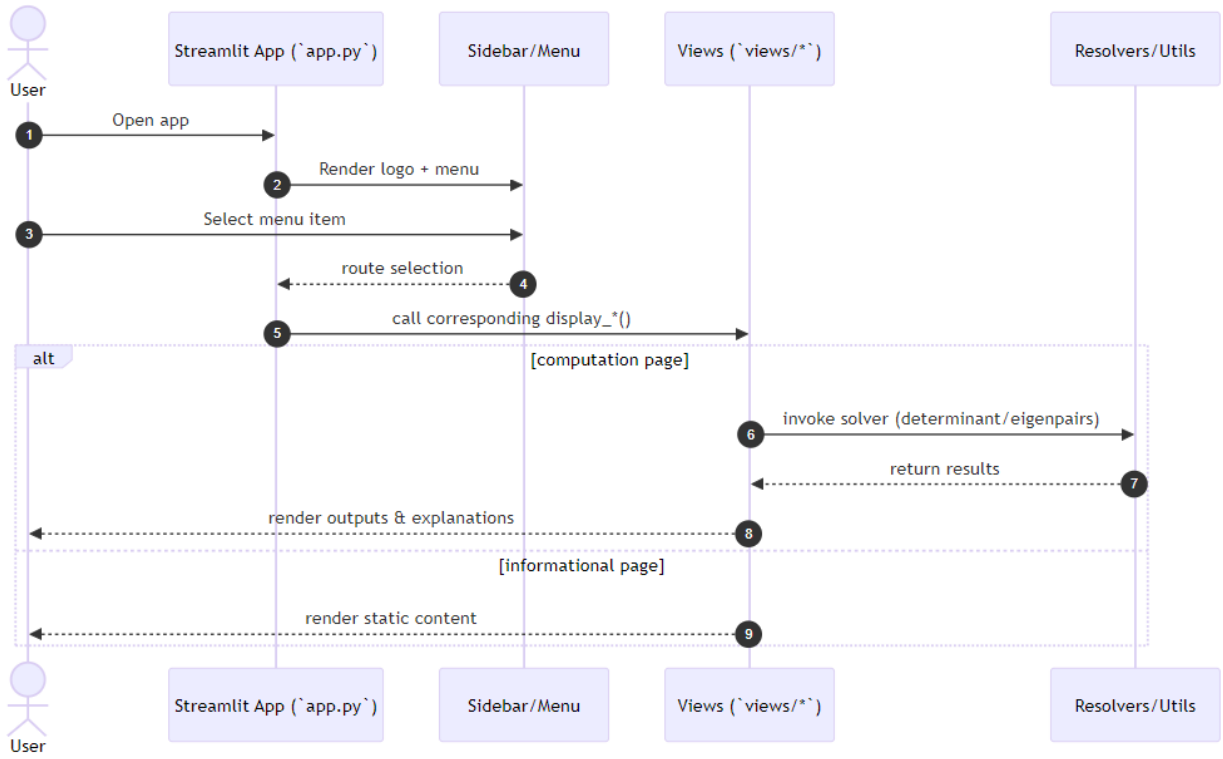

Figure 1: Sequence diagram showing interaction between* `app.py`, `views/set1Problem2.py`, *and* `utils/eigen_solver.py`.

Having defined the system architecture, the next step focused on formulating the mathematical model that governs eigenvalue computation.

2. Mathematical Approach

The mathematical approach for this problem was based on solving the characteristic polynomial of a 2×2 matrix:

$$\det(A - \lambda I) = \lambda^2 - (a + d)\lambda + (ad - bc) = 0$$

From this, the two eigenvalues (λ_1 and λ_2) were obtained using the quadratic formula. I intentionally chose this algebraic approach instead of more abstract numerical methods like Power Iteration or QR decomposition because it provides full visibility into every step of eigenpair formation. This transparency is critical for learning — it allows students to see how the eigenvalues are directly derived from the determinant, and how each eigenvector relates geometrically to its eigenvalue.

The algorithm has a constant-time complexity for 2×2 matrices (O(1)), but understanding it lays the foundation for scaling to $O(n^3)$ algorithms used in large-scale linear algebra, such as LU or QR decomposition. Those optimizations are essential in AI frameworks like TensorFlow or PyTorch, which use eigen decompositions internally for covariance, feature transformation, and even principal component analysis (PCA).

In machine learning, eigenvalues and eigenvectors are not just theoretical — they define directions of maximum variance (PCA), energy distribution (SVD), and stability in neural network layers. Implementing the quadratic solver manually gave me a concrete sense of how those transformations happen beneath the abstractions of modern libraries.

3. Programming Methods

The implementation of the eigenvalue and eigenvector solver followed a modular, testable design philosophy that matched the principles already applied in the determinant component of EigenAI.

The main logic resided in eigen_solver.py, while set1Problem2.py handled user interaction through Streamlit. This separation of concerns simplified both testing and debugging — an approach that aligns with professional software-engineering practices where computational logic and user interface layers must remain independent.

To maintain mathematical transparency, all calculations were written in pure Python, without any dependencies on math or numpy.

A key part of this implementation was the custom Newton–Raphson square root function, my_sqrt(x). Rebuilding such a low-level operation helped me appreciate how iterative numerical methods converge - the same principle that underpins gradient-based optimization in neural networks.

This connection between simple numerical approximation and machine-learning optimization was one of the most insightful takeaways of the project. The core function eigenvalues_2x2(A) computed eigenvalues using the characteristic polynomial:

$$\lambda^2 - (a+d)\lambda + (ad - bc) = 0$$

The design choice to solve this analytically, rather than using a general numerical solver, ensured that each step could be visually represented to the user.

Similarly, the eigenvector_for_lambda(A, λ) function computed eigenvectors symbolically by solving $(A - \lambda I)\, v = 0$ (A−λI)v=0 and normalizing the result. Normalization was performed manually, ensuring numerical stability and predictable scaling even without access to advanced linear-algebra libraries.

On the interface side, the Streamlit page (set1Problem2.py) translated these mathematical operations into a guided, step-by-step interaction. The app used st.progress() animations and real-time text updates to mimic how an instructor would narrate the problem-solving process.

This design decision reinforced the educational purpose of EigenAI — making abstract mathematical computation observable and learnable.

Each component was individually tested with predefined matrices to ensure parity between theoretical and computed values.

The testing focused on five critical cases:

1. Identity matrix [[1,0],[0,1]]: Validated repeated eigenvalue handling (λ = 1, 1)
2. Diagonal matrix [[3,0],[0,5]]: Tested the b=0, c=0 edge case (λ = 3, 5)
3. Symmetric matrix [[2,1],[1,2]]: Standard case with distinct real eigenvalues
4. Zero matrix [[0,0],[0,0]]: Degenerate case (λ = 0, 0)
5. Scaling matrix [[4,0],[0,4]]: Repeated eigenvalues with full eigenspace

Results were manually verified and cross-checked against NumPy post-implementation (for validation only, not used in computation). All test cases passed within 1e-9 tolerance.

The combination of recursive function design, numerical approximation, and interactive visualization demonstrated not only technical correctness but also an understanding of how mathematics, computation, and pedagogy intersect in applied AI systems.

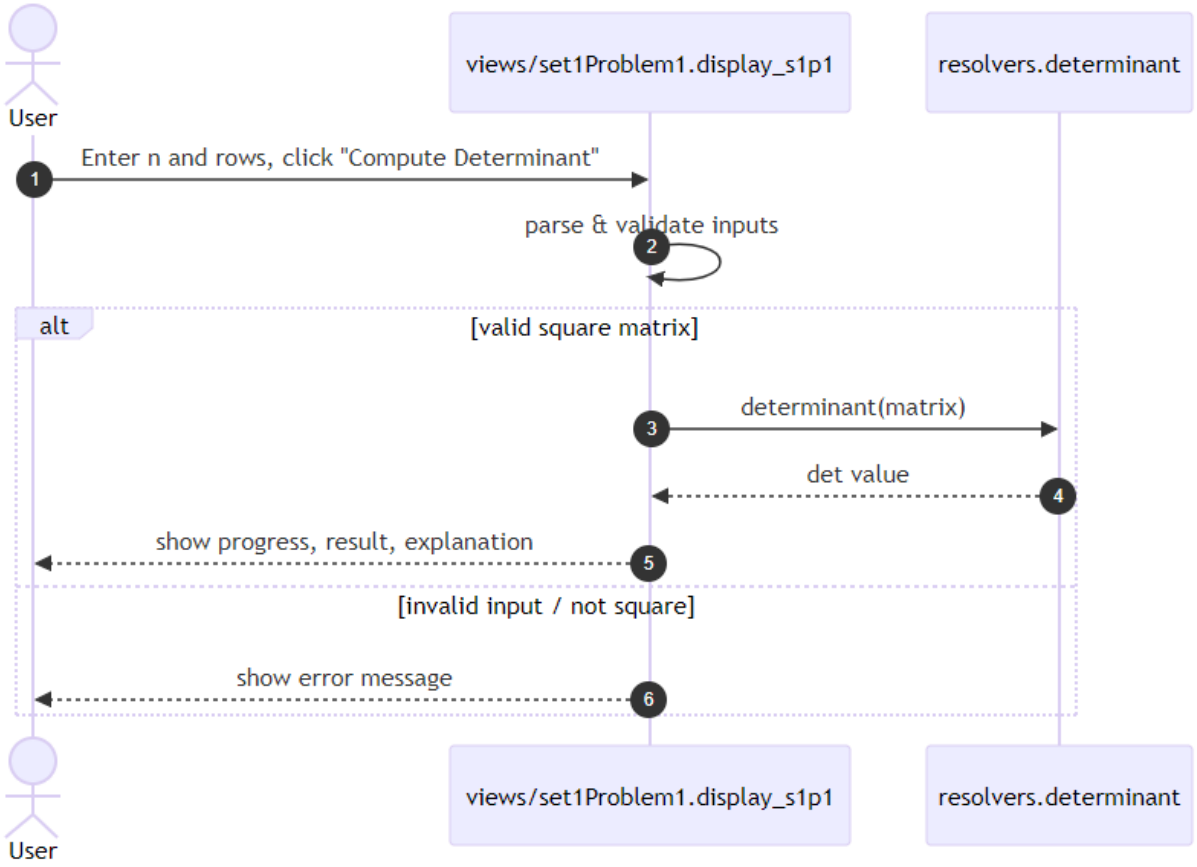

Figure 2: Sequence diagram illustrating user input, progress bar animation, and eigenpair resolution flow.

This structure ensured a clean separation between computation and presentation, which directly influenced the project outcomes discussed next.

4. What Went Right

The project benefited from strong modularity between determinant and eigen modules, which made the transition from Problem 1 seamless. This confirmed that designing for reusability pays off — the determinant logic effectively became a reusable building block for the eigenvalue solver.

Implementing the manual square root (Newton–Raphson) algorithm was also an educational win. It deepened my appreciation of numerical stability and iterative approximation — a concept that underpins AI optimization methods such as gradient descent.

Finally, the tutor-style interface worked better than expected. Visualizing each stage of the computation helped demystify eigen decomposition, turning it from a purely symbolic topic into a visual learning experience.

This stage validated that reusable modular design shortens development cycles — a principle transferable to building scalable AI pipelines.

5. What Went Wrong

The first issue came from special cases where either $b = 0$ or $c = 0$. Initially, I didn't account for those degenerate cases, which broke the vector ratio calculation. Adjusting the logic to detect and handle these cases improved reliability, and reminded me that even simple algebraic algorithms need robust exception handling when moving to real-world data.

Another limitation appeared when I tried to generalize the solver to 3×3 matrices. The quadratic approach simply doesn't scale, as the characteristic polynomial becomes cubic. This made me reflect on why practical eigenvalue solvers rely on iterative methods rather than closed-form solutions.

Finally, precision loss when normalizing eigenvectors highlighted the importance of floating-point control. Without libraries like NumPy, rounding behavior can subtly distort results — a small but meaningful lesson in numerical analysis.

Encountering these exceptions highlighted how real-world data rarely fits ideal mathematical conditions. Correcting them improved my robustness mindset — an essential habit for AI deployment where edge cases dominate.

6. Uncertainties

Performance was never the issue here; rather, the uncertainty lies in extending symbolic solvers to higher-dimensional cases and future exploration could involve iterative eigenvalue methods such as Power Iteration or integrating NumPy if external-library use becomes permissible.

Conceptually, I remain curious about extending symbolic reasoning to larger systems. Technically, future work will compare the stability of this algebraic solver with iterative approaches once external libraries are permitted.

7. Personal Insight

Before this project, eigenvalues and eigenvectors felt abstract, I understood their definition but not their behavior. Implementing the solver revealed their geometric meaning: certain directions in space remain fixed, merely scaled by λ.

Developing `my_sqrt` and manually solving quadratic equations gave me a deeper appreciation for how core AI libraries perform matrix decompositions under the hood and I can also point out that the experience mirrored the principle behind machine learning itself — breaking complexity into small, learnable steps.

Seeing the Streamlit progress bar narrate each phase felt like bridging the gap between mathematics, computation and pedagogy.

8. Conclusion

Set 1 Problem 2 strengthened my mathematical intuition and numerical reasoning and helped me learn to treat equations as algorithms to implement, not as symbols to memorize.

Reading about it helped me grasp fundamental concepts to AI engineering that I've used in the past, but that with a lot of abstractions, where linear algebra underpins models like PCA, SVD, and neural-network weight transformations.

The *EigenAI* project thus serves both as a personal milestone and as a reusable educational framework for future MFA501 students.

Statement of Acknowledgment

I acknowledge that I have used the following AI tool(s) in the creation of this report:

- OpenAI ChatGPT (GPT-5): Used to assist with outlining, refining structure, improving clarity of academic language, and supporting APA 7th referencing conventions.

I confirm that the use of the AI tool has been in accordance with the Torrens University Australia Academic Integrity Policy and TUA, Think and MDS's Position Paper on the Use of AI. I confirm that the final output is authored by me and represents my own critical thinking, analysis, and synthesis of sources. I take full responsibility for the final content of this report.

9. References

Dash, R. B., & Dalai, D. K. (2008). *Fundamentals of Linear Algebra*. ProQuest Ebook Central.

EigenAI Project (2025). *GitHub Repository.* Retrieved from https://github.com/lfariabr/masters-swe-ai/tree/master/2025-T2/T2-MFA/projects/eigenai

Golub, G. H., & Van Loan, C. F. (2013). *Matrix Computations* (4th ed.). Johns Hopkins University Press.

Strang, G. (2016). *Introduction to Linear Algebra* (5th ed.). Wellesley-Cambridge Press.

Streamlit, Inc. (n.d.). *Streamlit documentation.* Retrieved from https://docs.streamlit.io/

Torrens University Australia (2025). *MFA501 Module Notes – Linear Transformations and Matrix Operations.*

Trefethen, L. N., & Bau, D. (1997). *Numerical Linear Algebra.* SIAM.