Summer Fellowship Report

On

Scilab Control System Toolbox development

Submitted by

Samiksha S

Under the guidance of

Prof. Prabhu Ramachandran
Aerospace Engineering Department
IIT Bombay

and mentor

Ms. Rashmi Patankar

July 2026

Acknowledgment

I am deeply grateful to my mentor Ms. Rashmi Patankar for her invaluable guidance, patience, and encouragement throughout my fellowship with the FOSSEE Team at IIT Bombay. Her willingness to explain concepts patiently and push me toward better and more direct solutions played a big role in how much I was able to learn during this fellowship. Working under her mentorship gave me the confidence to take on problems I would have otherwise found intimidating.

I also sincerely thank Prof. Prabhu Ramachandran and Prof. Kumar Appaiah for their invaluable guidance, which helped me understand the larger purpose behind Scilab, FOSSEE, and open-source systems. Their perspective helped me appreciate the value of contributing to open-source tools.

I want to extend my heartfelt appreciation to my friends who were by my side throughout the course of this fellowship. Their encouragement truly did improve my morale.

This internship has been a very meaningful experience. It has, and will continue to, help my academic journey. It has shaped the way I think about programming, documentation, and coordination. I remain committed to continuing my work on the Scilab control system toolbox and other FOSSEE initiatives going forward. There is still a fair amount of work left to be done within Scilab's toolboxes, and I am looking forward to being part of that process in the coming months.

Lastly, I am thankful to everyone who has been a part of this journey, supporting me at every step along the way, and I will carry that gratitude forward as I continue to build on what I have learned here.

Contents

Chapter 1

Introduction

The Free/Libre and Open Source Software for Education (FOSSEE) project is an initiative focused on promoting the use of open-source software tools in education and research. Its goal is to encourage the adoption of freely available alternatives to proprietary software in academic institutions across India. Through activities such as software development, training programs, content creation, and community support, FOSSEE helps integrate FLOSS tools into teaching and learning environments. The project operates under the National Mission on Education through Information and Communication Technology (ICT), supported by the Ministry of Education (MoE), Government of India.

Scilab is a free and open-source platform for numerical computation and scientific programming. It provides a high-level programming environment, efficient numerical computation capabilities, an integrated development environment, and visualization tools for creating 2D and 3D graphical representations.

Scilab is widely used for applications such as signal processing, statistical analysis, optimization, mathematical modeling, and simulation of dynamic systems. Its functionality can be expanded through toolboxes that provide specialized features for areas such as control systems, image processing, data analysis, and scientific simulations. Scilab serves as one of the major open-source alternatives to MATLAB, along with GNU Octave. While Octave focuses more on MATLAB syntax compatibility, Scilab provides a distinct environment with comparable capabilities for scientific and engineering applications.

The FOSSEE Scilab Control System Toolbox is a specialized collection of functions developed and maintained by FOSSEE, IIT Bombay, for the analysis, design, and simulation of control systems. It provides tools that support various aspects of control engineering, including system modeling, time-domain and frequency-domain analysis, controller design, and system manipulation.

The toolbox includes functions for working with transfer functions, state-space models, and linear time-invariant systems. It also provides capabilities for analyzing system properties such as controllability and observability, performing system interconnections, and supporting controller design methods. These features make the toolbox useful for students, researchers, and engineers working in areas such as automation, robotics, aerospace, and process control.

By providing an open-source alternative for control system analysis and design, the Scilab Control System Toolbox helps improve accessibility to engineering software and supports practical learning and research in control systems.

Chapter 2

Control System Toolbox Development

2.1 Overview

The Control System Toolbox forms an important part of Scilab's ecosystem, developed as part of the FOSSEE (Free/Libre and Open Source Software for Education) initiative. It allows engineers, educators, and researchers to model, simulate, and analyze control systems, entirely within Scilab.

The toolbox covers a wide range of functionality, including LTI representation, controllability and observability analysis, and a Riccati equation solver.

Coming into this fellowship, several of these functions did not yet exist in native Scilab and were either missing entirely or only partially implemented, with Octave's control package serving as the primary reference for how they were meant to behave. This meant that anyone relying on the Scilab control toolbox for these operations either had no equivalent to work with, or had to fall back on Octave itself to get the required functionality. Hence this fellowship provided a nice avenue to implement certain functions withing Scilab.

The focus of this internship was to build these functions from scratch as native Scilab implementations, using Octave's versions as the reference for expected behavior. This went beyond simply translating syntax; it meant working out how to express the same logic using Scilab's own functions and conventions, while keeping performance in mind. This effort brought a few clear benefits:

- New functionality in Scilab: These functions simply did not exist for Scilab users before, so building them gave the toolbox capabilities it previously lacked entirely.
- Independence from External Dependencies: Users can now perform these operations directly in Scilab, without needing to install or switch to Octave at all.
- Consistent behavior with Octave: Each function was built and tested to match Octave's output, so anyone moving between the two systems can expect the same results.

The sections that follow walk through the methodology used for this translation work, along with the specific challenges encountered along the way.

2.2 Development workflow

The development process involved studying the existing Octave implementations, translating them into Scilab, and validating the results through testing. The workflow consisted of understanding the function logic, adapting syntax and unsupported features, implementing equivalent Scilab solutions, and refining the code based on test outcomes.

The main steps followed were:

- Studying the reference Octave implementation and documentation
- Translating the logic into Scilab
- Finding existing dependencies made by other fellows
- Recreating missing or unsupported features where required
- Testing and refining the implementation using different test cases

For simpler control system functions, the translation was mostly direct. However, functions involving Riccati equations and staircase forms required additional analysis because their computations relied on SLICOT routines internally. These functions required studying the underlying numerical methods and implementing equivalent logic in Scilab.

2.2.1 Reviewing Octave Source Code

No Scilab code was written until the function itself was properly understood, what it was meant to compute, how it was structured internally, and what algorithm it relied on.

Since these functions come from Octave's control package, the source repository at https://octave.sourceforge.io/control/ served as the primary reference throughout. This groundwork typically covered:

- Working out the math the function was actually built on.
- Flagging any reliance on sub-functions or Octave-only features that Scilab had no equivalent for.
- Getting a rough sense of how hard the eventual translation would be, based on what Scilab could already do natively.

2.2.2 Systematic Line-by-Line Translation to Scilab

Understanding each function before implementing it on Scilab code was really important to this phase. This meant understanding the purpose, structure, dependencies, and underlying algorithm of control system functions.

Finding Octave documentation for certain functions requires one to search online, typically, it will be found in Octave's Control Package Repository. Most functions are written in pure Octave, a .m file, while some may be implemented in C++, a .cc file.

Not every function was easy to unpack. Some, especially those involving Riccati equations and staircase forms, rely on SLICOT, a Fortran library called by Octave rather

than implemented directly. For these functions, Octave only revealed input handling, so understanding them required studying SLICOT's numerical methods and recreating the logic natively in Scilab.

A few recurring mismatches came up throughout this project:

- Block-ending keywords differ: Octave's endif, endfor, and similar terminators all collapse into a single end in Scilab.
- Built-in constants aren't interchangeable: values such as %pi, %i, %T, and %F in Scilab don't behave identically to what Octave uses, and had to be substituted with care rather than assumed equivalent.
- String handling and indexing don't line up cleanly between the two languages, which meant restructuring logic in places instead of translating it directly.
- Error handling was another point of friction, Octave's print_usage() has no Scilab counterpart, so it was replaced with error("message") wherever it appeared.

2.2.3 Handling SLICOT-Based Routines

A significant part of the difficulty in this internship came from functions that rely on SLICOT internally, since SLICOT is written in Fortran 77 and its source is not something Scilab can call into or inspect directly. This meant the actual computation being performed was effectively hidden from view, and simply looking at how Octave passed data in and out of the routine gave no real insight into the underlying algorithm.

Piecing together what these routines were actually doing took several days of work. AI tools were used to help break down the general process, working through the steps a SLICOT routine like sl_sb02od or sl_sg02ad would follow internally, and mapping that against what was already known about the mathematical problem being solved. This helped narrow down that a Schur decomposition-based approach, already available as a built-in Scilab function, followed the same underlying logic as the SLICOT routine and could be used as the basis for the re-implementation.

Even with this direction established, the first drafts produced with AI assistance were not simply accepted as-is. Several passes were needed to check that the logic had not been shortcut or oversimplified along the way, for instance, skipping edge cases the original SLICOT routine handled, or producing a result that matched only for the specific test inputs used rather than the general case. Each version was refined and checked against Octave's actual output before being considered complete, ensuring the final Scilab implementation stayed faithful to what SLICOT was doing rather than an approximation of it.

2.2.4 Testing and Validation

After implementing a function, the same set of inputs was run through both the Octave and Scilab versions and the outputs compared directly. Existing Octave test cases were reused where available; for functions without ready-made tests, or where SLICOT logic had been reconstructed rather than translated directly, inputs were built manually to cover edge cases like singular matrices or unusual system dimensions.

This process took multiple passes for most functions, with mismatches, incorrect assumptions about default behavior, or numerical differences from using Schur decomposition instead of SLICOT's native routine. Each function was revised and re-tested until its output matched Octave's consistently, and these same test cases were later reused to write the documented examples for each function.

2.2.5 AI-Assisted Development

AI tools were used throughout this project, mainly to support the translation and debugging process rather than to carry it out independently. Most of the actual implementation and verification remained manual work, but AI assistance was useful in a few specific ways:

- Breaking down unfamiliar routines: Helped explain the general steps a SLICOT routine or an unfamiliar control theory concept was likely following, making it easier to understand what needed to be reimplemented.
- Speeding up initial debugging: Helped narrow down likely causes of a mismatch, such as an incorrect dimension or misapplied formula, before the actual fix was worked out and verified manually.
- Drafting a starting point: Produced an initial version of certain functions or test structures, which was then checked line by line and revised to make sure nothing was oversimplified or skipped.
- Suggesting edge cases: Occasionally pointed out input scenarios that might not have been immediately obvious, which were then tested and confirmed independently.

Even with these benefits, AI-generated output could not simply be trusted as-is. It sometimes produced code that appeared correct but quietly avoided the harder parts of a problem, so every suggestion needed to be checked carefully against Octave's actual behavior. Used this way, as a learning aid and a starting point rather than a substitute for understanding, AI has real potential to lower the barrier for newcomers picking up control systems concepts, and could meaningfully support onboarding more contributors into open-source scientific computing projects like this one.

2.3 Current Status

2.3.1 Documentation Pattern

Documentation for the Scilab functions was modeled closely on Octave's format, since each function is meant to mirror the behavior of its Octave counterpart as closely as possible. Following the same documentation style also makes it easier for users already familiar with Octave to pick up the Scilab equivalents without much friction. Each function's documentation is written directly above its declaration as a comment block, and typically includes the following components, in this order:

- Description: Provides a comprehensive explanation of the function's behavior, including default operation and expected inputs and outputs.

- Calling Sequence: Describes the order and format of parameters required to invoke the function.
- Parameters: Details the expected input types and acceptable values for each argument.
- Dependencies: Lists any other functions that the current function relies on. In several cases, these were functions already written and documented by fellow FOSSEE interns as part of the same toolbox, and referencing their work helped maintain consistency across the codebase.
- Examples: Demonstrates correct usage through practical cases to help users understand how to apply the function effectively. These examples were also used as test cases to verify that the Scilab implementation produced the same output as its Octave counterpart.

Since the functions are being rewritten in native Scilab code but not redesigned from the ground up, the documentation itself needed only minor adjustments from its Octave source, as the core functionality, structure, and usage patterns remain the same.

2.3.2 Functions

Throughout the course of this project, several control system functions were successfully implemented and documented. These functions cover different Scilab object types, including transfer function @tf, representations, linear time-invariant systems @lti, and standalone control functions. The work involved understanding the existing Octave implementations, translating the functionality into Scilab, performing necessary modifications, and validating the results through testing. The completed functions and their details are presented in Table 2.1:

S.No	Function Name	Function Name in Octave	Dependencies
1	dare	dare	__sl_sb02od__, __sl_sg02ad__
2	care	care	__sl_sb02od__, __sl_sg02ad__
3	dlqr	dlqr	dare, care, issample, dssdata
4	dlqe	dlqe	dare
5	lqr	lqr	dare, care, issample, dssdata
6	lqe	lqe	lqr
7	dss	dss	__sys_data__
8	ctrbf	ctrbf	__sl_tb01ud__, ssdata
9	obsvf	obsvf	ctrbf
10	Westlandlynx	Westlandlynx	
11	vertcat	vertcat	__sys_group__
12	horzcat	horzcat	__sys_group__
13	plus	plus	__sys_group__
14	minus	minus	__sys_group__
15	filtdata	filtdata	tf, tfdata, issiso, strncmpi, isdt

Table 2.1: List of Implemented Control System Functions

2.3.3 Difficulties Encountered

The translation process came with its own share of difficulties, mainly stemming from internal dependencies, some which happen to be beyond the scope of a project. Some other issues encountered were discrepancies between function names, implementation into other toolboxes such as signal processing toolbox and understanding FORTRAN

- Different default behaviors: Some functions behaved differently between Octave and Scilab by default, requiring careful adjustments to match Octave's expected output, also needed thorough renaming and looking into dependencies as well.
- Unavailable functions: Several sub-functions used in Octave's implementations simply did not exist in Scilab and had to be re-implemented from scratch using equivalent logic.
- SLICOT-dependent functions: A number of functions relied internally on SLICOT, a Fortran 77 library with no direct equivalent or binding in Scilab. This meant trying to understand the algorithm within these functions and implement it as accurately as possible in Scilab, three SLICOT-dependent functions were made during this fellowship.
- Complex data types: Octave supports data types which have no native counterpart in Scilab and needed a workaround.
- Testing difficulties: Several functions had no ready-made test cases, so inputs had to be constructed manually and results compared against Octave to confirm correctness. AI usage was particularly useful and gave good test cases to compare.

Using AI tools for translation and debugging helped speed things up, but brought its own set of problems:

- Finding loopholes instead of solving the problem: AI-generated code would sometimes technically pass a test case while quietly sidestepping the actual logic it was supposed to implement, which made results look correct until tested more carefully.
- Unnecessary additions: Generated code often included extra functions, checks, or complexity that had no equivalent in the original Octave implementation, requiring manual trimming to keep the Scilab version consistent with the source.
- Debugging generated code: Fixing these issues took its own effort, since the code needed to be checked closely against Octave's actual behavior rather than assumed correct.

Despite these challenges, testing against Octave's output remained the deciding factor throughout, ensuring that whatever approach was used, native, SLICOT-derived, or AI-assisted, the final function behaved exactly as expected.

2.3.4 Incomplete Implementations and FIXME Issues

No issues.

2.4 Future Scope

While a substantial portion of the control system toolbox has been ported over the course of this internship, several areas remain open for further development. The kalman function and related filtering routines are a natural next step, since state estimation is a core requirement for many control applications and currently remains incomplete in the Scilab-native toolbox. Extending this work to cover related estimation functions would significantly improve the toolbox's usability for real-world control problems.

Additionally, functions such as tf (transfer function representation) and bode (frequency response plotting) are widely used in control system analysis and were not fully covered within the scope of this internship. Completing native Scilab implementations for these would round out the toolbox considerably, since they are often among the first tools an engineer reaches for when analyzing a system.

Beyond these specific functions, continued testing against the Scilab-Octave toolbox, along with expanding documentation and examples, would help ensure the long-term reliability and adoption of the toolbox within the broader Scilab community.

Chapter 3

Learnings

I have gained several valuable experiences from this internship opportunity, some of which are enumerated below.

1. Technical Skills Enhancement:
    - Gained practical experience in Scilab, Octave, Linux, Git, and GitHub while working on control system toolbox development.
    - Improved understanding of control system concepts, LTI systems, state-space representations, and numerical computation methods.
    - Developed skills in translating Octave code into native Scilab implementations and handling differences between the two environments.
2. Control Toolbox Development Experience:
    - Learned the process of analyzing existing control system functions and recreating their functionality in Scilab.
    - Gained experience in implementing and testing functions related to linear system operations and control system analysis.
    - Developed problem-solving skills by identifying unsupported features and creating suitable Scilab-based solutions.
3. Testing and Debugging Practices:
    - Gained experience in designing test cases to validate the correctness of implemented functions.
    - Learned to compare Scilab outputs with Octave results to identify and resolve inconsistencies.
    - Improved debugging skills through iterative testing and refinement of implementations.
4. Documentation and Code Maintenance:
    - Developed experience in preparing technical documentation for implemented control system functions, including their purpose, usage, and behaviour.
    - Learned the importance of maintaining clear documentation for improving code readability and supporting future development.

- Improved skills in organizing implementation details and recording testing information for open-source projects.

5. Open-Source Development and Collaboration:

   - Gained exposure to the workflow of contributing to an open-source scientific computing project.
   - Learned the importance of writing maintainable code and following structured development practices.
   - Developed better understanding of collaborative development through version control, feedback, and continuous improvements.

6. Adaptability:

   - Learned to revise implementations repeatedly based on feedback from review, adjusting code structure, style, or logic even after a function appeared complete.
   - Gained experience incorporating changing expectations and standards over the course of the internship, keeping earlier work consistent with newer conventions as they were introduced.

7. Soft Skills Development:

   - Improved the ability to communicate technical progress and blockers clearly during regular check-ins with the mentor.
   - Learned to ask focused questions when stuck, rather than spending excessive time on a problem alone.
   - Developed better understanding of collaborative development through version control, feedback, and continuous improvements.

Chapter 4

Conclusion

In conclusion, my internship experience at FOSSEE, IIT Bombay, working on the development and enhancement of the Scilab Control Toolbox and contributing to the Scilab-Octave Toolbox has been a valuable and enriching experience. This internship provided me with the opportunity to work on open-source software development while gaining practical knowledge in control systems, numerical computation, and scientific programming.

The primary focus of my work was translating control system functions from Octave into native Scilab implementations. This involved studying existing implementations, adapting the logic to Scilab, identifying differences between the two environments, and performing extensive testing to ensure accuracy and reliability. Through this process, I contributed to improving the availability and functionality of control system tools within the Scilab environment.

Additionally, my work on functions dependent on SLICOT helped me understand the integration of external numerical libraries and the process of recreating equivalent functionality within Scilab. These efforts contribute towards reducing dependency on external tools and improving the overall usability of the control system toolbox.

Throughout the internship, I developed stronger skills in software development, debugging, numerical methods, and control system analysis. The experience of working on a real open-source project has enhanced my understanding of collaborative development practices and the importance of maintaining reliable scientific computing tools.

I am grateful to my mentor Ms. Rashmi Patankar for her guidance and support throughout the internship. I would also like to thank Prof. Prabhu Ramachandran, Prof. Kumar Appaih, and the FOSSEE team for providing me with this opportunity to contribute to open-source software development.

In conclusion, this internship has strengthened my technical foundation and provided valuable experience in applying theoretical concepts to practical software development. The knowledge and skills gained during this period will be highly beneficial for my future academic and professional work.

Reference

REPOSITORY: https://github.com/s-amdot/scilab-control-system-toolbox-functions/

- https://gnu-octave.github.io/pkg-control/
- https://docs.octave.org/latest/
- https://help.scilab.org/docs/5.3.0/en$_U$S/*index.html*
- https://scilab.in/fossee-scilab-toolbox/control-system-toolbox
- A. Varga, "Numerical Methods for Linear Control Systems"