Semester Long Internship Report

On

Scilab Control System Toolbox Development

Submitted by

Akash S

Under the guidance of

Prof.Kannan M. Moudgalya
Department of Chemical Engineering, IIT Bombay

Mentor

Ms. Rashmi Patankar
Project Manager, Scilab Team, FOSSEE Project, IIT Bombay

Faculty Guide

Dr. Krishnaveni V
Professor and Head, Department of ECE,
PSG College of Technology

September 1, 2025

Acknowledgment

I would like to take this opportunity to express my sincere gratitude to the FOSSEE Team at IIT Bombay for giving me the opportunity to work on the development of the Control System Toolbox in Scilab as part of my internship.

I am deeply grateful to Prof. Kannan M. Moudgalya, Department of Chemical Engineering, IIT Bombay, for providing this wonderful opportunity to students to explore and enhance their learning.

I extend my heartfelt thanks to my mentor, Ms. Rashmi Patankar, Project Manager, Scilab Team, FOSSEE Project, for her constant guidance, support, and encouragement throughout the internship. Her valuable feedback and insights were instrumental in shaping my learning experience.

I would also like to acknowledge the entire FOSSEE Team for fostering a collaborative and inspiring environment that enabled me to grow both technically and professionally.

My gratitude also goes to the Department of Electronics and Communication Engineering, PSG College of Technology, for providing me with a strong academic foundation and for continuously motivating students to explore opportunities beyond the classroom.

Finally, I wish to express my special thanks to Dr. Krishnaveni V, Professor and Head of the Department of ECE, PSG College of Technology, for her constant support and for inspiring me, through her Control Systems course, to explore applications beyond academics, an inspiration that greatly motivated me to contribute to this open-source initiative.

Contents

Chapter 1

Introduction

The Free/Libre and Open Source Software for Education (FOSSEE) project is an initiative that advocates the adoption of open-source software tools to enhance the quality of education in India. Its primary objective is to reduce the reliance on proprietary software in academic institutions by promoting freely available alternatives. Through a range of activities, including training, content creation, and tool development, FOSSEE supports the integration of FLOSS tools into teaching and research. The project also contributes to the development and improvement of open-source software to align with the evolving needs of academia.

FOSSEE is a part of the National Mission on Education through Information and Communication Technology (ICT), an initiative of the Ministry of Education (MoE), Government of India.

Scilab is a free, open-source and cross-platform numerical computation software. It includes a high-level, numerically oriented programming language, an efficient computation engine, an integrated development environment, as well as 2D and 3D visualization capabilities.

Scilab can be used for signal processing, statistical analysis, numerical optimization, and modeling, simulation of explicit and implicit dynamical systems. Scilab capabilities can be extended using a toolbox mechanism such as image processing, data analysis dynamic, fluid simulation.

Scilab is one of the two major open-source alternatives to MATLAB, the other one is GNU Octave. Scilab puts less emphasis on syntactic compatibility with MATLAB than Octave does, but is similar enough to easily transfer skills between the two systems.

FOSSEE Scilab Control System Toolbox is a comprehensive suite designed for the analysis, design, and simulation of control systems, developed and maintained by FOSSEE, IIT Bombay.

It provides a wide array of functions that cover both basic and advanced control engineering topics, enabling users to perform system modeling, time and frequency domain analysis, controller design, and system optimization.

The toolbox includes functions for evaluating system properties such as controllability and observability, as well as for manipulating transfer functions and

designing PID controllers. It also offers tools for matrix operations, system interconnections, and signal generation, which simplify many tasks involved in control system analysis.

With its wide range of capabilities and focus on educational use, the Scilab Control System Toolbox is an invaluable resource for students, researchers, and engineers in fields like automation, robotics, aerospace, and process control.

Chapter 2

Control System Toolbox Development

2.1 Overview

The Control System Toolbox is a vital part of Scilab's ecosystem developed under the FOSSEE (Free/Libre and Open Source Software for Education) initiative. It empowers engineers, educators, and researchers to carry out comprehensive analysis, simulation, and design of control systems—ranging from basic SISO systems to complex MIMO systems—entirely within the Scilab environment.

The toolbox provides a broad set of functions, including tools for transfer function modeling, state-space representation, controllability and observability analysis, time-domain and frequency-domain simulations, and feedback design. These capabilities support both classical control techniques (like root locus, Bode plot, and Nyquist stability criteria) and modern control approaches (like pole placement and optimal control).

During the internship, the focus was on identifying and enhancing areas where the toolbox depended on external environments such as Octave for computation. These dependencies often introduced issues such as reduced computational speed, incompatibility with certain platforms, and difficulty in distribution and maintenance. Many functions were either partially implemented or only served as wrappers that offloaded processing to Octave.

The core objective of this internship was to convert these Octave-dependent functions into fully Scilab-native implementations. This involved not only syntactic translation but also ensuring semantic accuracy and performance optimization. Several advantages were gained through this transition:

- **Improved Performance:** Native Scilab functions eliminate the latency and resource usage involved in communicating with an external interpreter.
- **Independence from External Dependencies:** The removal of the FOSSEE Scilab-Octave Toolbox (FSOT) as a requirement simplifies the installation process and enhances portability.

- **Better Compatibility and Stability:** Native functions are easier to test, debug, and deploy across a variety of systems, especially in academic and research environments where Octave installation may not be feasible.

The development process required a structured approach involving deep code analysis, accurate line-by-line translation, rigorous testing, and continuous refinement. The subsequent sections outline this methodology in detail.

2.2 Development Workflow

The development process evolved as familiarity with the toolbox grew. Below is a general overview of the workflow followed throughout the internship:

- Studying the reference Octave implementation
- Translating logic to Scilab, line by line
- Identifying unsupported or missing features and recreating them in Scilab
- Testing thoroughly and refining based on outcomes

2.2.1 Studying Reference Implementations

Before implementation, it was necessary to understand the purpose, structure, and algorithm of each control system function. For Octave-based functions, documentation and source code were usually available through the Octave Forge repositories.

Key aspects of this phase included:

- Understanding the mathematical logic behind each function
- Identifying which parts used sub-functions or relied on specific Octave features
- Estimating the difficulty of translation based on Scilab's support for similar constructs

2.2.2 Line-by-Line Translation to Scilab

This phase involved adapting the Octave/MATLAB code to Scilab syntax and structure. Although Scilab and Octave share many similarities, they differ in certain key areas:

- **Control flow:** Octave requires explicit block endings like `endif` and `endfor`, whereas Scilab uses just `end`.
- **Constants:** In Scilab, predefined constants like `%pi`, `%i`, `%T` (true), and `%F` (false) differ from their Octave equivalents.

- **String and indexing differences:** Scilab handles string manipulation and matrix operations differently in many cases.

Additionally, Octave's built-in error handling (`print_usage()`) had to be replaced with `error("message")` in Scilab.

This step also required creative adjustments when a direct translation was not feasible.

2.2.3 Handling Missing or Incompatible Functions

During translation, several challenges were encountered due to missing or incompatible Scilab functions. In such cases, alternative approaches were taken:

- **Function reimplementation:** When no equivalent function existed, a new version was implemented using Scilab constructs.
- **Behavioral differences:** Some functions, despite having the same name in both environments, behaved differently. In such cases, the expected behavior was replicated using custom Scilab code.

Occasionally, functions in the reference implementation were written in C++ or used data types like structs or images, which are not fully supported in Scilab. Such instances were handled either by writing algorithmic equivalents or avoiding those implementations altogether.

2.2.4 Testing and Iteration

Testing was an essential part of the workflow. In many cases, test cases were found at the end of the reference implementation's source file. When not available or insufficient, custom test cases were written to validate the correctness and robustness of the Scilab version.

Multiple calling scenarios and edge cases were considered to ensure comprehensive coverage. Each function underwent iterative refinement until its outputs aligned closely with the reference outputs.

2.2.5 Leveraging AI for Translation and Debugging

An additional component of the workflow was the use of AI (Artificial Intelligence) assistance tools to accelerate the translation process. AI proved especially useful at multiple stages of development::

- **Algorithm Extraction:** Obtained the underlying algorithm from the provided Octave code through AI explanations, which helped in understanding the logical flow before starting the Scilab implementation
- **Code Translation:** Converted Octave code fragments into Scilab syntax while preserving functionality, with AI suggesting equivalent Scilab functions for unsupported Octave features.

- **Code Structuring:** After writing the initial Scilab version, AI was used to improve indentation, formatting, and introduce logical naming conventions for identifiers, making the code more readable and maintainable.
- **Conceptual Understanding:** Facilitated comprehension of new technical concepts that were critical to building accurate Scilab functions.
- **Error Analysis and Debugging:** When partial outputs or unexpected results were obtained, AI was used to analyze error messages and identify possible logical or syntactic issues, providing corrected snippets or alternative approaches.

This reduced the time spent searching documentation and allowed more focus on understanding the underlying mathematics of each function.

2.3 Current Status

Following the workflow previously described, I successfully have 25 functions from Octave to Scilab. Each function is accompanied by documentation at the top of its file. Furthermore, in my repository, you can find additional test cases and documentation in a README file for each function.

2.3.1 Documentation Pattern

Since the Scilab functions are developed to replicate the behavior of their Octave counterparts, it is logical to refer to Octave's documentation format when documenting these functions.
In Scilab, the documentation is included alongside the function declaration as a large comment block. Each function's documentation typically consists of the following components:

1. **Calling Sequence**: Describes the order and format of parameters required to invoke the function.
2. **Parameters**: Details the expected input types and acceptable values for each argument.
3. **Description**: Provides a comprehensive explanation of the function's behavior, including default operation, expected inputs and outputs, and relevant dependencies.
4. **Examples**: Demonstrates correct usage through practical cases to help users understand how to apply the function effectively.

It is important to note that while the functions are being re-written to use native Scilab code, the associated documentation generally requires minimal changes. This is because the functionality, design, and usage patterns remain consistent with their Octave versions.

All documentation created during this process has been included in the respective function files and summarized in the project's README file. The complete set of developed and documented functions is available in the GitHub repository.

2.3.2 Functions Completed

Throughout the course of this project, several functions were successfully developed and documented. These span across different Scilab object types, including `@iddata`, `@lti`, and standalone control functions. The work encompassed the design, implementation, and thorough documentation of each function. The completed functions are listed in table 2.1.

S.No	Function Name	Function Name in Octave	Dependencies
1	cat_iddata	cat(iddata)	iddata
2	detrend_iddata	detrend(iddata)	iddata, @iddata/size
3	fft_iddata	fft(iddata)	iddata, @iddata/size
4	get_iddata	get(iddata)	iddata
5	size_iddata	size(iddata)	iddata
6	vertcat_iddata	vertcat(iddata)	iddata, @iddata/cat, @iddata/size
7	ctranspose	ctranspose	@lti/size
8	size_lti	size(lti)	
9	ssdata	ssdata	
10	acker	acker	ctrb
11	augstate	augstate	
12	Boeing707	Boeing707	
13	covar_control	covar	@lti/ssdata
14	ctrb	ctrb	
15	dsort	dsort	
16	esort	esort	
17	gensig	gensig	
18	iddata	iddata	
19	mag2db	mag2db	
20	mktito	mktito	@lti/size
21	obsv	obsv	ctrb
22	options	options	
23	pidstd	pidstd	
24	strseq	strseq	
25	thiran	thiran	issample

Table 2.1: List of Completed Functions

2.3.3 Challenges Faced

During the development of Scilab-native implementations of signal processing functions, challenges were encountered at different stages. These can be broadly

categorized into challenges faced before using AI assistance and challenges faced after incorporating AI-based support. The challenges encountered before using AI assistance were:

- **Unavailable Functions:** Some sub-functions used in Octave implementations were not available in Scilab and had to be re-implemented using equivalent logic.
- **Different Default Behaviors:** Functions such as `lyap` behave differently in Octave and Scilab. Special care was needed to align the behavior with Octave's expectations.
- **Unavailability of SLICOT Library:** The Subroutine Library in Systems and Control Theory (SLICOT) provides Fortran 77 implementations of numerical algorithms for computations in systems and control theory. This library was used in various functions of Octave's Control System Toolbox, and translating the Fortran code into Scilab was challenging.
- **Handling C++ Implementations:** Certain Octave functions were implemented in C++, which added complexity in understanding and rewriting their logic in Scilab.
- **Complex Data Types:** Octave supports advanced data types such as iddata which are not natively supported in Scilab.
- **Testing Difficulties:** Some functions did not include ready-made test cases, requiring manual construction of test inputs and comparison with Octave results to validate correctness.

The challenges faced after incorporating AI-based support were:

- **Over-Simplification Risks:** AI-generated translations sometimes oversimplified functions, omitting edge cases or nuances present in Octave implementations.
- **Debugging AI-Generated Code:** AI-based outputs occasionally introduced syntactic or semantic inconsistencies that needed manual debugging in Scilab.
- **Ensuring Consistency Across Functions:** With AI assistance speeding up development, maintaining uniform coding style and ensuring seamless integration with previously developed Scilab functions became an additional responsibility.
- **Validation Still Essential:** Despite AI support, extensive testing against Octave remained essential to ensure correctness, particularly for edge cases and numerical stability.

Chapter 3

Learnings

During the course of my internship, I had several valuable learning experiences that helped me grow technically and professionally. These are summarized below:

1. Technical Skills Enhancement

- Gained hands-on experience with Scilab and Octave, especially in the context of translating and optimizing functions.
- Developed proficiency in writing technical documentation using LaTeX and `README.md` files.
- Learned to debug and optimize native Scilab functions for better performance.

2. Conceptual and Theoretical Learning

- Improved understanding of control systems and their practical implementation through function development and testing.
- Gained deeper insight into mathematical functions like `lyap`, which required careful study and translation of algorithms from Octave.

3. Feedback and Collaboration

- Received regular and constructive feedback from mentors, which helped refine both technical work and documentation.
- Collaborated effectively through peer discussions, which enhanced understanding and brought different perspectives to problem-solving.
- Maintained task tracking using Google Sheets to monitor progress and plan work efficiently.

Chapter 4

Conclusion

In conclusion, my internship at FOSSEE, IIT Bombay, focused on the development and enhancement of the Scilab Control System Toolbox, has been an enriching and transformative experience. This opportunity allowed me to apply and expand my knowledge of control systems, open-source development, and collaborative coding practices.

The central goal of my work was to rewrite and optimize several existing Octave-based functions, making them compatible with and native to Scilab. These efforts were aimed at improving performance, eliminating the dependency on the Scilab-Octave bridge, and contributing directly to a more robust, standalone Control System Toolbox. Over the course of the internship, I successfully implemented and documented numerous functions including `ctrb`, `obsv`, `covar`, `gensig`, and more, enhancing both functionality and usability within the toolbox.

Additionally, I contributed to improving the Scilab-Octave Toolbox's compatibility and helped test and clean up code for smoother integration with current Scilab versions. My work required careful function analysis, rewriting in Scilab syntax, adding comprehensive documentation (in README files), and verifying correctness through rigorous testing.

This experience not only helped me improve my technical proficiency in Scilab, Octave, and Git but also deepened my understanding of control theory, version control practices, and the importance of clean documentation. I also became proficient in using LaTeX for formal report writing.

I am sincerely thankful to my mentor, Ms. Rashmi Patankar, for her clear guidance, consistent feedback, and encouragement. Her mentorship played a vital role in shaping my learning path during the internship. I also thank PSG College of Technology for their support and for encouraging students to participate in such meaningful open-source initiatives.

This internship has solidified my interest in working on open-source tools for education and research. I look forward to continuing my contributions to FOSSEE and applying the skills and experience I've gained in future projects and professional roles.

Reference

- https://github.com/akash-sankar/CSToolboxFunctions
- https://gnu-octave.github.io/pkg-control/
- https://help.scilab.org/docs/5.3.0/en_US/index.html
- https://docs.octave.org/latest/