Winter Internship Report

On

Scilab Signal Processing Toolbox development

Submitted by

Abinash Singh

Under the guidance of

Prof.Kannan M. Moudgalya
Chemical Engineering Department
IIT Bombay

Mentor

Ms. Rashmi Patankar

February 28, 2025

Acknowledgment

I am deeply grateful to my mentor, Ms. Rashmi Patankar, for her invaluable guidance, support, and encouragement throughout my journey with the FOSSEE Team at IIT Bombay.

I began as a Semester-Long Intern in 2024, and this report marks the completion of my Winter Internship (2024). This experience has been enriching, allowing me to enhance my technical skills further and contribute meaningfully to open-source development.

I would also like to sincerely thank Prof. Kannan M. Moudgalya and Prof. Kumar Appaih for their insightful guidance, which has significantly shaped my understanding of open-source systems.

Additionally, I extend my heartfelt gratitude to my friends who supported me in completing the screening tasks for this internship. Their encouragement and assistance were crucial in overcoming challenges and achieving key milestones.

I remain committed to contributing to Scilab and other FOSSEE initiatives. In the coming months, my primary focus will be on advancing the Signal Processing Toolkit, and I look forward to making further meaningful contributions.

I am thankful to everyone who has been part of this journey, offering their support and inspiration along the way. Your unwavering belief in my abilities has made this experience truly fulfilling.

Contents

Chapter 1

Introduction

Scilab is a free and open-source, cross-platform numerical computational package and a high-level, numerically oriented programming language.

It can be used for signal processing, statistical analysis, image enhancement, fluid dynamics simulations, numerical optimization, and modeling, simulation of explicit and implicit dynamical systems, and (if the corresponding toolbox is installed) symbolic manipulations.

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.

IIT Bombay is leading the effort to popularise Scilab in India and the Scilab Signal Processing Toolbox is one of its endeavors toward the cause. This effort is part of the Free and open-source Software for Science and Engineering Education (FOSSEE) project, supported by the National Mission on Education through ICT of the Ministry of Education.

FOSSEE Scilab Signal Processing Toolbox is a comprehensive suite designed for the analysis, manipulation, and visualization of signals, developed and maintained by FOSSEE, IIT Bombay.

It offers a wide range of functions that cover fundamental and advanced signal processing techniques, including filtering, spectral analysis, and time-frequency analysis.

Users can implement various types of digital filters (such as FIR and IIR), perform Fourier transforms, and analyze the frequency content of signals. The toolbox also supports wavelet transform methods, which are essential for non-stationary signal analysis.

With its intuitive interface and extensive testing, the Signal Processing Toolbox is a powerful tool for engineers, researchers, and ed- educators working in fields like telecommunications, audio processing, and biomedical signal analysis.

Chapter 2

Signal Processing Toolbox Development

2.1 Overview

The toolbox is comprised of an amalgamation of various functions (listed in the directory FOSSEE-Signal-Processing-Toolbox/macros/) of mainly two categories: functions that perform the required computation natively in Scilab, and functions that pass the input from the user to be processed by Octave.

To convert functions of the latter type to the former type has been the primary goal of this internship project. The motivation for this is multifaceted, but a few are listed as follows:

- Calling Octave's signal processing function using Scilab requires an intermediary toolbox, the FOSSEE Scilab-Octave Toolbox. This makes FSOT and Octave a dependency for the Scilab Signal Processing Toolbox, which is non-optimal.
- Using Scilab-native code for computations is much more performant than having Octave deal with them and simply uses Scilab as an interface.
- The FOSSEE Scilab-Octave Toolbox is still in development and hence, somewhat lacking in features. For example, any functions that have boolean values, structs, or graphs/images as input or output cannot be accessed using the toolbox. This significantly diminishes its usefulness while handling signal processing functions.

This process of re-writing functions in native Scilab code involves some steps, which are explained from here onwards.

2.2 Development Workflow

The workflow underwent significant changes as I gained experience during this internship. Here is a summary of the workflow:

- Reading the Octave implementation
- Translating line by line
- Comparing the functions and writing missing ones
- Testing and iterating

2.2.1 Reading Octave Implementation

This analysis is crucial for planning our code translation process from Octave to Scilab. It's important to consider that certain sub-functions in Octave may not have direct equivalents in Scilab.

To find the Octave documentation for a desired function, search online and determine which package it belongs to. Typically, functions will belong to one of two packages:

1. Octave's signal package: You can access the source code intended for translation at https://octave.sourceforge.io/signal/ Most functions are written in pure Octave, although some may be implemented in C++.
2. Octave's core package: Download the source code of the latest version of Octave and search for the desired function within the downloaded folder.

The outcomes of this process include:

- Estimation of time constraints.
- Identification of sub-functions that are not available in Scilab.
- Assessment of the complexity involved in the translation process.

2.2.2 Line By Line Translation

This task requires meticulous line-by-line translation for successful adaptation. The focus is primarily on syntax differences between Octave and Scilab.

Here are some key differences to note. If you encounter difficulties, refer to the documentation for Octave and Scilab:

- In Octave, if, else, for, while, and switch statements should be closed with endif, endfor, endwhile, and endswitch, respectively. In Scilab, you simply use "end".
- Constants like! , pi, eps, j, true, and false are defined differently: in Octave as !, pi, eps, j, true/false, and in Scilab as ˜ %pi, %i, %eps, %T /%F.

- "print_usage" function is not available in Scilab so replace it with some error("message")
- The overall syntax is quite similar, but these differences are important to keep in mind.

2.2.3 Comparing Inbuilt Functions And Writing Missing Ones

Now that you have identified which sub-functions are present in the code and checked their availability in Scilab, there is still a possibility of encountering errors. Scilab and Octave functions with the same name may behave differently.

The best approach is to consult the documentation for both platforms and adjust the implementation accordingly. If the differences are significant, consider writing a wrapper function or creating separate implementations.

Octave Scilab equivalents

Octave	Scilab
length(a)	max(size(a))
string	s̃tring
end	$

Another example is When comparing the max and min functions between Scilab and Octave, it's important to note a difference: Octave computes min/max along the columns of a matrix, whereas Scilab computes max/min over the entire matrix.

Paying attention to these nuances is crucial to avoid potential errors.

Now, regarding sub-functions that are unavailable in Scilab, you can follow a similar workflow: consult Octave's documentation, obtain the source code, and translate it to Scilab.

Challenges may arise with functions implemented in C++. In such cases, you can extract the algorithm from the C++ code and attempt to implement it directly in Scilab or utilize Scilab's C++ API. Alternatively, consider creating a wrapper C++ function to interface with the code and dynamically link it to Scilab. However, I don't recommend this if you do not have a good command of C++.

2.2.4 Test And Iterate

This is a crucial but often tedious part of the workflow. To lighten the load, you can find test cases at the bottom of the Octave source code files. Attempt these first, and If you encounter difficulties running some test cases, don't worry—create your examples instead.

Compare the outputs of these examples with those from Octave and your implemented function. Ensure to write diverse test cases covering different calling sequences, aiming for at least one example for each type.

2.3 Previous Status

Following the previously outlined workflow, I successfully translated the following during my semester-long fellowship in 2024 :

- 27 functions independently.
- 2 functions in collaboration with another intern.

In total, our team had translated approximately 60 functions.
Each function is accompanied by documentation at the top of its file and test cases at the bottom.

2.4 Current Status

My winter Internship in 2024 with FOSSEE continued this work and achieved the following:

- 19 more functions are ported from octave to Scilab.
- assigned issues and to do to 16 functions that I was not able to port during the winter internship.

2.4.1 Documentation pattern

Since the functions are intended to mirror their implementation in Octave, it's prudent to use Octave's documentation as a reference for documenting the corresponding Scilab functions.

Additionally, consult Matlab's documentation for these functions, as Octave may have certain bugs. If your test cases fail despite following all steps, compare your code's output with Matlab's to troubleshoot further.

The documentation for a function sits just below the function's declaration, and is written as one big comment block. There are four components to a function's documentation:

1. Calling Sequence: The order of evaluation for the function for the set of given parameters.
2. Parameters: The expected values for the function's parameters. This section outlines the type as well as the acceptable values for these parameters.
3. Description: A comprehensive description of the function. This section outlines default behaviors, expected input and output, and how to interpret them, dependencies and other such data
4. Examples: An example to demonstrate the correct usage of the function.

It is worth noting that in the process of re-writing functions to use Scilab code, the documentation part does not require a lot of modifications because the intended behavior for the function is, most of the time, already well-documented and in line with their Octave counterparts.

All of my work is available in my GitHub repository GITHUB

2.4.2 Functions Completed During Semester Long Internship 2024

S.No	Function	Dependencies/custom functions
1	fftn	
2	fht	
3	fft1	
4	fft21	
5	fftconv	
6	ifft1	
7	ifft2	
8	ifftn	
9	idct2	
10	idct1	idct1
11	idst1	dst1
12	czt	fft1 , ifft1
13	xcorr2	
14	shanwavf	
15	rceps	fft1 , ifft1
16	pulstran	
17	hilbert1	fft1 , ifft1 , ipermute
18	grpdelay	fft1
19	pwelch	fft1
20	tfe	pwelch
21	mscohere	pwelch
22	cpsd	pwelch
23	cohere	pwelch
24	arch_fit	autoreg_matrix , ols
25	arch_test	ols
26	spectral_xdf	fft1
27	spectral_adf	ifft1 , fft1
28	unwrap2	ipermute
39	cplxreal	cplxpair , ipermute

2.4.3 Functions Completed During Winter Internship 2024

S.No	Function	Dependencies/custom functions
1	besself	besselap bilinear postpad prepad sftrans tf2zp zp2tf
2	bitrevorder	digitrevorder
3	cell2sos	
4	sos2cell	
5	cummax	
6	cummin	
7	ellip	ellipap sftrans zp2tf
8	ncauer	ellipap
9	periodgram	fft1 hamming
10	impz	fftfilt
11	iirlp2mb	
12	marcumq	
13	freqz	fft1 postpad unwrap2
14	invfreq	
15	invfreqz	invfreq
16	invfreqs	invfreq
17	findpeaks	
18	impinvar	deconv residue
19	invimpinvar	impinvar

2.4.4 issues and possible Improvements

- **pwelch** : (semilogx,semilogy,etc) and other plot functions can't handle negative values
- **arch_fit and arch_test** : singular matrix exception handling can be improved using try-catch statements.
- There is some bug in the argn() function because of which we get 1 even if our output args are zero. Temporary fix: functions with plot will plot if there is 1 output arg.
- Documentation can be improved with practical examples

Chapter 3

Xcos Architecture Analysis

3.1 Overview

Xcos is a graphical editor within the Scilab environment designed for modeling and simulating hybrid dynamical systems. Its architecture is composed of three primary components: the Xcos Editor, the Scicos Compiler, and the Simulator.

3.2 Architecture Components

- **Xcos Editor**: The graphical user interface (GUI) for creating and editing Xcos diagrams. Implemented in Java.
- **Scicos Compiler**: Translates Xcos diagrams and block definitions into simulation code. Developed using the Scilab language and OCaml
- **Simulator**: Executes the generated code to perform the simulation. A large C program leveraging computational functions written in C, Fortran, or C++.

3.3 Component Interactions

The workflow is as follows:

1. User creates a model in the Xcos Editor. It is converted to a .xcos file with the help of interface functions.
2. The Scicos Compiler processes the model to generate simulation code.
3. The Simulator executes the generated code to produce simulation results.

Xcos employs a unique approach to simulation where the computational functions for various blocks are primarily written in C/FORTRAN and dynamically linked to the Xcos/Scilab environment.

This means that the actual simulation computations are not performed by Scilab directly, but rather by these external C/FORTRAN functions.

This approach is chosen for efficiency and performance reasons, as C/FORTRAN is generally more suitable for computationally intensive tasks compared to interpreted languages like Scilab.

While it is possible to write computational functions in Scilab, this is mainly recommended for prototyping purposes. For production-level simulations, C/FORTRAN functions are preferred due to their superior performance.

The use of dynamically linked C/FORTRAN functions allows Xcos to leverage the strengths of these languages while still maintaining the flexibility and ease of use of a graphical modeling environment.

3.4 Miscellaneous details

Xcos blocks rely on two essential functions for their operation:

- **Interface Function** – Defines the block's properties and behavior.
- **Computation Function** – Executes numerical computations during simulation.

These functions work together to define, simulate, and execute the block's logic within the Xcos environment.

3.5 Interface Function in Xcos

The **interface function** (also called the Block Definition Function) is responsible for:

- Defining block properties (inputs, outputs, parameters).
- Setting the graphical representation of the block.
- Specifying how the block interacts with other elements in Xcos.
- Associating the block with its corresponding computation function.

The interface function registers the block in Xcos but does not perform computations. That is handled by the computation function. This function is not required if we somehow create an Xcos file . Xcos on cloud have a parser for making the Xcos file.

3.6 Computation Function in Xcos

The **computation function** (also called the Simulation Function) is responsible for:

- Processing input values and computing outputs.

- Executing mathematical operations.
- Handling state updates in dynamic systems.
- Managing real-time execution when needed.

3.7 Execution Flow in Xcos

During a simulation, the following sequence occurs:

1. The user creates a block using the interface function.
2. Xcos initializes the block, setting up inputs, outputs, and parameters.
3. The simulation engine calls the computation function for each time step.
4. The computation function processes the input and computes the output.
5. Xcos updates the simulation and proceeds to the next step.

This cycle continues until the simulation is completed.

3.8 Conclusion

- The **interface function** defines the block's appearance, ports, and properties.
- The **computation function** performs numerical calculations during the simulation.
- Computation can be done in **Scilab** (prototype ony) or **C** (faster execution for computational tasks).
- Well-defined interface and computation functions enable efficient use of blocks within Xcos simulations.

3.9 Summary

Our objective was to explore the integration of Octave's functionality with Xcos and its potential support for other programming languages.

However, since the computational aspects of Xcos rely primarily on C/Fortran routines, there is no significant performance advantage in integrating Xcos with another platform.

From a user perspective, though, integrating Xcos with other platforms could provide a more user-friendly modeling tool for Octave users and those utilizing other languages.
This integration could enhance accessibility and usability while maintaining Xcos's robust simulation capabilities.

Chapter 4

Learnings

I have had a lot of great experiences from this internship opportunity, some are enumerated below.

1. Technical Skills Enhancement:
    - Gained proficiency in Scilab, Linux, dynamic linking, c++ Octave, Git, and GitHub.
    - Developed advanced coding skills like testing and documentation.
    - Improved understanding of technical concepts and practical applications.
2. Feedback and Continuous Improvement:
    - Learned to receive and act on constructive feedback.
    - Gained an understanding of the importance of continuous learning and improvement.
    - Developed the ability to self-assess and seek growth opportunities.
3. Professionalism and Work Ethic:
    - Developed a strong sense of professionalism in a workplace setting.
    - Learned the importance of punctuality, reliability, and accountability.
    - Gained experience in maintaining a professional demeanor in various sit– uations.
4. Adaptability and Flexibility:
    - Learned to adapt to new environments and changing circumstances.
    - Gained experience in managing multiple tasks and shifting priorities.
    - Developed resilience and the ability to thrive in a dynamic work environment- menu.

Chapter 5

Conclusion

My internship experience at FOSSEE, IIT Bombay, has been an enriching and intellectually stimulating journey. Having initially joined as a Semester-Long Intern in 2024, this Winter Internship (2024) has allowed me to further deepen my expertise and make meaningful contributions to the Scilab Signal Processing Toolbox.
Throughout this phase, I have focused on improving the functionality, efficiency, and usability of the toolbox while reinforcing my understanding of open-source development. A key aspect of my work involved enhancing and optimizing signal processing functions within Scilab. This included refining existing implementations, ensuring compatibility, and improving performance to reduce dependencies on external toolkits. Additionally, I contributed to resolving issues within the Scilab-Octave Toolbox, which I was not able to fix because of changes in the Scilab API. Also, the Scilab-Octave interface had various limitations and was inefficient in terms of memory and performance Beyond these contributions, I also conducted an architectural analysis of Xcos to explore its integration with programming languages and Octave. This investigation aimed to assess how Xcos can be leveraged for advanced simulations and computational workflows, potentially enhancing its interoperability with external tools and expanding its application scope. Looking ahead, I remain committed to continuing my contributions to FOSSEE projects, particularly in advancing the Signal Processing Toolbox. I am deeply grateful to my mentor, Ms. Rashmi Patankar, for her continuous guidance, encouragement, and support. Her insights have been instrumental in my growth throughout this internship. I also extend my sincere thanks to Prof. Kannan M. Moudgalya and Prof. Kumar Appaih for their valuable guidance, as well as my friends for their unwavering support. This internship has not only strengthened my technical and problem-solving skills but has also reinforced my passion for open-source development and scientific computing. I look forward to applying the knowledge and experience gained here to future projects and continuing my journey as an open-source contributor.

Signal processing Toolbox development

Reference

- https://github.com/abinash108/Signal-processing-toolkit-devlopment-
- https://github.com/FOSSEE/fossee-scilab-octave-toolbox
- https://octave.sourceforge.io/pkg-repository/signal/
- https://scilab.in/fossee-scilab-toolbox/signal-processing-toolbox
- http://www.scicos.org/

- Book : Modeling and Simulation in Scilab$_{S}cicoswithScicosLab$4.4