Semester-Long Internship Report

On

Scilab Octave Toolbox

Submitted by

Shagun Katoch
National Institute of Technology, Hamirpur

Under the guidance of

Prof. Kumar Appaiah
Department of Electrical Engineering
IIT Bombay

Mentor

Mr. Rupak Rokade

July 5, 2021

Acknowledgment

The internship opportunity we had with the FOSSEE Team, IIT BOMBAY, was a great chance for learning and professional development. Therefore, we consider ourselves as very lucky individuals as we were provided with an opportunity to be a part of it. We are also grateful for having a chance to meet so many wonderful people and professionals across the country who led us through this internship period.

We are using this opportunity to express our deepest gratitude and special thanks to Prof. Kannan M. Moudgalya, head of FOSSEE team, IIT Bombay, for giving us an opportunity to be a part of this project.

We express our deepest thanks to Prof. Kumar Appaiah, professor in the Department of Electrical Engineering, IIT Bombay, for taking part in useful decisions and giving necessary advices and guidance to make life easier. We choose this moment to acknowledge his contribution gratefully.

It is our radiant sentiment to place on record our best regards, deepest sense of gratitude to our mentor, Mr. Rupak Rokade for the continuous support which was extremely valuable for our study both theoretically and practically and helping us to learn a lot many things.

We perceive this opportunity as a big milestone in our career development. We will strive to use gained skills and knowledge in the best possible way, and we will continue to work on their improvement, in order to attain desired career objectives. Hope to continue cooperation with all of you in the future.

Contents

Chapter 1

1 Introduction

SCILAB is an open-source numerical, programming, and graphics environment available for free from the French Government's "Institut Nationale de Recherche en Informatique et en Automatique - INRIA (National Institute for Informatics and Automation Research)." It is similar in operation to MATLAB and other existing numerical/graphic environments and can be run using a variety of operating systems including UNIX, Windows, Linux, etc. It includes a large number of intrinsic numeric, programming, and graphics functions.

GNU Octave is a high-level, open-source language, primarily intended for numerical computations. It provides a convenient command-line interface for solving linear and nonlinear problems numerically, and for performing other numerical experiments. It is easily extensible and customizable via user-defined functions written in Octave's language, or using dynamically loaded modules written in C++, C, Scilab, or other languages.

1.1 About Scilab Octave Toolbox

FOSSEE Octave toolbox is a toolbox in scilab maintained and developed by FOSSEE(Free and Open Source Software in Education), IIT Bombay. It is basically a toolbox which aims to bring the power of octave right inside scilab. So, we can call all the octave functions directly from scilab using this.

The Octave toolbox for Scilab makes use of both the Octave-C++ API and the Scilab-C++ API. Firstly, the input is fetched using the Scilab API and stored (Initially, only int/double/complex data type was supported), the input is then sent to the fun() function where the input data is then passed to Octave's API for computation. The returned data is then checked to see for a successful response or an error. If an error is encountered, a general error is thrown in Scilab (Initial case), or if the computation is successful, then

the output is sent back using Scilab's API. It can solve the following types of problems:

1. Linear algebra problems
2. Finding the roots of nonlinear equations
3. Integrating ordinary functions
4. Manipulating polynomials
5. Integrating ordinary differential and differential-algebraic equations etc.

FOSSEE Octave toolbox mainly uses 2 octave libraries, namely :

1. loctinterp
2. loctave

Along with this, toolbox also uses mkoctfile to compile the C source code in to a dynamic loadable .oct file for octave.

1.2 Version Control

Git is a version control system for tracking changes happening in files of computers which comes along with web-based hosting services for repositories. It can be used to coordinating work among multiple people across the world. We have made the toolbox in our systems as git repositories (local repository) to push it to GitHub (to create a remote repository).

GitHub is a web-based hosting service for version control using Git which offers all of the distributed version control and source code management (SCM) functionality of Git as well as adding its own features. We have used GitHub for version control using Git. We used Github for two more major purposes that are:

1. Linking it with Travis CI, the continuous integration tool, where we can deploy and test our projects which are in Github.
2. Adding Doxygen, which is the de-facto standard tool for generating documentation from annotated C++ sources.

The file ".travis.yml" in Github indicates the Travis CI to test our projects whereas the "Doxyfile" indicates the file to configure Doxygen documentation.

1.3 Travis CI

Travis CI is a continuous integration tool, where we can deploy and test our projects which are in Github. We have linked our repository to our Travis CI account and then we initiated a build in Travis. Travis just looks into the .travis.yml file in the repository, and based on the script that we have mentioned, it builds our project.

```
language: cpp
os: linux
dist: focal
arch: arm64-graviton2
addons:
  apt:
    packages:
      - scilab
      - octave
      - build-essential
      - liboctave-dev
      - octave-signal
      - octave-struct
      - octave-communications
      - octave-strings

script:
- cd $TRAVIS_BUILD_DIR/
- cd src/
- bash make.sh
- cd $TRAVIS_BUILD_DIR/
- scilab-cli -f tests/test.sce
```

Figure 1: The basic configuration of the travis engine

We have built the test.sce of the toolbox in Travis CI, which is used to validate the functions present in the toolbox.

On successful completion of the build we get an indication of build successful whereas if the build fails, then we get an indication of build failed which indicates that we have to revisit the test file to fix the errors or bugs. After every build, be it successful or failed builds, we get notification mails

from Travis regarding the commit information and the build status. So each time, we triggered the builds by entering the desired commit messages of the git repository, to test the toolbox.

1.4 Doxygen

Doxygen is a popular tool to document the code which can be used to generate code for a variety of languages. It is great at generating the documentation for the class definitions (the member variable, methods, etc.), class hierarchies (inheritance hierarchy), etc. But, it does not do much with documenting the algorithm (which is typically what you have in your .cpp files). There are two main steps in using Doxygen:

1. To use Doxygen, we write comments in code using the format that Doxygen understands. The comments are included in the header files (.h) files. But, we should also comment code in your .cpp files, though Doxygen won't use them extensively.

2. Then, you simply run Doxygen, which generates an html folder (in our case, inside the doc folder in Github) with the index.html file in it. The documentation for the code is now in an easy to read html file.

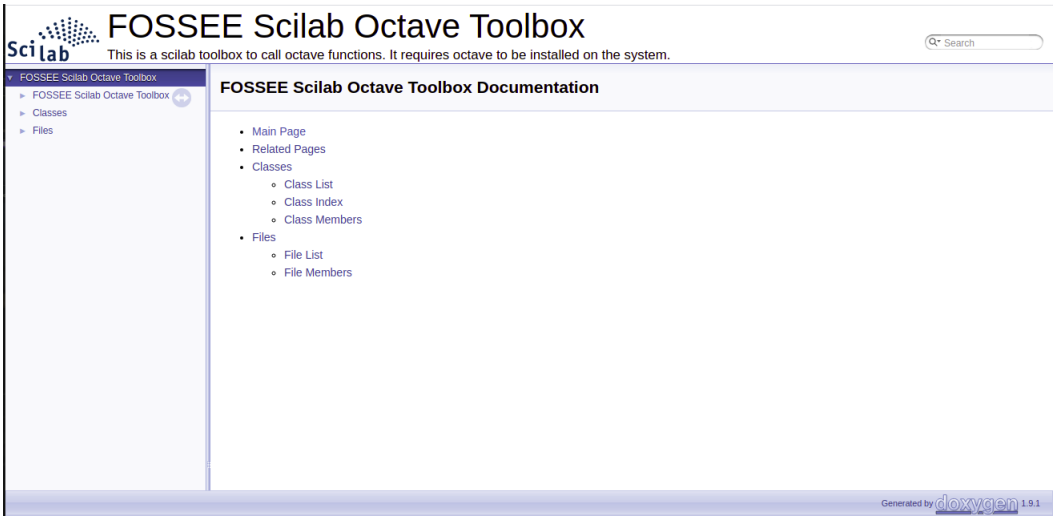

Figure 2: Overview of Doxygen documentation

Chapter 2

2 Structure of Scilab Octave toolbox

Followings are the list of visible files and folders in the main directory of the fossee-scilab-octave-toolbox of Scilab:

- builder.sce
- loader.sce
- demos
- etc
- jar
- cleaner.sce
- unloader.sce
- README.md
- .travis.yml
- Doxyfile
- help
- doc
- locales
- macros
- sci_gateway
- src
- tests
- thirdparty

2.1 File Structure

2.1.1 builder.sce

This file builds the macros, help and the loader.sce files. Type the command exec builder.sce in the scilab console to execute this file. Both these have to be executed every time, to load the toolbox for usage.

2.1.2 loader.sce

It is basically a file that calls the function scilab_octave.start present in the etc folder. This has to be executed first on opening the toolbox, using the command exec loader.sce in the scilab console.

2.1.3 demos

It contains demo files for the toolbox.

2.1.4 etc

etc directory contains the initialization and finalization script of the toolbox which runs at the beginning and termination of the toolbox. They are executed while executing the loader and unloader files.

2.1.4.1 scilab_octave.start

The name of the initialization script for a toolbox is the name of the toolbox followed by ".start". It is executed when we run the loader.sce file. It's purpose includes :

1. Load function libraries from macros directory.
2. Load gateway and shared libraries form sci_gateway and thirdparty directory.
3. Load help from help directory.
4. Load demos from demos directory.

2.1.4.2 scilab_octave.quit

The name of the finalization script for a toolbox is the name of the toolbox followed by ".quit". It is executed when we run the loader. It's purpose includes :

1. Unlink the toolbox libraries.
2. Remove any preferences that were set by the toolbox.

2.1.5 jar

This folder has a scilab_en_US.jar file. This file is basically a Java Archive package file format typically used to aggregate many Java class files and associated metadata and resources (text, images etc.) into a file for distribution.

2.1.6 cleaner.sce

This file is generated by builder.sce. On executing this file, we actually delete the loader.sce and the unloader.sce files. One caution to the users is that do not edit this file.

2.1.7 unloader.sce

It is used to unload the toolbox.

2.1.8 .travis.yml

This file is included to facilitate the builds we can trigger in Travis CI. Travis CI provides a default build environment and a default set of steps for each programming language. We can customize any step in this process in .travis.yml. .travis.yml can be very minimalistic or have a lot of customization in it. So each time when a build is triggered by the user, an new scilab instance is opened in the terminal without any gui (scilab-cli) and executes test.sce in the tests folder of the toolbox (because of the -f flag used here).

2.1.9 Doxyfile

Doxygen uses a configuration file to determine all of its setting. This configuration file is a free-form ASCII text file with a structure that is similar to that of a Makefile, with the default name Doxyfile. It is parsed by Doxygen and essentially, consists of a list of assignment statements. Each statement consists of a TAG_NAME written in capitals, followed by the equal sign (=) and one or more values.

2.1.10 help

The help section that covers all the functions that the toolbox currently consists of.

2.1.11 macros

Macros folder contains scilab function files. Files with extensions other than sci will not be compiled when the builder is run. Scilab macros can be:

1. A Scilab function file which returns the result after computation.
2. A Scilab function which calls a C, C++ or FORTRAN code.

3. A Scilab function which calls a binary library.

2.1.12 sci_gateway

Now the sci_gateway directory contain all necessary files to create the builder for the primitive octave_fun. For this, the builder file is in the sci_gateway/cpp/ directory, this builder (named builder_gateway_cpp.sce) creates the new shared libraries to link the compiled C and new Scilab interface routines and generates a loader. This loader file calls the addinter function to load dynamically the shared library.

2.1.13 src

The src directory contain all source code files for the octave_fun function i.e. to call octave from c and then pass the result to scilab.

2.1.14 tests

This folder contains 2 tests file that are used to validate all the functions present in the toolbox macros. One is the demo.sce file which is used for the demo purposes whereas other is the test.sce file which is executed in the Travis CI (Continuous Integration).

2.1.15 thirdparty

It contains the header files and dynamic link libraries for the fun function for both, Linux and Windows.

2.2 Guidelines to use the toolbox

The toolbox is be available at **https://github.com/FOSSEE/fossee-scilab-octave-toolbox**. The users are requested to go through the README.md file prior using the toolbox.

Else, the users are requested to follow the following steps:

- **Prerequisites for Linux**

    1. sudo apt-get install build-essential (117 MB download)

2. sudo apt-get install liboctave-dev (103 MB download)
3. sudo apt-get install octave
4. sudo apt-get install scilab
5. Now, install the required octave packages using the below command in linux terminal-sudo apt-get install octave-pkg name
   For example, to install signal package in octave, do-sudo apt-get install octave-signal

- **Prerequisites for Windows**

1. Download and Install Scilab 6.0.1 x64 from Scilab.org
2. Download and Install Octave 4.4.1 x64.
3. Install Mingw Tollbox for Scilab (https://atoms.scilab.org/toolboxes/mingw/0.10.5).
4. Create an user variable called 'OCTAVE_HOME' with value equal to the installation directory of Octave.

Now follow the folllowing steps to build and load the toolbox.

1. Clone this repository as it is.
2. Go to the main folder and execute the builder.sce using exec builder.sce.
3. Execute loader.sce using exec loader.sce and start using the functions in the toolbox.

This step should be repeated every time you restart the Scilab to load the toolbox again. Once the toolbox is built and loaded by following the steps mentioned above, we can verify the functioning of the toolbox by executing the test.sce by exec tests/test.sce.

Chapter 3

3 Contributions

3.1 Task 1

3.1.1 Problem Statement

Capturing the original octave error messages and returning them to scilab-This toolbox basically makes use of both the Octave-C++ API and the Scilab-C++ API. So, firstly, the input is fetched using the Scilab C++ API, which is then sent to the fun() function where the data is passed to Octave's API for computation using the feval function. The returned data is then checked to see for a successful response or an error. Now earlier, if an error was encountered in the octave side of this toolbox, it threw only following errors, instead of the specific error by the octave:

- Octave interpreter exited with status = ...
- error encountered in Octave evaluator!
- Octave unable to process!

```
--> octave_fun("idivide",10)


Octave unable to process!
Correct usage:
 octave_fun("octave_function",input1,input2,...)
 octave_fun("octave_function",input1,input2,...,optional_input1,optional_input2,.
 octave_fun("octave_function","octave_package",input1,input2,...)
 octave_fun("octave_function","octave_package",input1,input2,...,optional_input1,
```

Figure 3: Original error

3.1.2 Role

Developer

3.1.3 Contributions

The issue was solved by redirecting the octave errors from fun.cpp to sci_octave.cpp using the C++ std::exception and then storing the errors using a stringstream buffer, which basically associates a string object with a stream allowing to read from the string as if it were a stream. This buffer was later retrieved to show the error on the scilab console. The messages returned by octave were further classified as errors or warnings based on the value of status_fun variable and buffer length. The modified workflow of the toolbox is shown in Figure 6.

```
// Capturing Errors and warnings
std::stringstream buffer_err;

// set our error buffer
std::cerr.rdbuf(buffer_err.rdbuf());

// call the fun() function
int status_fun = fun(argptr, funptr);

// grab error buffer contents
std::string err = buffer_err.str();
Scierror(999, "HELLO\n%s", err);
if (!err.empty() && status_fun == 0)
    sciprint("Warning from Octave\n%s", err.c_str());
buffer_err.str("");


if (status_fun == 1)
{
    Scierror(999, "Error from Octave\n%s", err.c_str());
    return 1;
}
```

Figure 4: Capturing Octave errors

```
--> octave_fun("idivide",10)

Error from Octave
error: Invalid call to idivide.  Correct usage is:

 -- idivide (X, Y, OP)

Additional help for built-in functions and operators is
available in the online version of the manual.  Use the command
'doc <topic>' to search the manual index.

Help and information about Octave is also available on the WWW
at https://www.octave.org and via the help@octave.org
mailing list.
```

Figure 5: Error Resolved

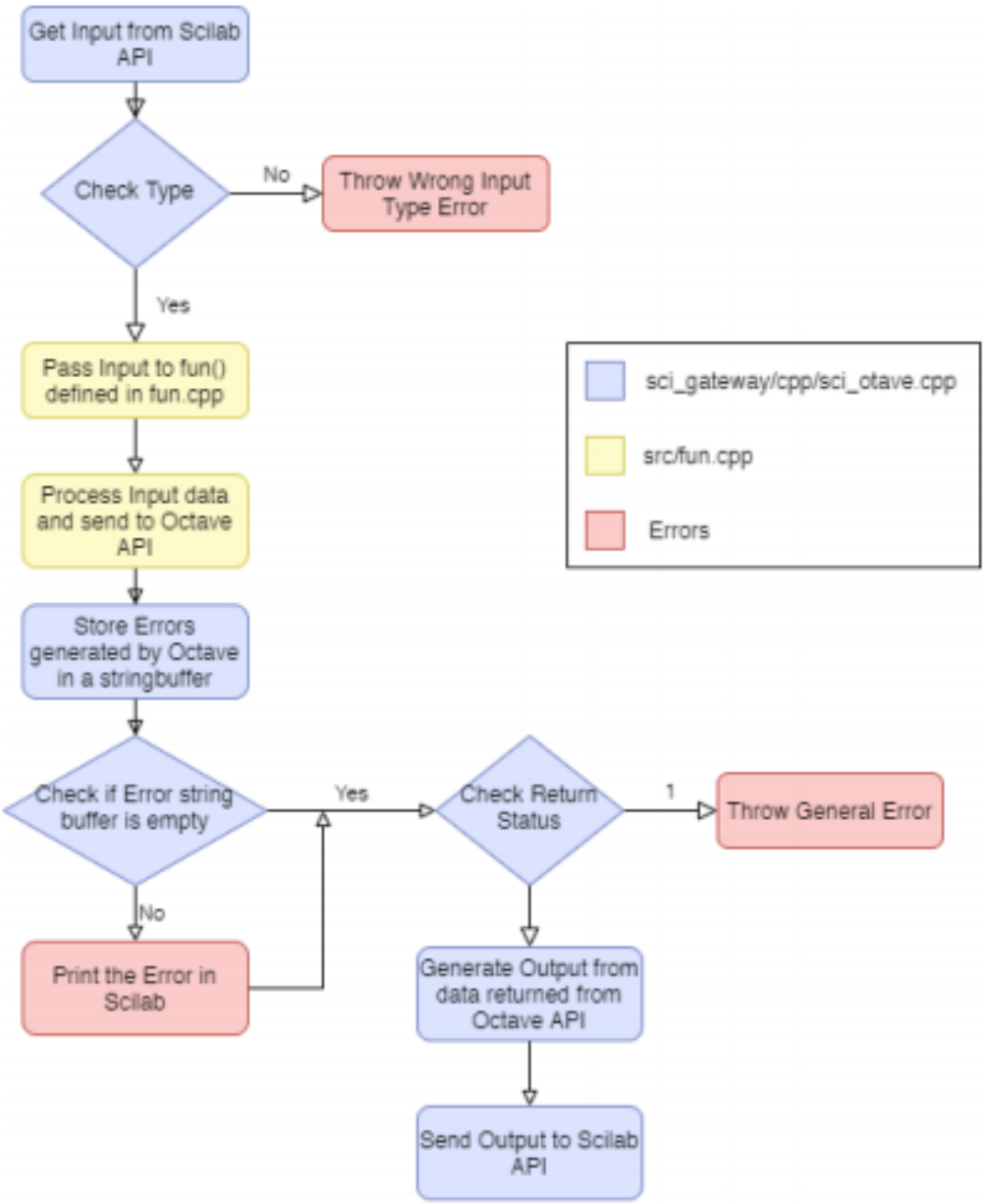

Figure 6: Modified Workflow of Toolbox

3.2 Task 2

3.2.1 Problem Statement

Handling octave functions that represent the input and output data in structure format-Initially, the toolbox only supported int/double/complex/real/string data types and not inputs which consisted of structure data. In this case, the toolbox gave an error of a wrong data type. So, the task was to extend the capability of toolbox to support the structure data types.

```
--> s=struct('date',13,'month','JAN')
 s  =

  date: [1x1 constant]
  month: [1x1 string]


--> s=octave_fun("setfield",s,'year',2021)

octave_fun: Wrong type of input argument 1.
```

Figure 7: Original Error

3.2.2 Role

Manager

3.2.3 Contributions

As manager of the task, I was responsible for conducting timely meetings for the progress update, taking decisions in regard to what needs to be done to meet the goals of our task and make sure that the task is completed on time. So, to solve this issue, the problem was basically divided into 4 parts as follows:

- Capturing the struct data from scilab in sci_octave.cpp and passing it to fun.cpp.
- Retrieving the data in fun.cpp and then passing it to the feval function for computation.

- Taking the output given by feval function and passing it back to sci_octave.cpp.
- At last, passing the data retrieved to display on the scilab console.

All these parts were assigned to each one of us. In regards to this, I was assigned the first part and hence, worked with the Scilab API functions like scilab_getfields and scilab_getStructMatrix2dData to retrieve the struct data given by a user.

```
--> s=struct('date',13,'month','Jan')
 s  =

  date: [1x1 constant]
  month: [1x1 string]


--> s=octave_fun("setfield",s,'year',2021)
 s  =

  date: [1x1 constant]
  month: [1x1 string]
  year: [1x1 constant]


-->
```

Figure 8: Error Resolved

3.3 Task 3

3.3.1 Problem Statement

Extending the toolbox usage to Windows OS-As earlier, the toolbox was limited to Linux OS (Debian/Ubuntu) only, it was required to extend the capability of the toolbox to windows OS also.

3.3.2 Role

Tester

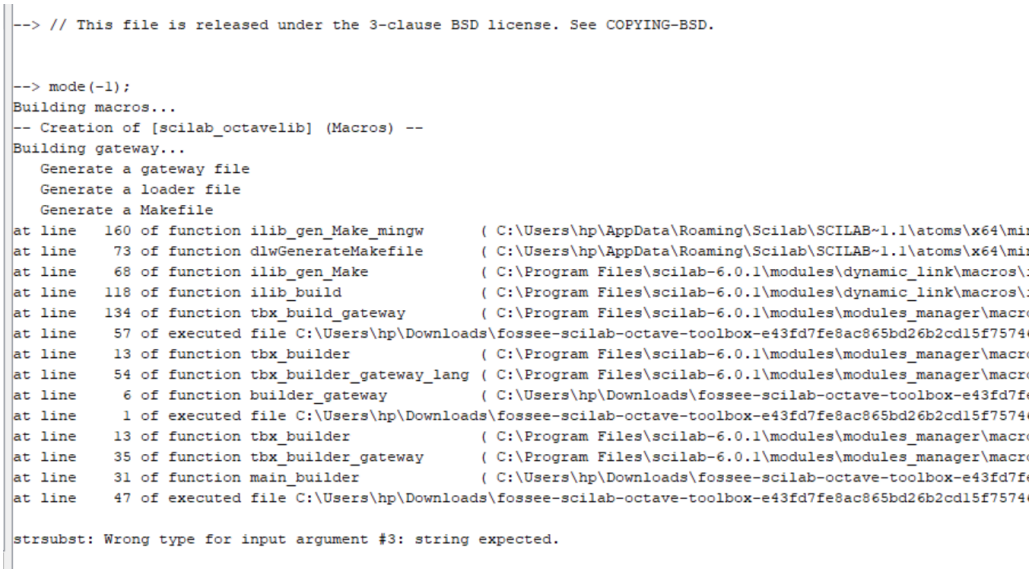

Figure 9: Original Error

3.3.3 Contributions

The builder.sce file had to be edited for windows to include the liboctave-6.dll and liboctinterp.dll files, present in the bin folder of the Gnu octave, which made the build successful, but loading the toolbox was still giving errors. This issue was addressed by adding the path of the bin folder of GNU Octave to environment variables. However, some functions were still not working fine. It was solved by adding the OCTAVE_HOME variable to environment variables so that octave could find all of its functionalities. I also provided the solution in case of the following error:

-library_name.a not found error on running "exec builder.sce".

,which usually happens when you build the toolbox on Windows for the first time or from scratch. To solve this, simply comment the commands:

-octave_lib_dir + "liboctave-6";

-octave_lib_dir + "liboctinterp-6";

in toolbox_root/sci_gateway/cpp/builder_gateway_cpp.sce. Now, run exec builder.sce to get a sucessful build. Uncomment the commands, you just commented and run the builder command once again. This time the toolbox

will build sucessfully. The toolbox still had very minor errors like test.sce script crashing or cleaner.sce failing, but they were also solved eventually.

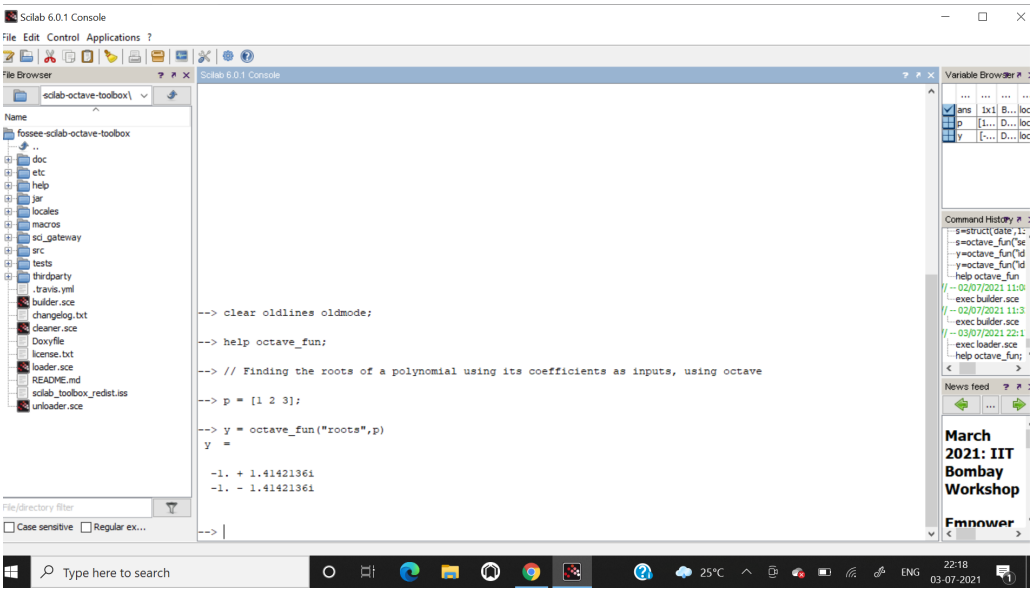

Figure 10: Error Resolved

Chapter 4

4 Conclusion

During the semester-long internship, I was exposed to software and open-source culture and had a very steep learning curve. I learned a variety of concepts and turned them into efficient and implementable ones. I have made a lot of progress and understood the implications of becoming a computer professional and software engineer. I discovered the concept of APIs and created an interface through them. Additionally, it also improved my C's proficiency level. Plus, I learned about the challenges of writing cross-platform software, how your code is affected when the operating system changes, how to modify your code, how to handle errors, and finally, how to fix it. In addition to my technical knowledge, I also learned about time management, critical and analytical thinking, and goal management. Finally, I would like to thank everyone who made my internship such a wonderful and memorable learning experience.

5 References

- https://octave.1599824.n4.nabble.com/Capturing-exception-messages-in-octave-C-API-td4694132.htmla4694163
- https://octave.org/doc/v6.1.0/Structures-in-Oct-002dFiles.html
- https://stackoverflow.com/questions/48226185/c-using-enum-inside-struct
- https://help.scilab.org/docs/6.0.2/en_US/api_struct.html