Dual Degree Project

Virtual Chemical Engineering Labs through Aakash

Aviral Chandra (09D02030)

Guide: Prof. Kannan Moudgalya

DEPARTMENT OF CHEMICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY, BOMBAY

MUMBAI, 400076

23rd October 2013

ACCEPTANCE CERTIFICATE

The project report on “Virtual Chemical Engineering Labs through Aakash” submitted by Mr. Aviral Chandra (Roll No. 09D02030) may be accepted for evaluation.

Place:

Department of Chemical Engineering

IIT Bombay

Date:

____

Prof. Kannan Moudgalya

DECLARATION OF AUTHORSHIP

I, Aviral Chandra, declare that this thesis titled, 'Virtual Chemical Engineering Labs through Aakash' and the work presented in it are in my own words. I have cited and referenced the work which I referred to while development. I confirm that I have abided by all the principles of academic honesty and that any finding opposite to my claims could evoke disciplinary actions. This work was done mainly towards the requirement of M.Tech degree in Chemical Engineering at I.I.T Bombay, I further clarify that

The references contain the research papers , that were consulted

All sources of help have been acknowledged

Signature

____

Aviral Chandra

Roll No. 09D02030

ACKNOWLEDGEMENTS

I take this opportunity to express my gratitude to my project guide, Professor Kannan M Moudgalya under whose esteemed guidance I got an opportunity to learn quite a lot about the field of remote labs and their great utility in the field of Chemical Engineering.

The submission made under this report is a result of many discussions with my guide which motivated me to undertake programming and development with a lot of zeal . Under his able guidance, I have been able to develop developmental skills for which I am extremely grateful.

I would also like to thank Kannan sir for linking me with the FOSSEE team at IIT Bombay. He has been a great mentor, motivator and always provided the right resources and guidance to overcome the roadblocks faced in the project.

I would take this opportunity to also thank Mr. Manoj Gudi who has always been a great co-developer. The development has been a collaborative effort with Manoj who has been a phenomenal support as his knowledge of hardware and low-level implementation is outstanding.

SYMBOLS AND CONSTANTS USED

Gc : Denotes the controller transfer function

G: Denotes the plants transfer function

p: denotes the proportional controller gain

i: denotes the integral mode time constant

d: denotes the derivative mode time constant

a: represents the coefficient of ‘s’ in the numerator in the definition of plants transfer function of first degree

b: represents the constant in the numerator in the definition of plants transfer function of first degree

d0: represents the coefficient of ‘s’ in the denominator in the definition of plants transfer function of first degree

d1: represents the constant in the denominator in the definition of plants transfer function of first degree

f: a constant that specifies the window size

s: a symbolic variable available in scilab

α: denotes the version of Sandhi ready to be released

ABSTRACT

Virtual Labs aims at providing remote access to physical laboratories. Ease of access, ability to learn at one’s own pace and the possibility of reuse make it a very attractive proposition. With the aforementioned benefits remote labs can clearly improve learning objectives.

While the idea of remote labs seems promising, it poses various challenges from the implementation standpoint. Remote labs include experiments which need data acquisition from remote hardware, running algorithms on a remote hardware and performing simulations.

There are various software tools that perform data acquisition, run algorithms on a remote hardware and can run simulations. Most of these are proprietary and hence come at a huge one-time cost and recurring license costs. The use of proprietary technology sounds quite unpromising especially when planning an initiative like remote labs for the masses. It makes the infrastructure unsustainable as the volumes increase.

The NMEICT initiative on virtual labs relies heavily on LabVIEW a proprietary data acquisition tool, also known for its visual programming editor.

This thesis lays down a framework to systematically develop an open source functional clone to LabVIEW. It has been named SANDHI, which comes from Devnagri and means to add or connect. In essence SANDHI is an in-house visual programming editor built exclusively for control system application. SANDHI has been developed in around 20 weeks. This thesis exposes the development cycle which eventually led to SANDHI.

The thesis also mentions that SANDHI is aimed to functionally replace LabVIEW in certain applications in the next development cycle**.**

Keywords: Virtual Labs, NMEICT, LabVIEW, SANDHI, proprietary, open-source

Table of Chapters

LIST OF APPENDICES

LIST OF FIGURES

LIST OF APPENDIX FIGURES

1 SLP Review

This project involves development and deployment of virtual labs for mobile platform. The SLP stage of the project consisted of literature survey, understanding and development of some preliminary tools to help in the main project.

The proposed architecture for the virtual lab in the SLP stage is as under

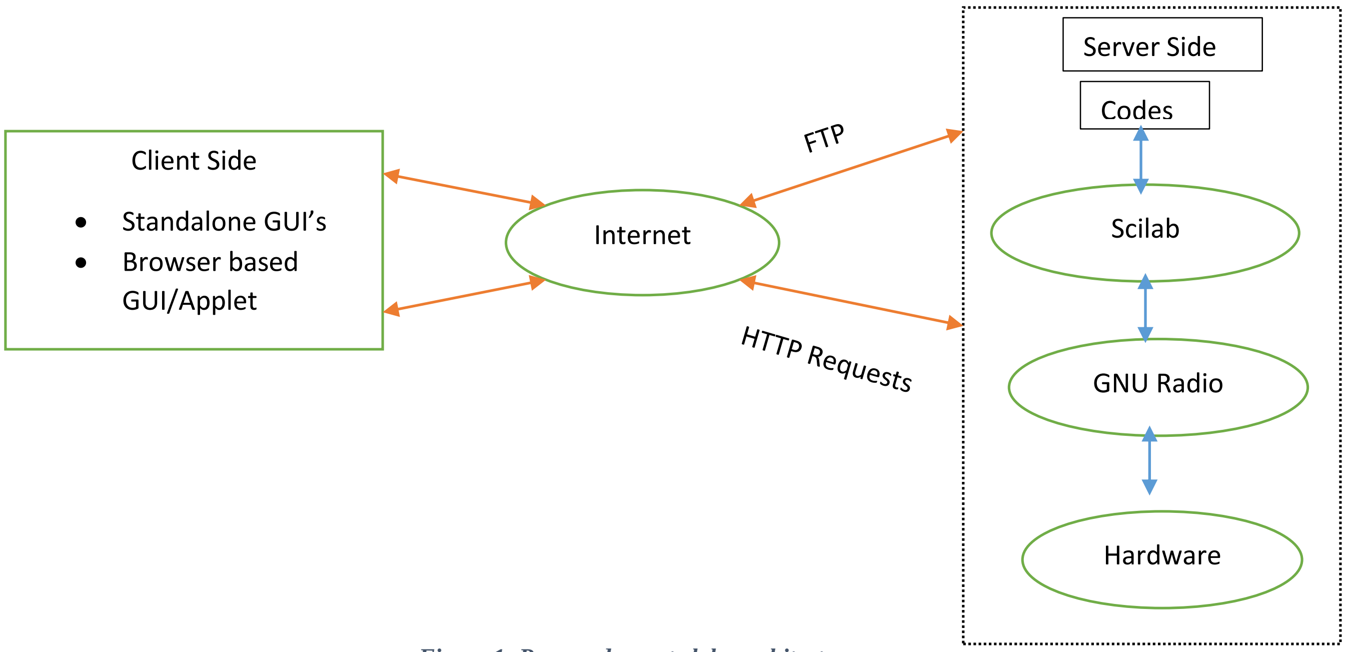


Figure 1: Proposed remote labs architecture

The first problem at hand is identification and development of requisite tools which would help accomplish the final goal.

GNU-Radio was identified as a very promising Data Acquisition tool which provided the following benefits.

- It has a massive driver library, thereby it supports a good range of hardware
- It has a clean GUI, various widgets that can be used to tweak the inputs in real time
- It's possible to implement blocks entirely in python as well as C++
- Its real time response to changes in calculation parameters is extraordinary.

GNU Radio appealed as a promising DAQ tool with a good driver support, clean GUI, and the potential to become a functional visual programming editor for control system applications.

This theses captures the development cycle of GNU Radio which could functionalize it to perform interaction with scilab's computation engine and work as a potential replacement to LabVIEW.

2 Developments

This thesis captures a development cycle which led to control system blocks in GNU Radio, it’s integration with scilab’s computation engine and implementation of feedback. Thereby making it an open source visual programming editor for control system applications and a potential clone to LabVIEW.

2.1 GNU Radio

It is an open source radio implementation which was supposed to be used by the Electrical Engineering community for the purpose of digital signal processing. It has a rich module of implemented device drivers and thereby supports a range of devices. GNURadio is a very promising visual programming tool as it makes it very easy for the developer to abstract his code, and provides a very easy to use framework to the developer. On top of all these advantages it is open source and hence making it potentially very attractive alternative to control system enthusiasts, who use SIMULINK/ LabVIEW for doing rapid prototyping of control schemes.

2.2 Blockly

Though the development started with GNURadio in mind, it was a matter of time before it was understood that the current build of GNU Radio was incapable of supporting feedback.

Feedback is a very vital component when we talk of Control Engineering. Working with the stock scheduler of GNU Radio it looked almost impossible to implement feedback without hard-coding it in the system. Hard-coding is not a good solution to any problem. If the developmental work has to be futuristic, it has to be adaptable and dynamic. Blockly entirely developed at MIT is a visual programming tool, using which school kids can learn programming easily.

Since it looked very intuitive and flexible, it was surely an option to look at. It's view is implemented entirely in Javascript and while it was a great option to work with, the challenges associated with blockly were that it didn't look easy to integrate a computation engine with implemented control libraries(like Scilab) with blockly.

2.3 Xcos

Xcos is a great visual programming editor. It is quite similar to SIMULINK in many aspects. It gives the luxury of ready control libraries in scilab. The development cycle around Xcos was planned, however it had to be stopped because of xcos's rigid framework was not adaptable to UI modifications in real time. It is not capable of supporting basic hardwares and writing device drivers is a huge task. This option was dropped on account of limited development time.

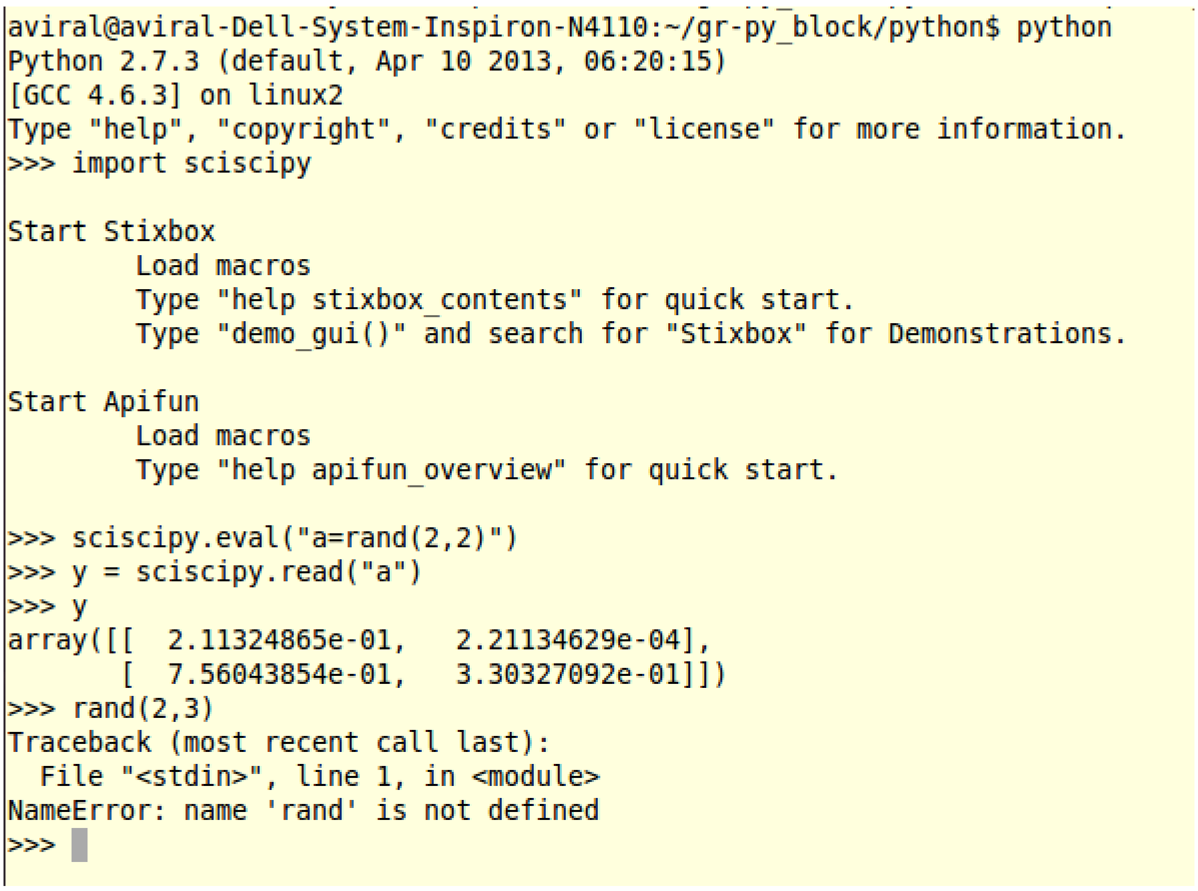
2.4 sciscipy

Sciscipy is an **Application Programming Interface** aimed for **Inter Process Communication** with scilab when in workspace of Python programming language.

The essence of open source is to develop on the work of others. This helps save a lot of time in the development cycle. It can be claimed that it's analogous to building cars and not trying to re-invent the wheel. 'sciscipy' is the wrapper which helps us call scilab from python. This API is extremely useful as it helps us import scilab libraries directly in the python workspace.

Sandhi is a pure python implementation and hence this API comes in handy in implementation of control libraries of scilab.

Screenshot of how sciscipy works is given below.



```
aviral@aviral-Dell-System-Inspiron-N4110:~/gr-py_block/python$ python
Python 2.7.3 (default, Apr 10 2013, 06:20:15)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import sciscipy

Start Stixbox
        Load macros
        Type "help stixbox_contents" for quick start.
        Type "demo_gui()" and search for "Stixbox" for Demonstrations.

Start Apifun
        Load macros
        Type "help apifun_overview" for quick start.

>>> sciscipy.eval("a=rand(2,2)")
>>> y = sciscipy.read("a")
>>> y
array([[  2.11324865e-01,   2.21134629e-04],
       [  7.56043854e-01,   3.30327092e-01]])
>>> rand(2,3)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'rand' is not defined
>>> 
```

Figure 2:Sciscipy implementation

As can be seen here the sciscipy helped us import functions of scilab into python workspace. It becomes clear on observing the last command. It shows that rand is not defined in python.

We have clearly taken advantage of a scilab functions to generate output in python workspace. It's extremely convenient to work with these functions to generate plots and connect programming blocks when in python workspace.

We identified the problems behind the broken API. The broken API was a very long-standing issue in the python community. The APPENDIX 3 lists down the complete solution.

2.5 GRAS for Feedback solution

GRAS stands for **GNU Radio Advanced Scheduler,** it has been used in all the developments. It was impossible to implement the feedback with the stock application scheduler. Application Scheduler is responsible for threading, controlling the data flow and managing the use of the computer resources like processor time to various processes.

2.6 Sandhi

This thesis presents development of a novel visual programming framework based on GNURadio. It has been named Sandhi as it means connecting and conveys our idea of connecting various blacks to come up with a robust visual program.

Sandhi is aimed to become a visual programming tool for replacing LabVIEW. It's current form is raw, however if the control community gets excited by this tool, this tool can achieve a lot of what LabVIEW can do. The release of α version of Sandhi is under way

The entire development cycle has been given and it's adaptability for any control library explained. The challenges faced during development have been clearly mentioned, so that a new developer does not face similar issues while trying to develop Sandhi on their own.

3 GNU Radio to Sandhi

This chapter exposes the development cycle which eventually led to Sandhi. It starts from capturing the difference of implementation of same scilab libraries in scilab and python workspace. It then goes on and eventually covers the abstraction of these functions made so that the user has to interact with a well defined system and not get into writing the functions all by themselves. It is very important for a good visual programming editor to be able to implement simple view even for the most complex development. The start of the chapter and the middle sections are just meant to focus on the complexities that arise and how they should be handled in a development cycle.

3.1 Identified Scilab function for implementation in python using sciscipy

a) csim - used for generating continuous time response of linear systems,

b) dsimul is used for generating discrete time response of linear systems.

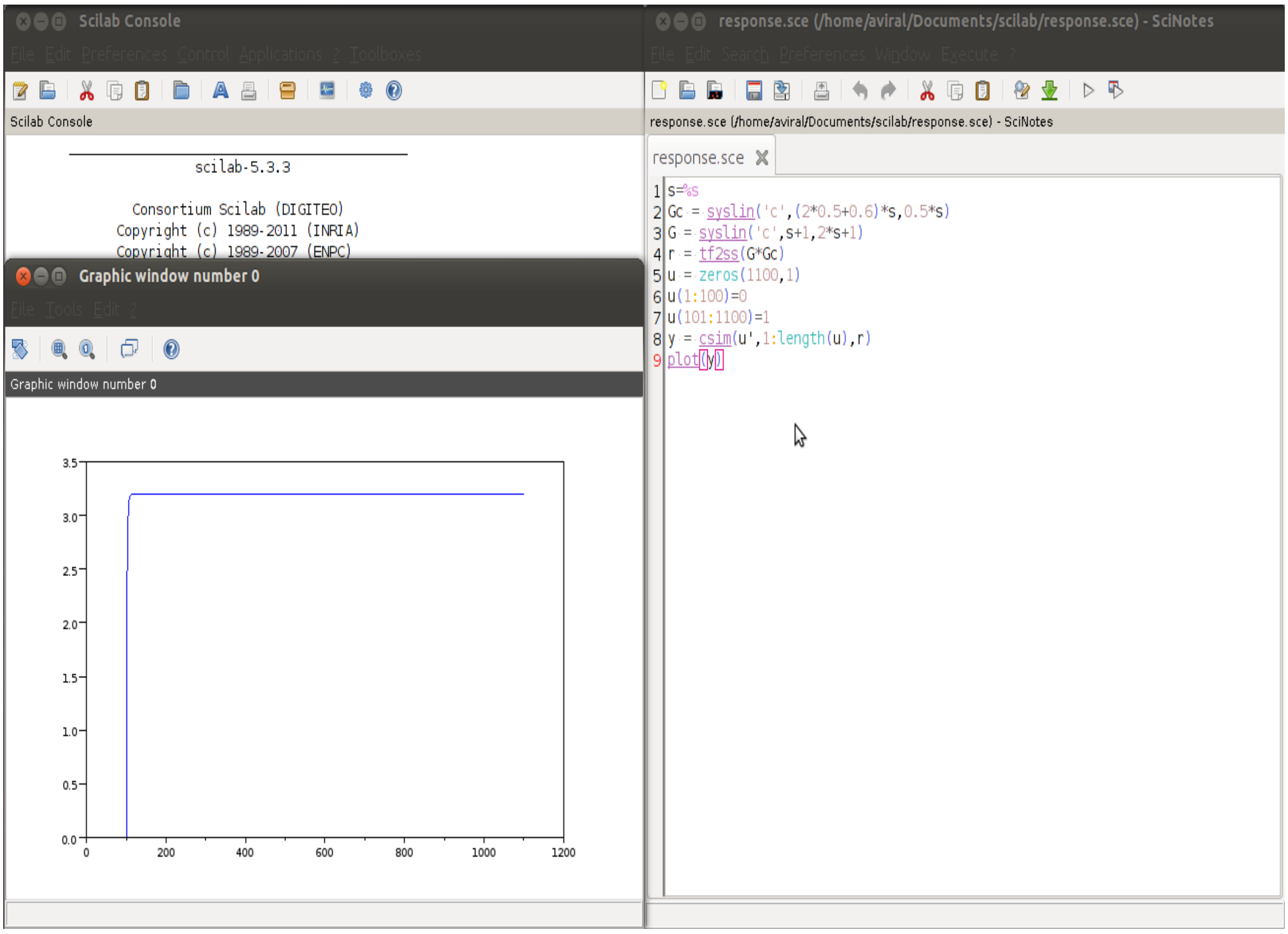
For the purpose of illustration impulse and step inputs are chosen and step and impulse response have been captured. The plots in python workspace have been generated using the python plotting libraries. Section 3.1 has 2 snapshots which cover these implementation.

Implementation in scilab

The scilab implementation was a normal scilab code and a plot was generated. The scilab function used for this purpose was '*syslin*'. *Syslin* is used to define linear systems. The symbolic computation was invoked by the first statement (s=%s). Two linear systems were defined, for the sake of discussion one can refer to Gc as the controller and G as the plant. We took a product of both, converted the product to state space and generated a constinuous time step response.

The same job was imitated in python and as can be seen in the second screenshot, it was done using the wrapper sciscipy and the plots were generated in python workspace using python's plotting libraries. What is to be observed in the second screenshot is that once the csim is defined in python workspace any input can be passed into the csim from the python workspace. This gives us tremendous advantage as the visual programming editor Sandhi has been implemented entirely in python. The source blocks of GNU Radio have been retained in Sandhi and as can be seen in Chapter 5, a proof of concept has been given for using the already implemented signal source blocks for the purpose of studying simulations

3.1.1 Screenshot of scilab implementation


Scilab Console
File Edit Preferences Control Applications ? Toolboxes
Graphic window number 0
File Tools Edit ?
response.sce (/home/aviral/Documents/scilab/response.sce) - SciNotes
File Edit Search Preferences Window Execute ?
response.sce
0
200
400
600
800
1000
1200
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5

```
scilab-5.3.3

Consortium Scilab (DIGITEO)
Copyright (c) 1989-2011 (INRIA)
Copyright (c) 1989-2007 (ENPC)
```

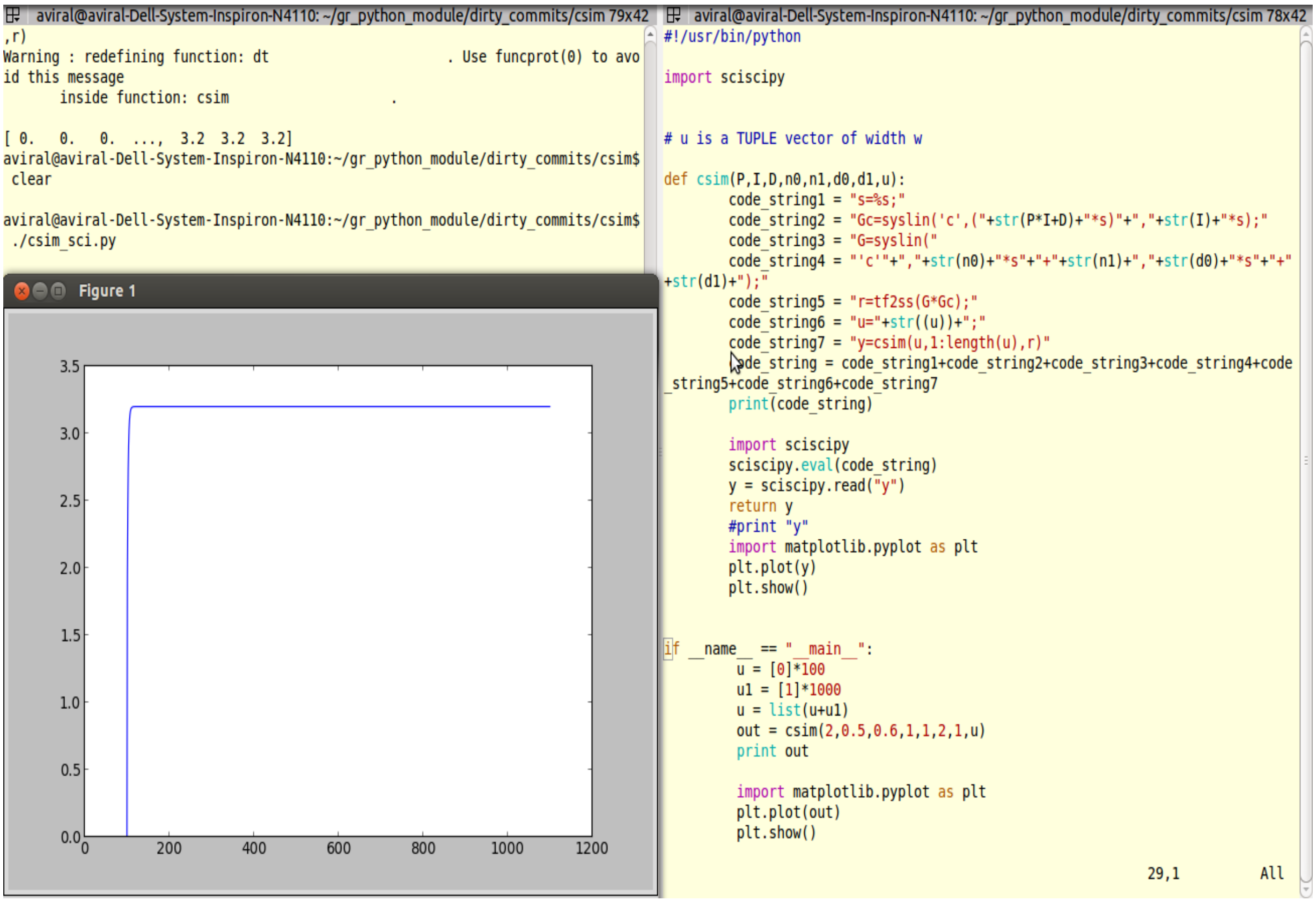
```
s=%s
Gc = syslin('c',(2*0.5+0.6)*s,0.5*s)
G = syslin('c',s+1,2*s+1)
r = tf2ss(G*Gc)
u = zeros(1100,1)
u(1:100)=0
u(101:1100)=1
y = csim(u',1:length(u),r)
plot(y)
```

Figure 3: csim implementation in scilab

A scilab implementation of csim, for a step input. The input had to be hard coded in here. Even when we functionalize it, we still remain in the scilab workspace and the abstraction like a user interface is possible but is not very stable. Even if the abstraction like a UI were to be implemented it would certainly not have had the advantages of advanced scheduler, and hence feedback.

Not going ahead with xcos was due to the fact that xcos has limitations when it comes to real-time data operations, as it's application scheduler is quite obsolete.

3.1.2 Screenshot of python implementation



```
aviral@aviral-Dell-System-Inspiron-N4110: ~/gr_python_module/dirty_commits/csim 79x42
,r)
Warning : redefining function: dt                    . Use funcprot(0) to avo
id this message
       inside function: csim                .

[ 0.   0.   0.  ...,  3.2  3.2  3.2]
aviral@aviral-Dell-System-Inspiron-N4110:~/gr_python_module/dirty_commits/csim$
 clear

aviral@aviral-Dell-System-Inspiron-N4110:~/gr_python_module/dirty_commits/csim$
 ./csim_sci.py
```

```
aviral@aviral-Dell-System-Inspiron-N4110: ~/gr_python_module/dirty_commits/csim 78x42
#!/usr/bin/python

import sciscipy


# u is a TUPLE vector of width w

def csim(P,I,D,n0,n1,d0,d1,u):
        code_string1 = "s=%s;"
        code_string2 = "Gc=syslin('c',("+str(P*I+D)+"*s)"+","+str(I)+"*s);"
        code_string3 = "G=syslin("
        code_string4 = "'c'"+","+str(n0)+"*s"+"+"+str(n1)+","+str(d0)+"*s"+"+"
+str(d1)+");"
        code_string5 = "r=tf2ss(G*Gc);"
        code_string6 = "u="+str((u))+";"
        code_string7 = "y=csim(u,1:length(u),r)"
        code_string = code_string1+code_string2+code_string3+code_string4+code
_string5+code_string6+code_string7
        print(code_string)

        import sciscipy
        sciscipy.eval(code_string)
        y = sciscipy.read("y")
        return y
        #print "y"
        import matplotlib.pyplot as plt
        plt.plot(y)
        plt.show()


if __name__ == "__main__":
         u = [0]*100
         u1 = [1]*1000
         u = list(u+u1)
         out = csim(2,0.5,0.6,1,1,2,1,u)
         print out

         import matplotlib.pyplot as plt
         plt.plot(out)
         plt.show()
                                                       29,1          All
```

Figure 4: csim implementation in python

As can be seen above the two plots are exactly similar, however they are implemented in workspace of two different languages. The sciscipy helped us make ***inter process calls*** to scilab only when we needed it. This wrapper helps us use the scilab's computation engine. It is worth mentioning that at no point in time the user realizes that the scilab's computation engine is being used.

Since Sandhi is deployed entirely in python and it uses an advanced scheduler, the above block would come in handy when we are trying to handle real-time operations.

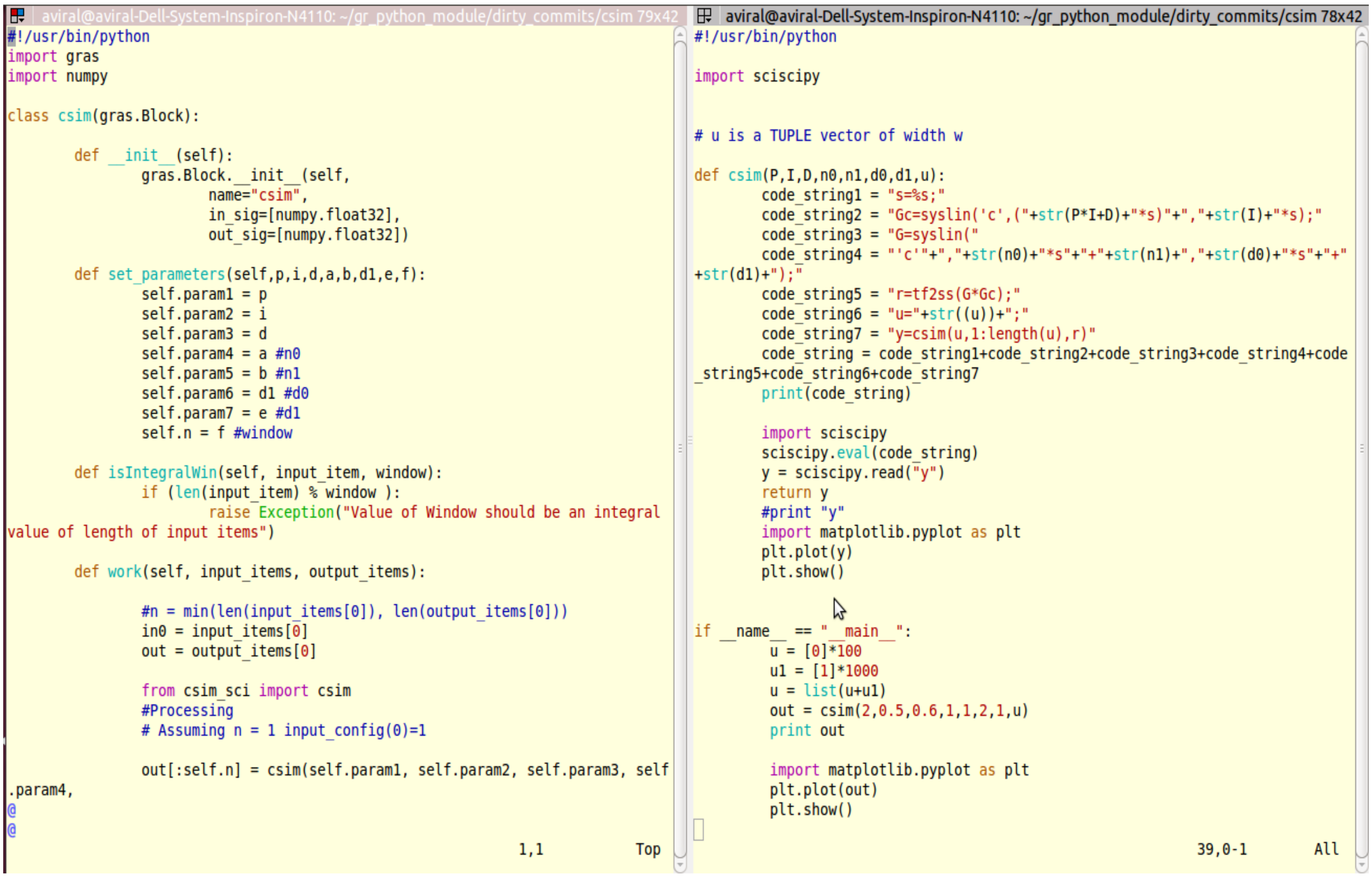
3.2 Abstraction

“In computer science, **abstraction** is the process by which data and programs are defined with a representation similar in form to its meaning (semantics), while hiding away the implementation details”

-Wikipedia(http://en.wikipedia.org/wiki/Abstraction_(computer_science))

It is very important to hide away the unnecessary implementation details, however this report shows all the steps of the 5-month development cycle, which led to the software Sandhi. It can help developers in the future to understand the steps involved while developing a software.

From a preliminary observation one can see that the screen partition to the right has no values. It is the first step in the abstraction of functions. Moreover it is the first step towards synthesis of blocks from raw python implementation of scilab functions.



aviral@aviral-Dell-System-Inspiron-N4110: ~/gr_python_module/dirty_commits/csim 79x42

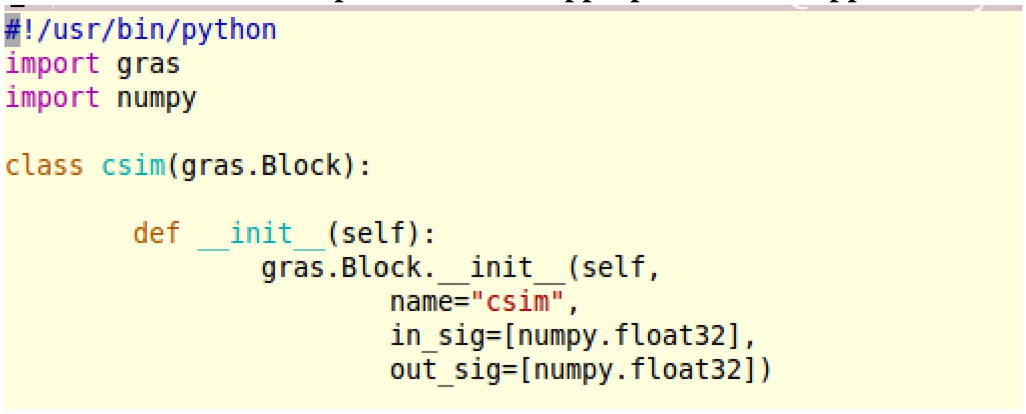
```
#!/usr/bin/python
import gras
import numpy

class csim(gras.Block):

        def __init__(self):
                gras.Block.__init__(self,
                        name="csim",
                        in_sig=[numpy.float32],
                        out_sig=[numpy.float32])

        def set_parameters(self,p,i,d,a,b,d1,e,f):
                self.param1 = p
                self.param2 = i
                self.param3 = d
                self.param4 = a #n0
                self.param5 = b #n1
                self.param6 = d1 #d0
                self.param7 = e #d1
                self.n = f #window

        def isIntegralWin(self, input_item, window):
                if (len(input_item) % window ):
                        raise Exception("Value of Window should be an integral
value of length of input items")

        def work(self, input_items, output_items):

                #n = min(len(input_items[0]), len(output_items[0]))
                in0 = input_items[0]
                out = output_items[0]

                from csim_sci import csim
                #Processing
                # Assuming n = 1 input_config(0)=1

                out[:self.n] = csim(self.param1, self.param2, self.param3, self
.param4,
@
@
                                                                  1,1            Top
```

aviral@aviral-Dell-System-Inspiron-N4110: ~/gr_python_module/dirty_commits/csim 78x42

```
#!/usr/bin/python

import sciscipy


# u is a TUPLE vector of width w

def csim(P,I,D,n0,n1,d0,d1,u):
        code_string1 = "s=%s;"
        code_string2 = "Gc=syslin('c',("+str(P*I+D)+"*s)"+","+str(I)+"*s);"
        code_string3 = "G=syslin("
        code_string4 = "'c'"+","+str(n0)+"*s"+"+"+str(n1)+","+str(d0)+"*s"+"+"
+str(d1)+");"
        code_string5 = "r=tf2ss(G*Gc);"
        code_string6 = "u="+str((u))+";"
        code_string7 = "y=csim(u,1:length(u),r)"
        code_string = code_string1+code_string2+code_string3+code_string4+code
_string5+code_string6+code_string7
        print(code_string)

        import sciscipy
        sciscipy.eval(code_string)
        y = sciscipy.read("y")
        return y
        #print "y"
        import matplotlib.pyplot as plt
        plt.plot(y)
        plt.show()


if __name__ == "__main__":
         u = [0]*100
         u1 = [1]*1000
         u = list(u+u1)
         out = csim(2,0.5,0.6,1,1,2,1,u)
         print out

         import matplotlib.pyplot as plt
         plt.plot(out)
         plt.show()

                                                                 39,0-1        All
```

Figure 5: Creating csim blocks(csim.py left partition) out of csim implementation in python(csim_sci.py right partition)

3.2.1 A brief on development with the appropriate code snippets



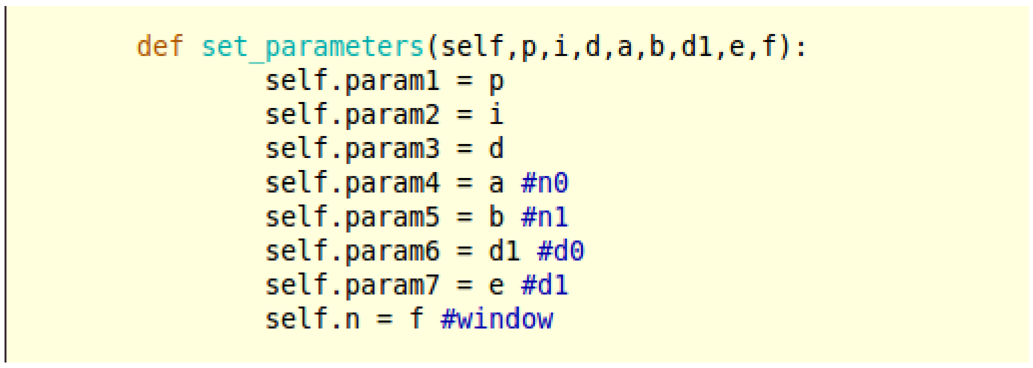
```
#!/usr/bin/python
import gras
import numpy

class csim(gras.Block):

        def __init__(self):
                gras.Block.__init__(self,
                        name="csim",
                        in_sig=[numpy.float32],
                        out_sig=[numpy.float32])
```

Figure 6:gras, numpy and csim blocks

The first line imports the advanced scheduler, this process scheduler is invoked and as can be seen in the third line we explicitly specify that csim is an advanced scheduler block and hence it can use the advanced application scheduler that comes with gras.

The second line imports a powerful python scientific package. Among other things it has a very efficient and powerful N-dimensional array object. It would make our life much simpler as array operations in python blocks having numpy array are extremely convenient.

The method __init__ is a python constructor which creates an empty class object. It is used later to access various methods of the class. It also defines the input/output signature for the numpy array that can be handled by the instances of csim class. Here they are float data types. Self is used to convey within this class



```
        def set_parameters(self,p,i,d,a,b,d1,e,f):
                self.param1 = p
                self.param2 = i
                self.param3 = d
                self.param4 = a #n0
                self.param5 = b #n1
                self.param6 = d1 #d0
                self.param7 = e #d1
                self.n = f #window
```

Figure 7: declaring parameters for csim blocks

As can be seen above set_parameters method is used to declare the parameters to be used by the program/block that implements csim class. It would become clear in the later chapters why the word program/block was used intentionally.

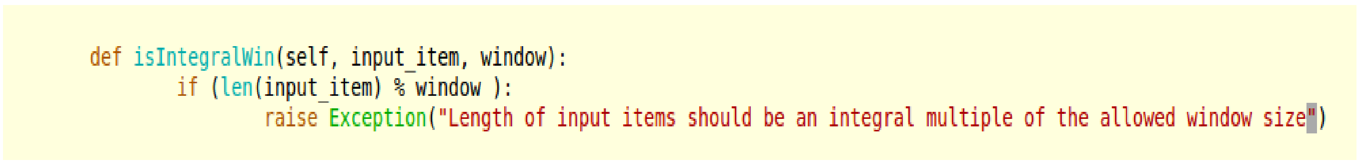
The seven parameters declared here are from the following equation

Equation:

$$\mathbf{Gc} = (\boldsymbol{p} + \frac{\mathbf{1}}{\boldsymbol{i} * \boldsymbol{s}} + \boldsymbol{d} * \boldsymbol{s}) \qquad \text{- Equation 1}$$

$$\mathbf{G} = \frac{\boldsymbol{a} * \boldsymbol{s} + \boldsymbol{b}}{\boldsymbol{d}\mathbf{1} * \boldsymbol{s} + \boldsymbol{e}} \qquad \text{- Equation 2}$$

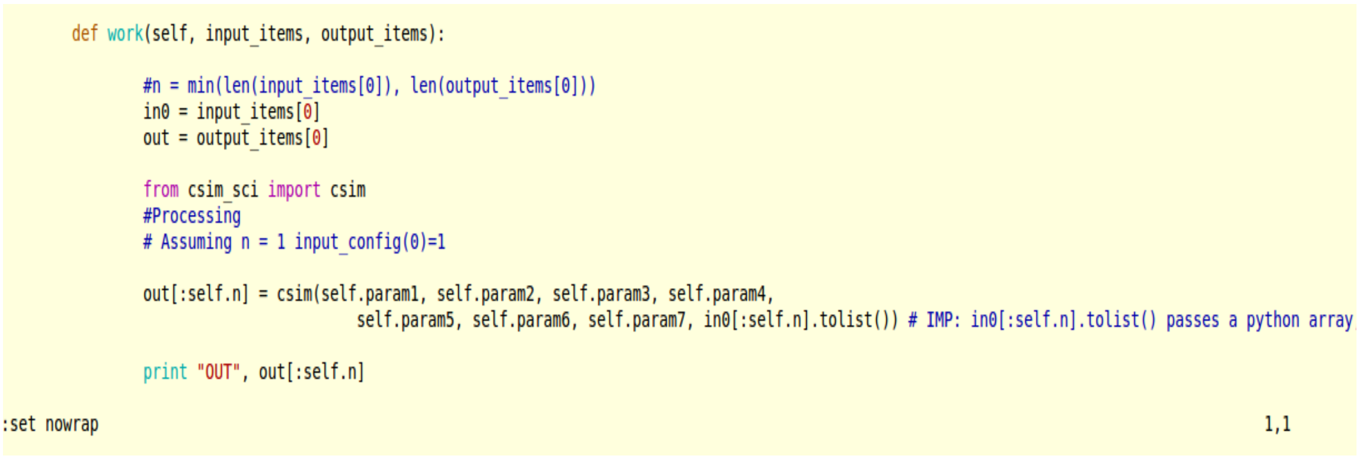
The eighth parameter ‘f’ specifies the i/o mapping. It helps map exact number of inputs to same number of outputs. It could be one input mapped to one output or an input vector of any finite size mapped to output vector of the same size.



```
def isIntegralWin(self, input_item, window):
        if (len(input_item) % window ):
                raise Exception("Length of input items should be an integral multiple of the allowed window size")
```

Figure 8:handling the input items in the array

This snippet implements the logic that length of input items should only be an integral multiple of the window size. It seems only logical to have this condition since it becomes difficult for the application scheduler to handle exceptions which ultimately lead to process halt/ unresponsive programs.

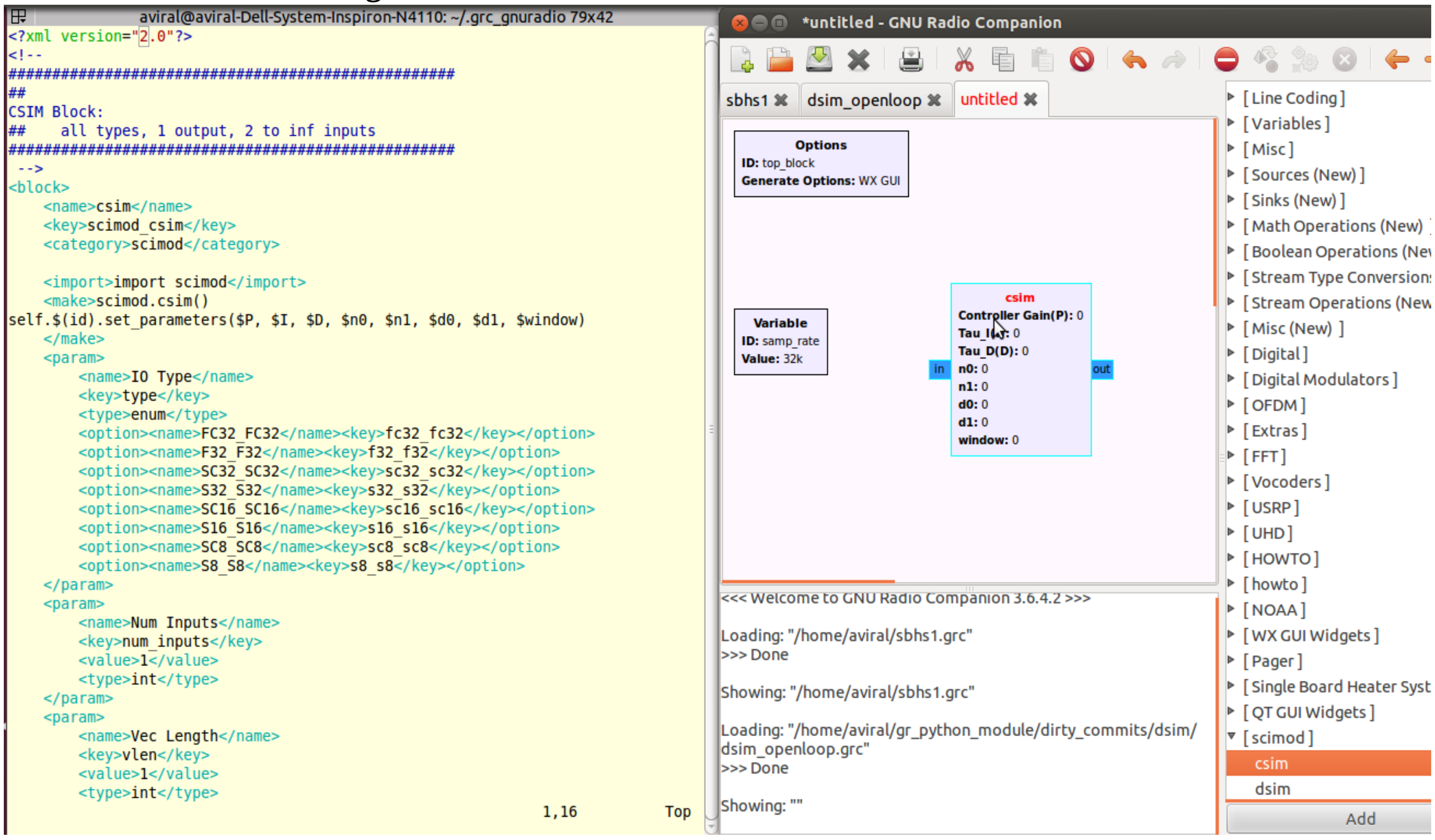


```
    def work(self, input_items, output_items):

        #n = min(len(input_items[0]), len(output_items[0]))
        in0 = input_items[0]
        out = output_items[0]

        from csim_sci import csim
        #Processing
        # Assuming n = 1 input_config(0)=1

        out[:self.n] = csim(self.param1, self.param2, self.param3, self.param4,
                            self.param5, self.param6, self.param7, in0[:self.n].tolist()) # IMP: in0[:self.n].tolist() passes a python array

        print "OUT", out[:self.n]

:set nowrap                                                                                    1,1
```

Figure 9: protocol for mapping outputs to inputs

This snippet captures the ability of python to map vector inputs to vector outputs using numpy arrays. On a careful observation it can be seen that in the mapping step just above the print statement, the input array has been converted to list. It was done achieved after many hit and trials as it was unknown that scilab was incapable of handling numpy array above certain sizes. However scilab is perfectly capable of handling lists of any size.

3.3 Generating the view



```
<?xml version="2.0"?>
<!--
###################################################
##
CSIM Block:
##    all types, 1 output, 2 to inf inputs
###################################################
 -->
<block>
    <name>csim</name>
    <key>scimod_csim</key>
    <category>scimod</category>

    <import>import scimod</import>
    <make>scimod.csim()
self.$(id).set_parameters($P, $I, $D, $n0, $n1, $d0, $d1, $window)
    </make>
    <param>
        <name>IO Type</name>
        <key>type</key>
        <type>enum</type>
        <option><name>FC32_FC32</name><key>fc32_fc32</key></option>
        <option><name>F32_F32</name><key>f32_f32</key></option>
        <option><name>SC32_SC32</name><key>sc32_sc32</key></option>
        <option><name>S32_S32</name><key>s32_s32</key></option>
        <option><name>SC16_SC16</name><key>sc16_sc16</key></option>
        <option><name>S16_S16</name><key>s16_s16</key></option>
        <option><name>SC8_SC8</name><key>sc8_sc8</key></option>
        <option><name>S8_S8</name><key>s8_s8</key></option>
    </param>
    <param>
        <name>Num Inputs</name>
        <key>num_inputs</key>
        <value>1</value>
        <type>int</type>
    </param>
    <param>
        <name>Vec Length</name>
        <key>vlen</key>
        <value>1</value>
        <type>int</type>
                                                                 1,16          Top
```

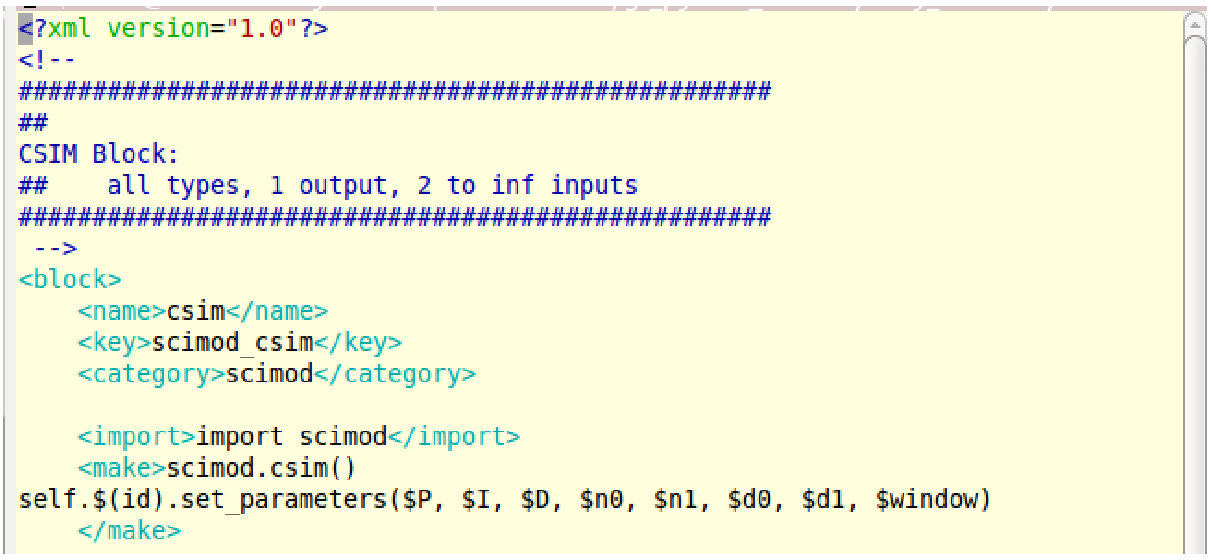
```
<<< Welcome to GNU Radio Companion 3.6.4.2 >>>

Loading: "/home/aviral/sbhs1.grc"
>>> Done

Showing: "/home/aviral/sbhs1.grc"

Loading: "/home/aviral/gr_python_module/dirty_commits/dsim/
dsim_openloop.grc"
>>> Done

Showing: ""
```

Figure 10:Generating the xml blocks and the view(a side by side rendering)

The out of tree module that gets created by the gr-modtool also helps us generate a skeleton for the xml block. The xml block defines the view. Upon completion it is included inside a hidden file in the home folder.

Say the xml block was called ***csim.xml***

It has to be included in ***/home/aviral/.grc_gnuradio/*** so that it appears in the GRC when the GRC is started using the command gnuradio-companion from the command line.



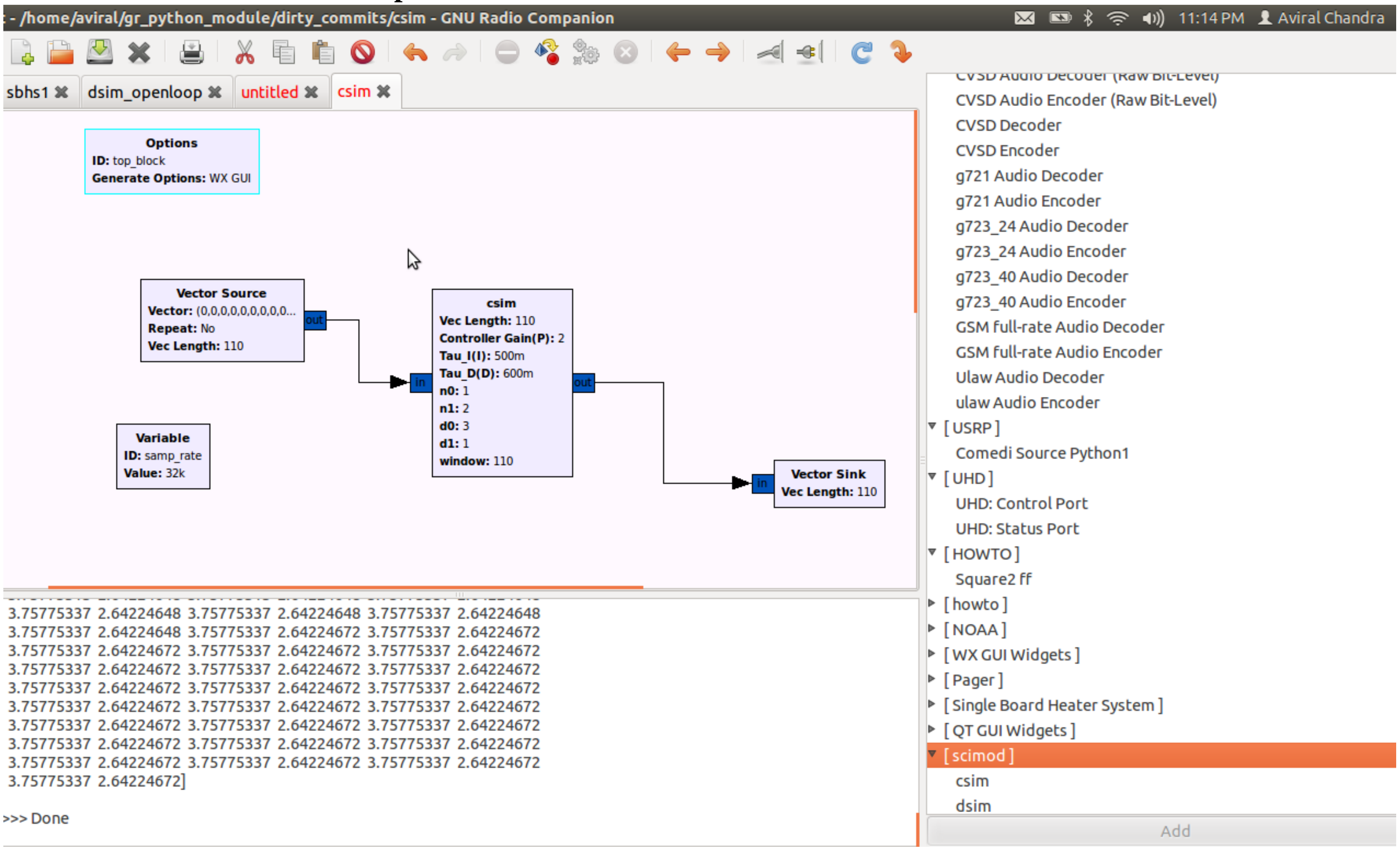
```
<?xml version="1.0"?>
<!--
###################################################
##
CSIM Block:
##    all types, 1 output, 2 to inf inputs
###################################################
 -->
<block>
    <name>csim</name>
    <key>scimod_csim</key>
    <category>scimod</category>

    <import>import scimod</import>
    <make>scimod.csim()
self.$(id).set_parameters($P, $I, $D, $n0, $n1, $d0, $d1, $window)
    </make>
```

Figure 11: python cheetah script passing the parameters P, I, D, n0, n1, d0, d1 and window length

3.4 Open loop testing of the csim and dsimul blocks

The dsimul blocks were developed using the same protocol, the APPENDIX 7 can be looked up to see its development code.

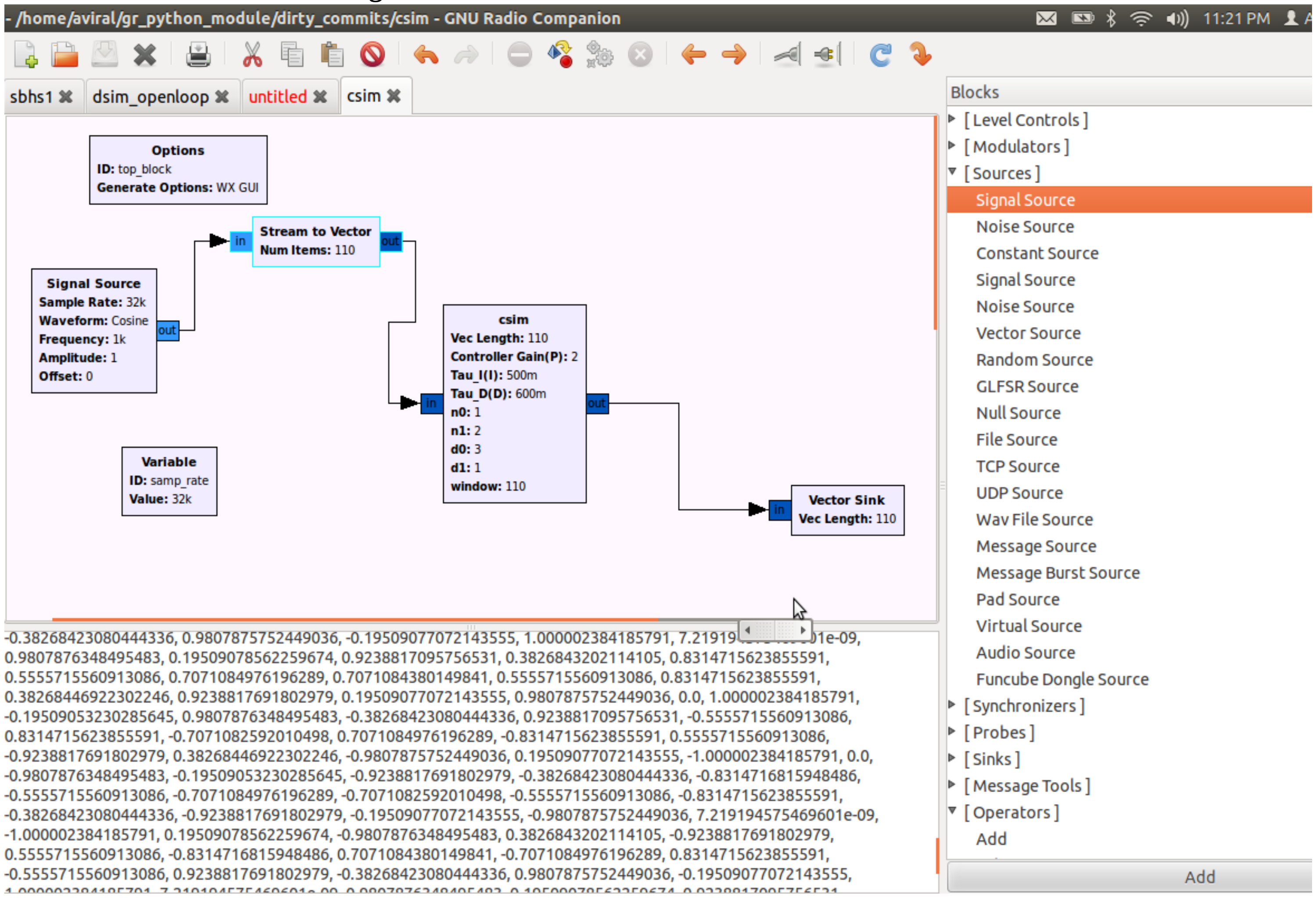
3.4.1 csim block : input vector source



```
3.75775337 2.64224648 3.75775337 2.64224648 3.75775337 2.64224648
3.75775337 2.64224648 3.75775337 2.64224672 3.75775337 2.64224672
3.75775337 2.64224672 3.75775337 2.64224672 3.75775337 2.64224672
3.75775337 2.64224672 3.75775337 2.64224672 3.75775337 2.64224672
3.75775337 2.64224672 3.75775337 2.64224672 3.75775337 2.64224672
3.75775337 2.64224672 3.75775337 2.64224672 3.75775337 2.64224672
3.75775337 2.64224672 3.75775337 2.64224672 3.75775337 2.64224672
3.75775337 2.64224672 3.75775337 2.64224672 3.75775337 2.64224672
3.75775337 2.64224672 3.75775337 2.64224672 3.75775337 2.64224672
3.75775337 2.64224672]

>>> Done
```

Figure 12: csim taking input from a vector source, o/p vector in lower left

3.4.2 csim block: signal source



Options
ID: top_block
Generate Options: WX GUI

Stream to Vector
Num Items: 110

Signal Source
Sample Rate: 32k
Waveform: Cosine
Frequency: 1k
Amplitude: 1
Offset: 0

csim
Vec Length: 110
Controller Gain(P): 2
Tau_I(I): 500m
Tau_D(D): 600m
n0: 1
n1: 2
d0: 3
d1: 1
window: 110

Variable
ID: samp_rate
Value: 32k

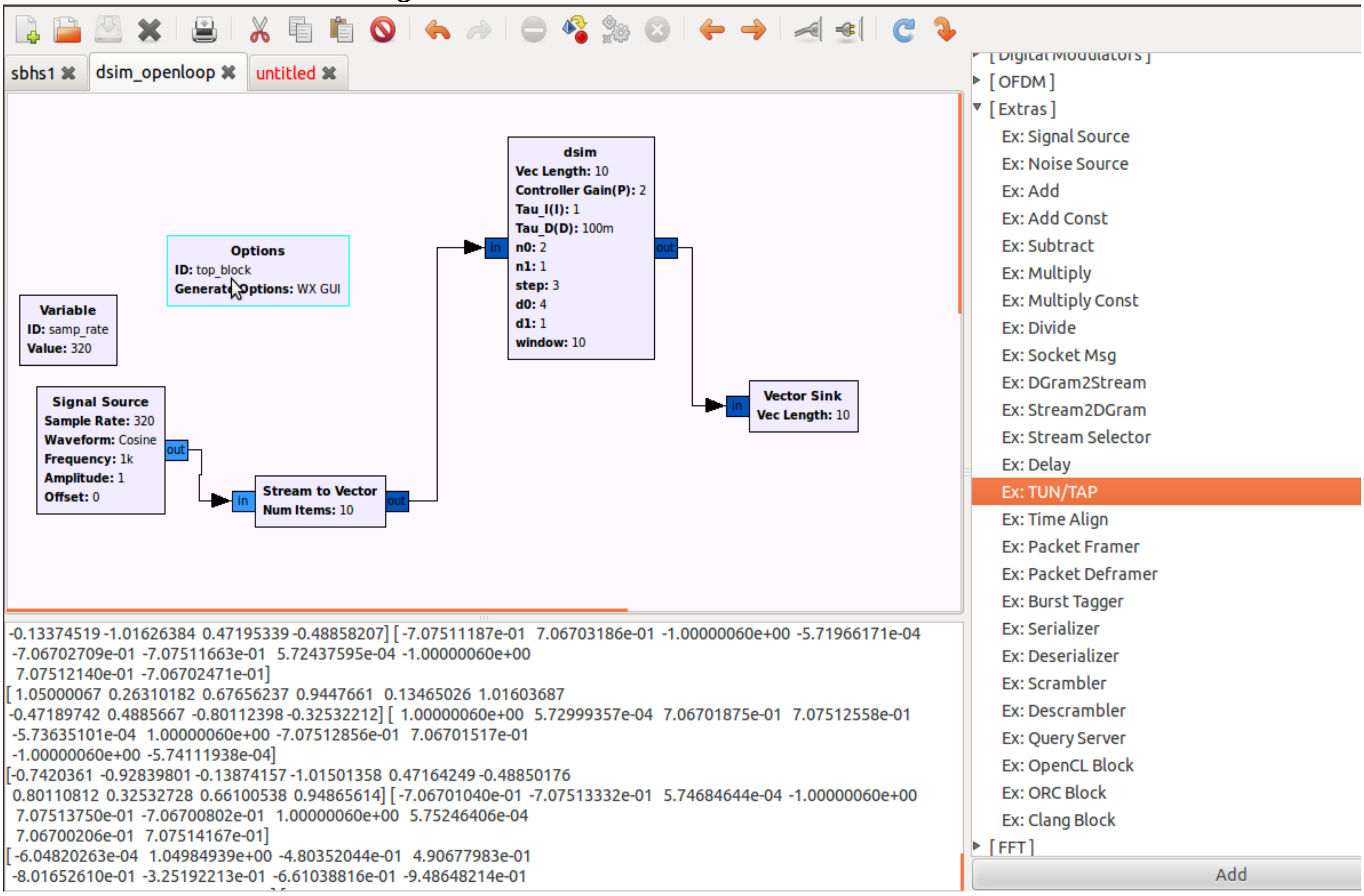
Vector Sink
Vec Length: 110

```
-0.38268423080444336, 0.9807875752449036, -0.19509077072143555, 1.000002384185791, 7.21919[illegible]01e-09,
0.9807876348495483, 0.19509078562259674, 0.9238817095756531, 0.3826843202114105, 0.8314715623855591,
0.5555715560913086, 0.7071084976196289, 0.7071084380149841, 0.5555715560913086, 0.8314715623855591,
0.38268446922302246, 0.9238817691802979, 0.19509077072143555, 0.9807875752449036, 0.0, 1.000002384185791,
-0.19509053230285645, 0.9807876348495483, -0.38268423080444336, 0.9238817095756531, -0.5555715560913086,
0.8314715623855591, -0.7071082592010498, 0.7071084976196289, -0.8314715623855591, 0.5555715560913086,
-0.9238817691802979, 0.38268446922302246, -0.9807875752449036, 0.19509077072143555, -1.000002384185791, 0.0,
-0.9807876348495483, -0.19509053230285645, -0.9238817691802979, -0.38268423080444336, -0.8314716815948486,
-0.5555715560913086, -0.7071084976196289, -0.7071082592010498, -0.5555715560913086, -0.8314715623855591,
-0.38268423080444336, -0.9238817691802979, -0.19509077072143555, -0.9807875752449036, 7.219194575469601e-09,
-1.000002384185791, 0.19509078562259674, -0.9807876348495483, 0.3826843202114105, -0.9238817691802979,
0.5555715560913086, -0.8314716815948486, 0.7071084380149841, -0.7071084976196289, 0.8314715623855591,
-0.5555715560913086, 0.9238817691802979, -0.38268423080444336, 0.9807875752449036, -0.19509077072143555,
```

Figure 13: csim taking input from a signal source, o/p vector in lower left

Note: This is the first time we've actually seen the advantage of having a visual programming language rich in source blocks. This perfectly justifies why an Electrical Engineering project was forked and moulded for a control system application.

3.4.3 dsim block: signal source



```
Options
ID: top_block
Generate Options: WX GUI

Variable
ID: samp_rate
Value: 320

Signal Source
Sample Rate: 320
Waveform: Cosine
Frequency: 1k
Amplitude: 1
Offset: 0

Stream to Vector
Num Items: 10

dsim
Vec Length: 10
Controller Gain(P): 2
Tau_I(I): 1
Tau_D(D): 100m
n0: 2
n1: 1
step: 3
d0: 4
d1: 1
window: 10

Vector Sink
Vec Length: 10
```

```
-0.13374519 -1.01626384 0.47195339 -0.48858207] [-7.07511187e-01  7.06703186e-01 -1.00000060e+00 -5.71966171e-04
 -7.06702709e-01 -7.07511663e-01  5.72437595e-04 -1.00000060e+00
  7.07512140e-01 -7.06702471e-01]
[ 1.05000067 0.26310182 0.67656237 0.9447661  0.13465026 1.01603687
-0.47189742 0.4885667 -0.80112398 -0.32532212] [ 1.00000060e+00  5.72999357e-04  7.06701875e-01  7.07512558e-01
 -5.73635101e-04  1.00000060e+00 -7.07512856e-01  7.06701517e-01
 -1.00000060e+00 -5.74111938e-04]
[-0.7420361 -0.92839801 -0.13874157 -1.01501358 0.47164249 -0.48850176
 0.80110812 0.32532728 0.66100538 0.94865614] [-7.06701040e-01 -7.07513332e-01  5.74684644e-04 -1.00000060e+00
  7.07513750e-01 -7.06700802e-01  1.00000060e+00  5.75246406e-04
  7.06700206e-01  7.07514167e-01]
[ -6.04820263e-04  1.04984939e+00 -4.80352044e-01  4.90677983e-01
 -8.01652610e-01 -3.25192213e-01 -6.61038816e-01 -9.48648214e-01
```

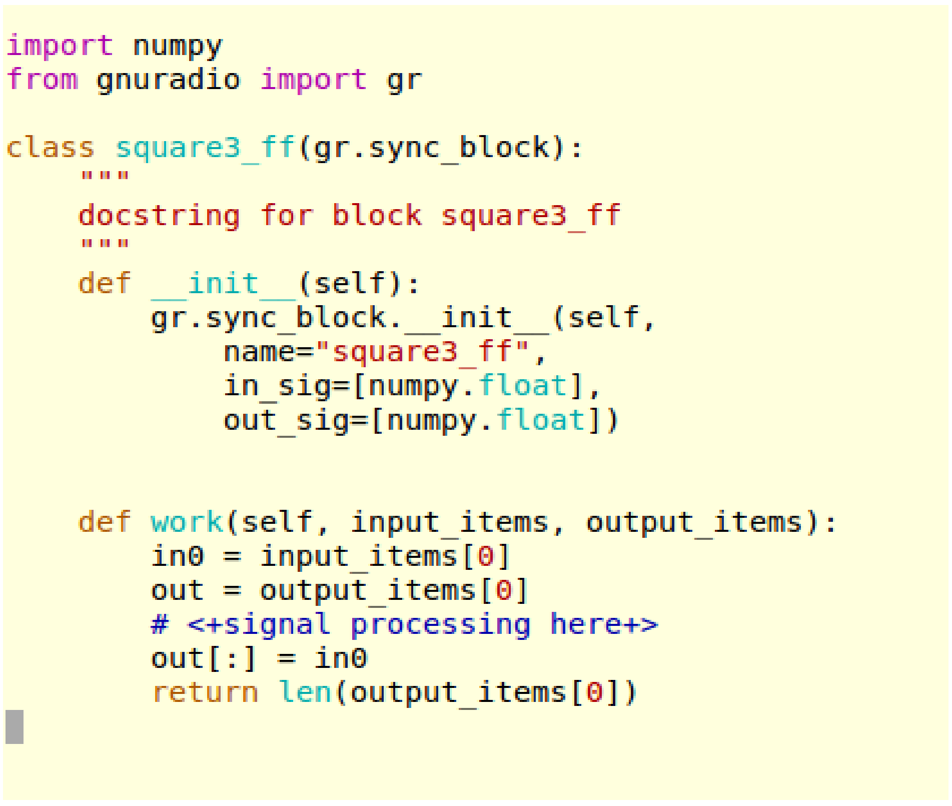
Figure 14: dsim taking input from a signal source, o/p vector in lower left

The dsiml function of scilab has been used to create the dsim block shown here. It gives an option to specify the plant-controller parameters and step size required to simulate it in an open loop mode.

3.5 GRAS and the feedback

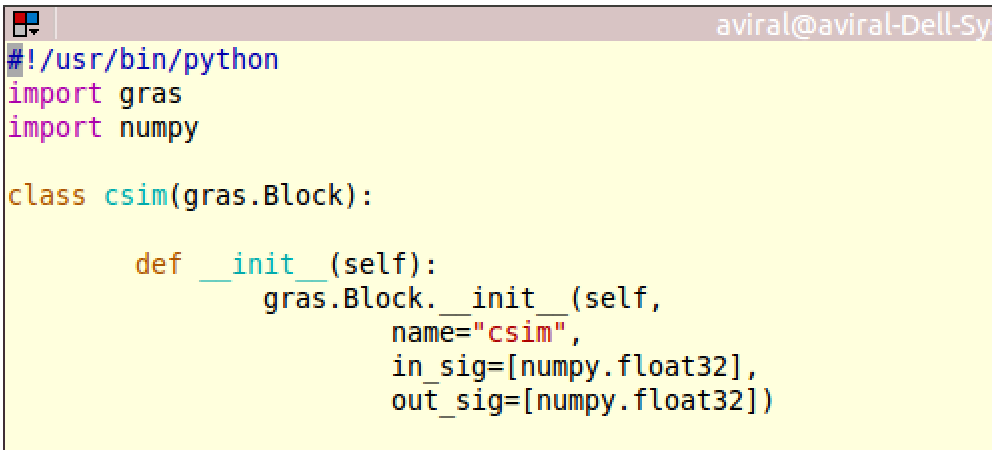
Stock Scheduler and **Advanced Scheduler** implementations have been illustrated below. It becomes clear from inspection of the screenshots that the blocks use an application scheduler which is imported in the second line in the first program and first line of the second program.

The stock scheduler fails as the hyper-threading library that's used by the program is not equipped to release threads pertaining to feedback situations. Feedback was made possible by collaborative development with a US-based developer Josh Blum, who re-wrote the threading library for our use-case.



```
import numpy
from gnuradio import gr

class square3_ff(gr.sync_block):
    """
    docstring for block square3_ff
    """
    def __init__(self):
        gr.sync_block.__init__(self,
            name="square3_ff",
            in_sig=[numpy.float],
            out_sig=[numpy.float])


    def work(self, input_items, output_items):
        in0 = input_items[0]
        out = output_items[0]
        # <+signal processing here+>
        out[:] = in0
        return len(output_items[0])
```

Figure 15: Implementation of GNU Radio stock scheduler, generated from gr-modtool. Import gr imports the stock process scheduler



```
aviral@aviral-Dell-Sy
#!/usr/bin/python
import gras
import numpy

class csim(gras.Block):

        def __init__(self):
                gras.Block.__init__(self,
                        name="csim",
                        in_sig=[numpy.float32],
                        out_sig=[numpy.float32])
```

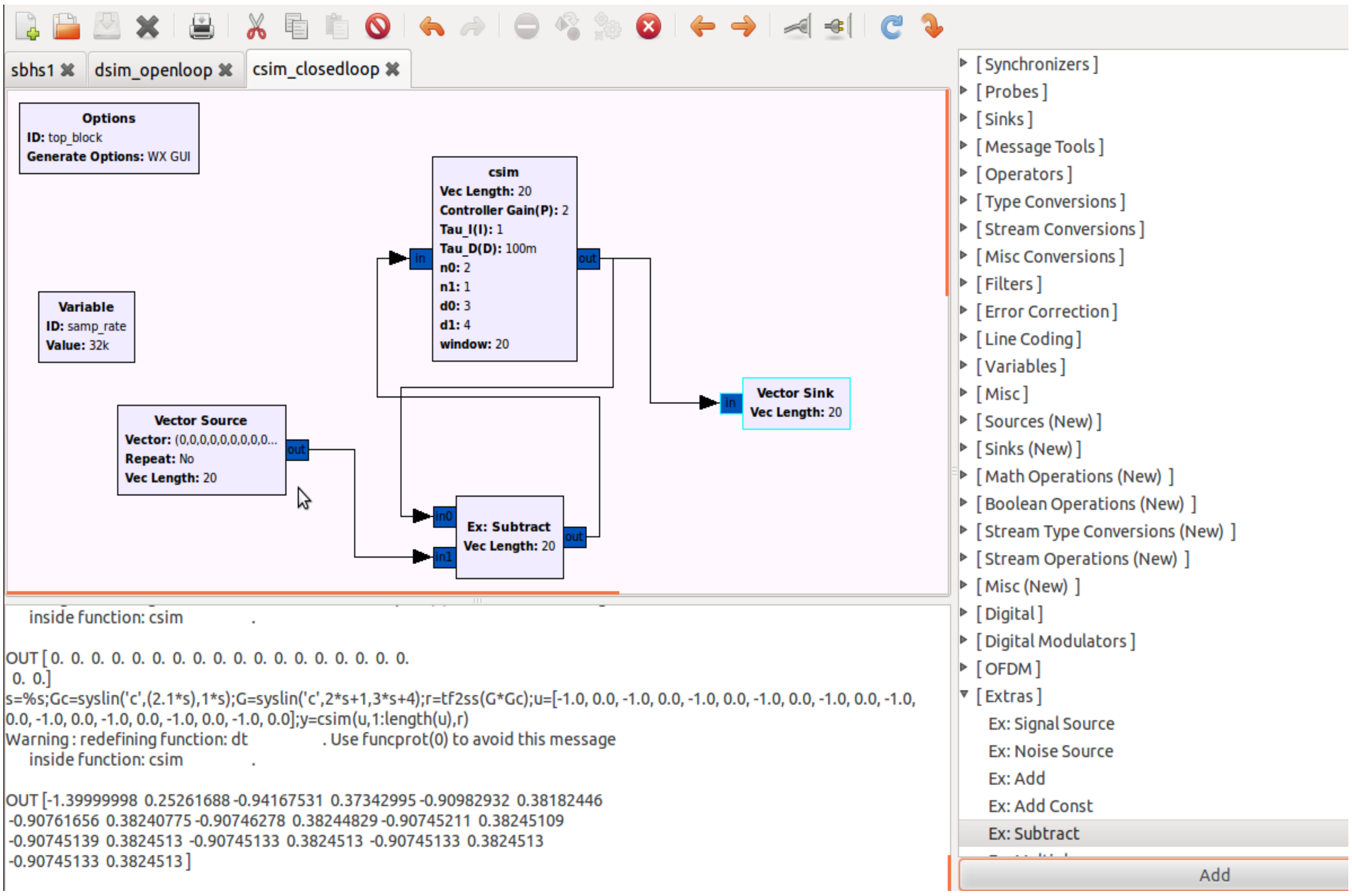
Figure 16: Implementation of gras, which is GNU-Radio Advanced Scheduler written for our program

Josh Blum, understood the roadblock which was stopping GNURadio from becoming a feedback-enabled control systems visual programming editor. Josh wrote a new application scheduler GRAS and handed it over for this application.

The compilation and build logic of this scheduler from the source code has been given in the APPENDIX 4

3.5.1 **The Feedback flow-graph**

The feedback flowgraph is generated by using the subtract block with preload set to 1. It works for the given scheduler and generates output as can be seen in the screenshot given below. It's worth mentioning that this flow-graph can never work with the stock scheduler since the stock scheduler's application scheduling can't understand feedback.



Options
ID: top_block
Generate Options: WX GUI

Variable
ID: samp_rate
Value: 32k

csim
Vec Length: 20
Controller Gain(P): 2
Tau_I(I): 1
Tau_D(D): 100m
n0: 2
n1: 1
d0: 3
d1: 4
window: 20

Vector Source
Vector: (0,0,0,0,0,0,0,0,0...
Repeat: No
Vec Length: 20

Vector Sink
Vec Length: 20

Ex: Subtract
Vec Length: 20

```
    inside function: csim             .

OUT [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
 0.  0.]
s=%s;Gc=syslin('c',(2.1*s),1*s);G=syslin('c',2*s+1,3*s+4);r=tf2ss(G*Gc);u=[-1.0, 0.0, -1.0, 0.0, -1.0, 0.0, -1.0, 0.0, -1.0, 0.0, -1.0,
0.0, -1.0, 0.0, -1.0, 0.0, -1.0, 0.0, -1.0, 0.0];y=csim(u,1:length(u),r)
Warning : redefining function: dt             . Use funcprot(0) to avoid this message
    inside function: csim             .

OUT [-1.39999998  0.25261688 -0.94167531  0.37342995 -0.90982932  0.38182446
 -0.90761656  0.38240775 -0.90746278  0.38244829 -0.90745211  0.38245109
 -0.90745139  0.3824513  -0.90745133  0.3824513  -0.90745133  0.3824513
 -0.90745133  0.3824513 ]
```

Figure 17:The feedback flow graph implemented for the first time in GNU Radio

4 Sandhi

Connecting blocks

4.1 Why Sandhi?

GNU Radio has a plethora of blocks which are of no use to us. They are appropriate only for electrical engineering applications. It made sense to remove these blocks and clean up the User Interface.

- Sandhi implements only the control relevant blocks developed so far.
- Sandhi has access to scilab's computation engine and scilab's control libraries via sciscipy
- Sandhi is feedback ready as the advanced scheduler has been made available for Sandhi.
- Sandhi has the conventional driver support available to GNU Radio and still carries the various sources, sinks and math operation blocks, that can be used efficiently to carry out simulations.

The commands that are followed for installing the software using CMake are as under:

a) mkdir build

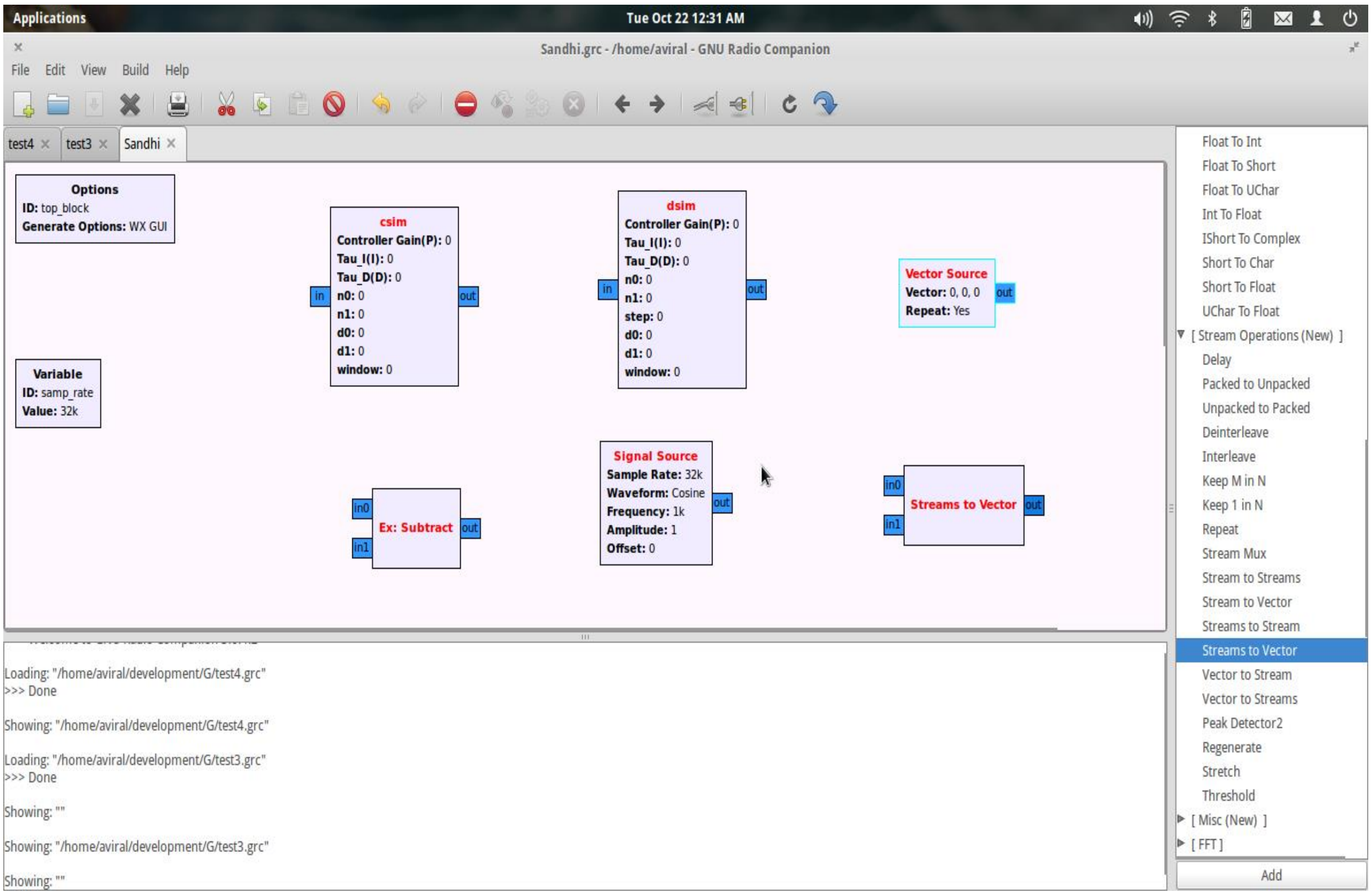
b) cd build/

c) cmake ../

d) make

e) make install

Clean UI, only control relevant blocks were retained by editing the CMake rules. CMake is a configuration file which is used to organize the software and its dependencies in the root file system It sets all the necessary file paths and makes the software compile ready. It also takes care of the hardware architecture on which the software is to be built. The CMake rules were edited with the help of Manoj and it was made appropriate to run on a 64-bit device as well as arm device.



Options
ID: top_block
Generate Options: WX GUI

Variable
ID: samp_rate
Value: 32k

csim
Controller Gain(P): 0
Tau_I(I): 0
Tau_D(D): 0
n0: 0
n1: 0
d0: 0
d1: 0
window: 0

dsim
Controller Gain(P): 0
Tau_I(I): 0
Tau_D(D): 0
n0: 0
n1: 0
step: 0
d0: 0
d1: 0
window: 0

Vector Source
Vector: 0, 0, 0
Repeat: Yes

Ex: Subtract

Signal Source
Sample Rate: 32k
Waveform: Cosine
Frequency: 1k
Amplitude: 1
Offset: 0

Streams to Vector

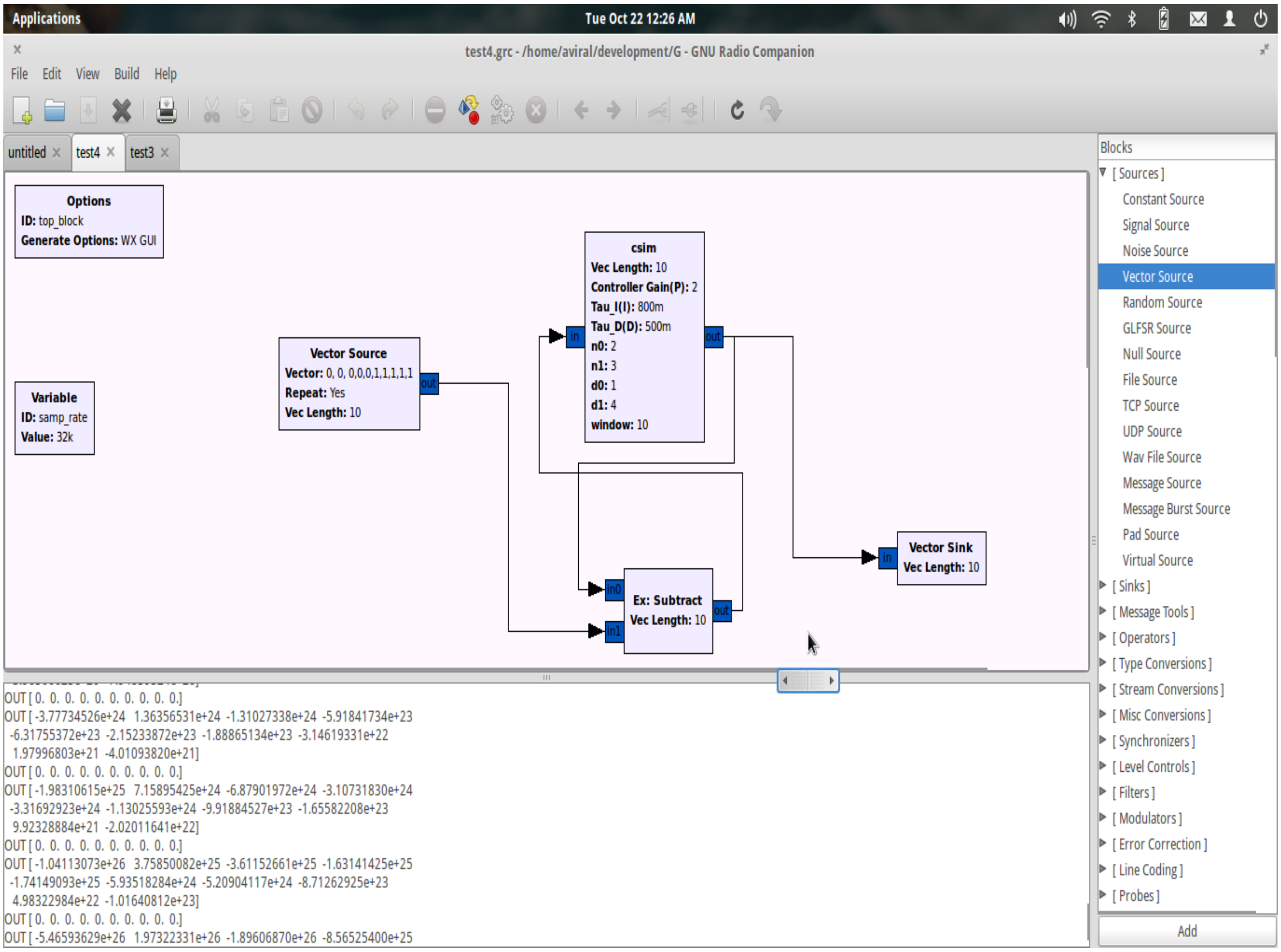
```
Loading: "/home/aviral/development/G/test4.grc"
>>> Done

Showing: "/home/aviral/development/G/test4.grc"

Loading: "/home/aviral/development/G/test3.grc"
>>> Done

Showing: ""

Showing: "/home/aviral/development/G/test3.grc"

Showing: ""
```

Figure 18: Sandhi and a few of the important blocks

4.2 Implementation of csim in Sandhi in a feedback mode

csim was implemented in Sandhi in the feedback mode and it was tested with two different signal source blocks in the feedback mode. The feedback is implemented by adding the subtract block as we work under the assumption that an error is fed to the plant-controller block whose dynamics have been coupled in csim.

The subtract block was initialized with a preload equal to 1 and the vector size was kept same as that of the signal source block. It would become clear in the next few screenshots.



Options
ID: top_block
Generate Options: WX GUI

Variable
ID: samp_rate
Value: 32k

Vector Source
Vector: 0, 0, 0,0,0,1,1,1,1,1
Repeat: Yes
Vec Length: 10

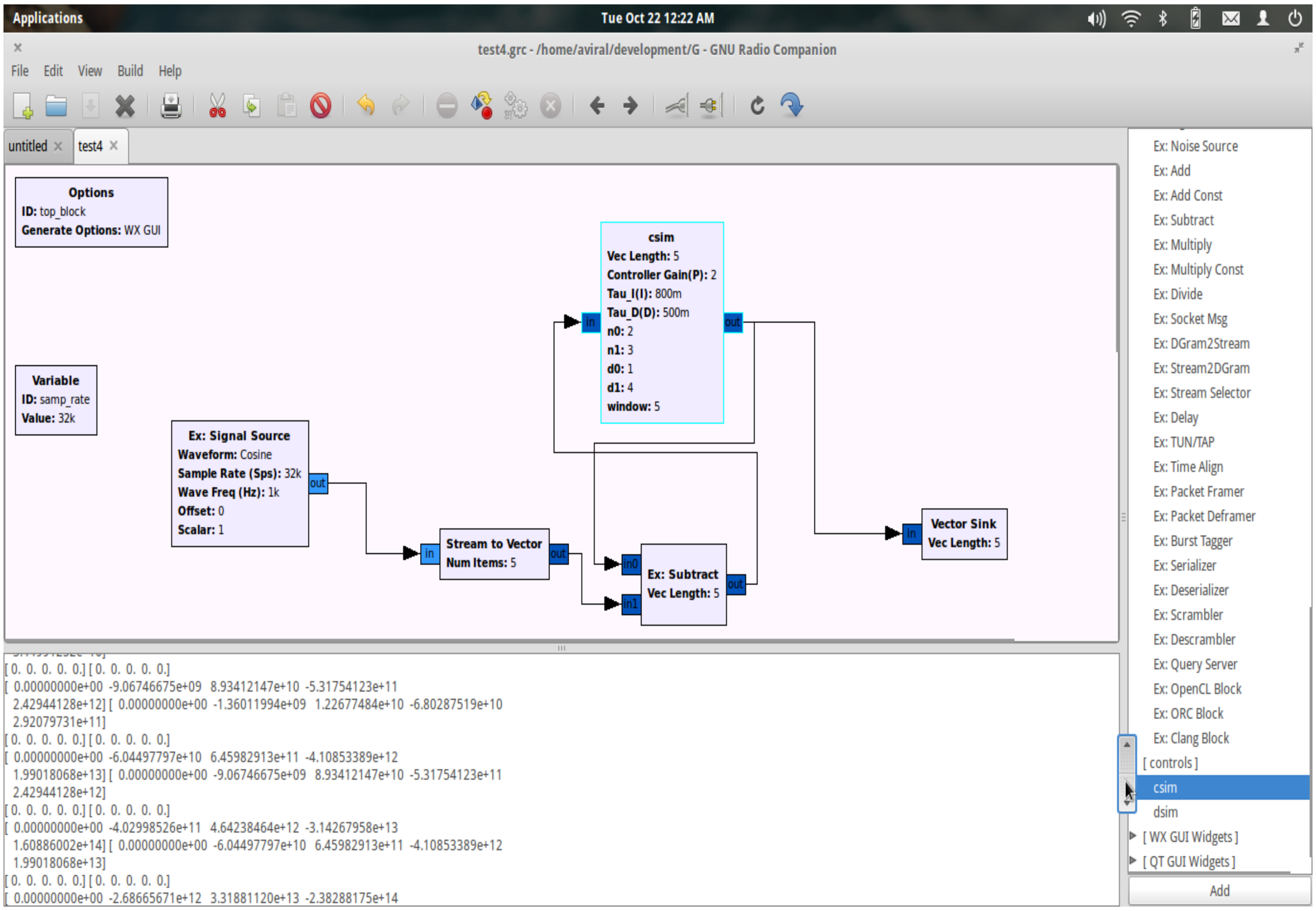
csim
Vec Length: 10
Controller Gain(P): 2
Tau_I(I): 800m
Tau_D(D): 500m
n0: 2
n1: 3
d0: 1
d1: 4
window: 10

Ex: Subtract
Vec Length: 10

Vector Sink
Vec Length: 10

```
OUT [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
OUT [ -3.77734526e+24  1.36356531e+24 -1.31027338e+24 -5.91841734e+23
 -6.31755372e+23 -2.15233872e+23 -1.88865134e+23 -3.14619331e+22
  1.97996803e+21 -4.01093820e+21]
OUT [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
OUT [ -1.98310615e+25  7.15895425e+24 -6.87901972e+24 -3.10731830e+24
 -3.31692923e+24 -1.13025593e+24 -9.91884527e+23 -1.65582208e+23
  9.92328884e+21 -2.02011641e+22]
OUT [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
OUT [ -1.04113073e+26  3.75850082e+25 -3.61152661e+25 -1.63141425e+25
 -1.74149093e+25 -5.93518284e+24 -5.20904117e+24 -8.71262925e+23
  4.98322984e+22 -1.01640812e+23]
OUT [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
OUT [ -5.46593629e+26  1.97322331e+26 -1.89606870e+26 -8.56525400e+25
```

Figure 19: closed loop implementation of csim on Sandhi with a vector source



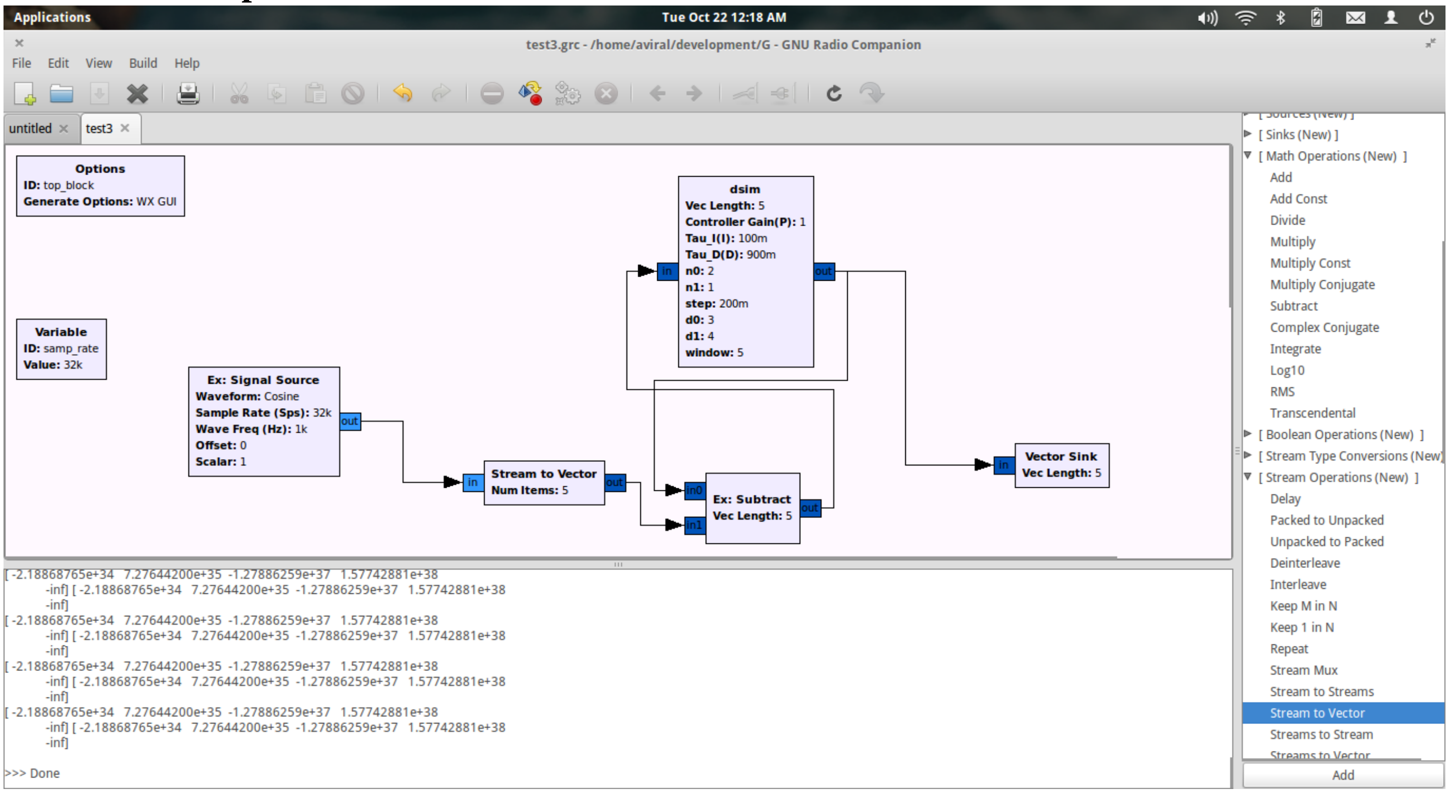
```
[ 0.  0.  0.  0.  0.] [ 0.  0.  0.  0.  0.]
[  0.00000000e+00  -9.06746675e+09   8.93412147e+10  -5.31754123e+11
   2.42944128e+12] [  0.00000000e+00  -1.36011994e+09   1.22677484e+10  -6.80287519e+10
   2.92079731e+11]
[ 0.  0.  0.  0.  0.] [ 0.  0.  0.  0.  0.]
[  0.00000000e+00  -6.04497797e+10   6.45982913e+11  -4.10853389e+12
   1.99018068e+13] [  0.00000000e+00  -9.06746675e+09   8.93412147e+10  -5.31754123e+11
   2.42944128e+12]
[ 0.  0.  0.  0.  0.] [ 0.  0.  0.  0.  0.]
[  0.00000000e+00  -4.02998526e+11   4.64238464e+12  -3.14267958e+13
   1.60886002e+14] [  0.00000000e+00  -6.04497797e+10   6.45982913e+11  -4.10853389e+12
   1.99018068e+13]
[ 0.  0.  0.  0.  0.] [ 0.  0.  0.  0.  0.]
[  0.00000000e+00  -2.68665671e+12   3.31881120e+13  -2.38288175e+14
```

Figure 20: closed loop implementation of csim on Sandhi with a signal source

We used an additional stream to vector block with number of items equal to the vector length the blocks are capable of handling in this situation. It converts the constant stream of data into chunks of vectors of length = vector length. It has to be specified while connecting this block.

It is important to set the preload condition of the subtract block equal to one in order to execute this flow-graph.

4.3 Implementation of dsimul in Sandhi



Options
ID: top_block
Generate Options: WX GUI

Variable
ID: samp_rate
Value: 32k

Ex: Signal Source
Waveform: Cosine
Sample Rate (Sps): 32k
Wave Freq (Hz): 1k
Offset: 0
Scalar: 1

Stream to Vector
Num Items: 5

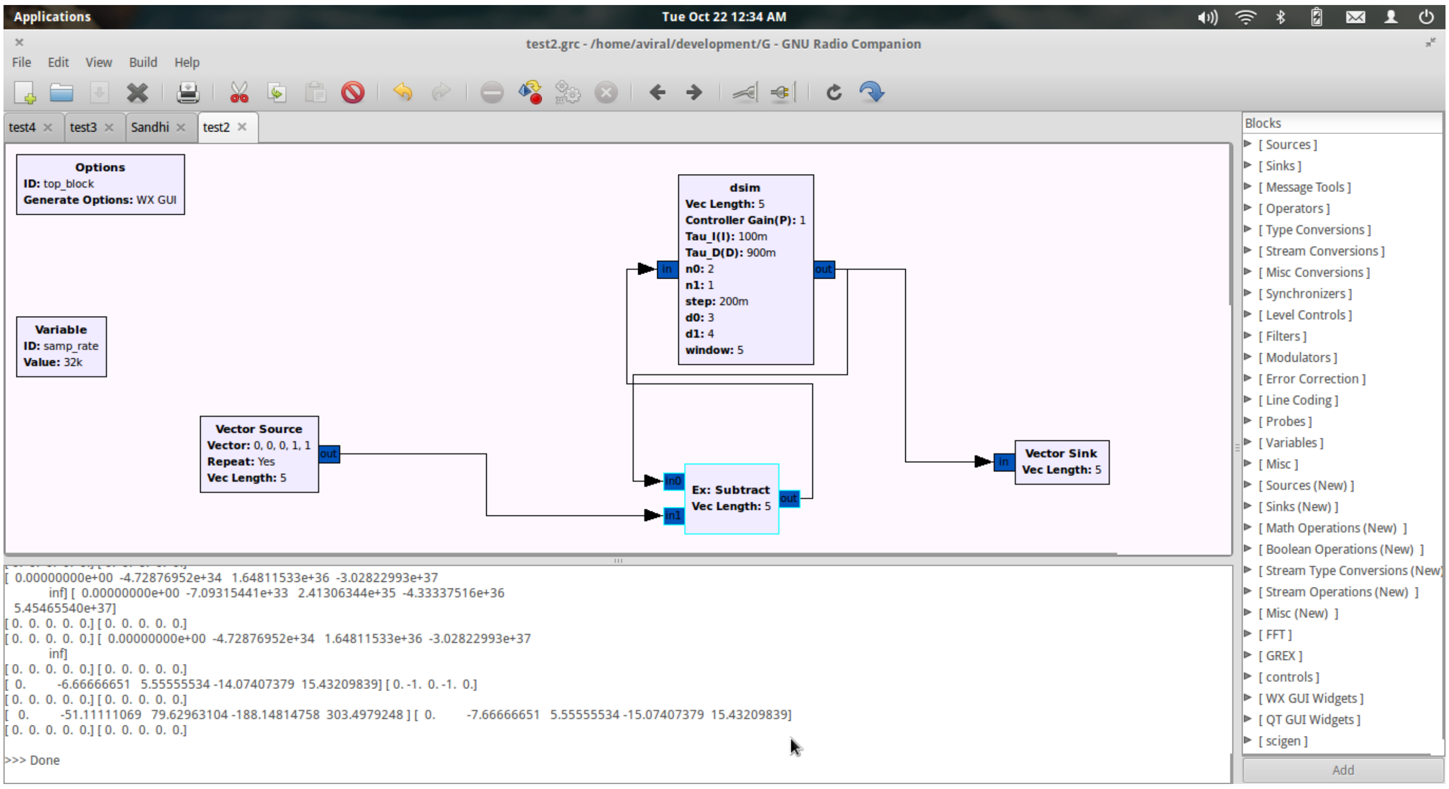
dsim
Vec Length: 5
Controller Gain(P): 1
Tau_I(I): 100m
Tau_D(D): 900m
n0: 2
n1: 1
step: 200m
d0: 3
d1: 4
window: 5

Ex: Subtract
Vec Length: 5

Vector Sink
Vec Length: 5

```
[ -2.18868765e+34  7.27644200e+35 -1.27886259e+37  1.57742881e+38
        -inf] [ -2.18868765e+34  7.27644200e+35 -1.27886259e+37  1.57742881e+38
        -inf]
[ -2.18868765e+34  7.27644200e+35 -1.27886259e+37  1.57742881e+38
        -inf] [ -2.18868765e+34  7.27644200e+35 -1.27886259e+37  1.57742881e+38
        -inf]
[ -2.18868765e+34  7.27644200e+35 -1.27886259e+37  1.57742881e+38
        -inf] [ -2.18868765e+34  7.27644200e+35 -1.27886259e+37  1.57742881e+38
        -inf]
[ -2.18868765e+34  7.27644200e+35 -1.27886259e+37  1.57742881e+38
        -inf] [ -2.18868765e+34  7.27644200e+35 -1.27886259e+37  1.57742881e+38
        -inf]

>>> Done
```

Figure 21:closed loop implementation of dsim on Sandhi with vector source



Options
ID: top_block
Generate Options: WX GUI

Variable
ID: samp_rate
Value: 32k

Vector Source
Vector: 0, 0, 0, 1, 1
Repeat: Yes
Vec Length: 5

dsim
Vec Length: 5
Controller Gain(P): 1
Tau_I(I): 100m
Tau_D(D): 900m
n0: 2
n1: 1
step: 200m
d0: 3
d1: 4
window: 5

Ex: Subtract
Vec Length: 5

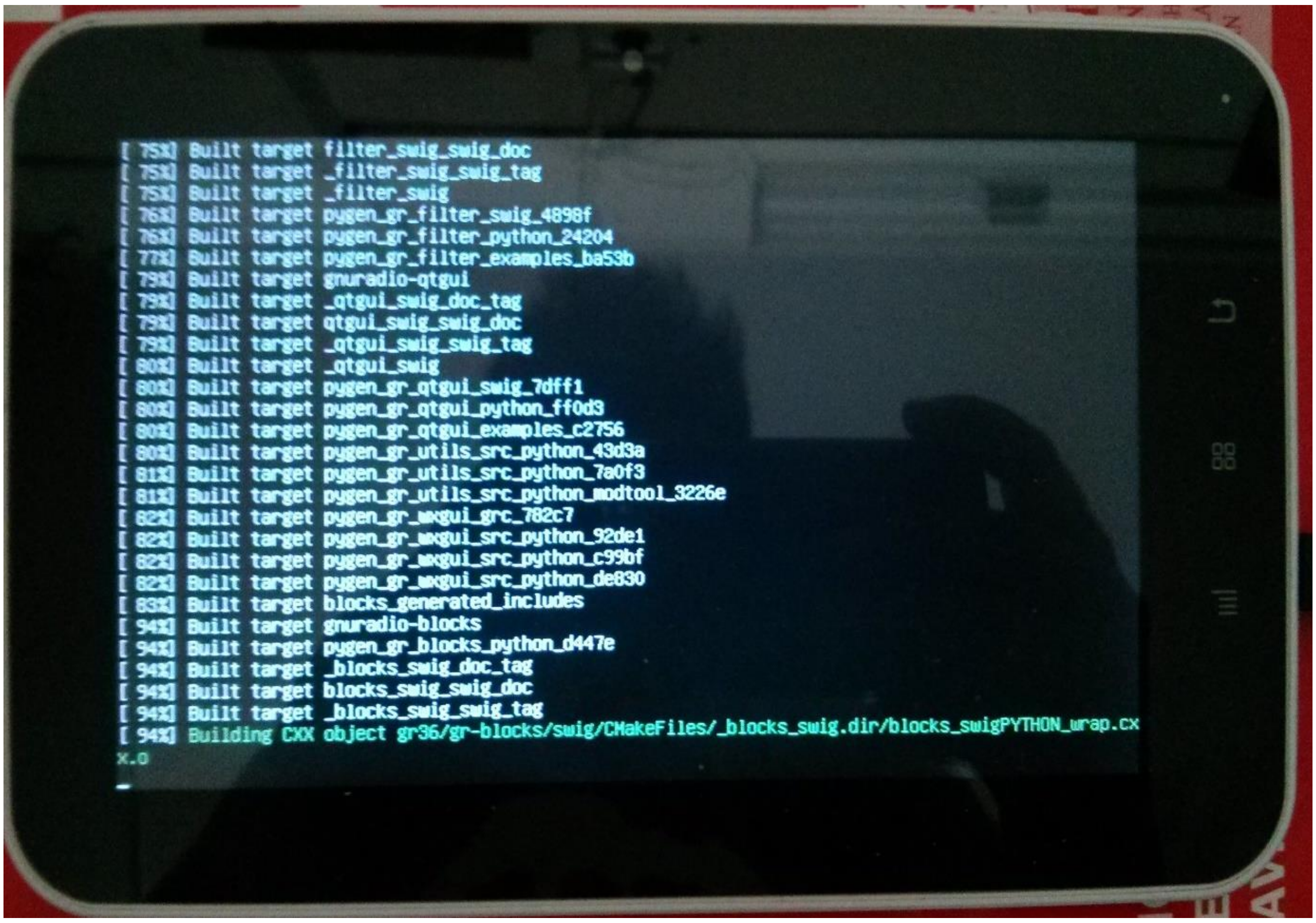
Vector Sink
Vec Length: 5

```
[ 0.00000000e+00 -4.72876952e+34  1.64811533e+36 -3.02822993e+37
        inf] [ 0.00000000e+00 -7.09315441e+33  2.41306344e+35 -4.33337516e+36
  5.45465540e+37]
[ 0.  0.  0.  0.  0.] [ 0.  0.  0.  0.  0.]
[ 0.  0.  0.  0.  0.] [ 0.00000000e+00 -4.72876952e+34  1.64811533e+36 -3.02822993e+37
        inf]
[ 0.  0.  0.  0.  0.] [ 0.  0.  0.  0.  0.]
[ 0.        -6.66666651   5.55555534 -14.07407379  15.43209839] [ 0. -1.  0. -1.  0.]
[ 0.  0.  0.  0.  0.] [ 0.  0.  0.  0.  0.]
[  0.        -51.11111069  79.62963104 -188.14814758  303.4979248 ] [ 0.        -7.66666651   5.55555534 -15.07407379  15.43209839]
[ 0.  0.  0.  0.  0.] [ 0.  0.  0.  0.  0.]

>>> Done
```

Figure 22: closed loop implementation of dsim on Sandhi with signal source

5 Sandhi on Aakash

The following screen shot captures the process of compilation and installation of Sandhi on Aakash. It would be available for test by the panel on the presentation day.

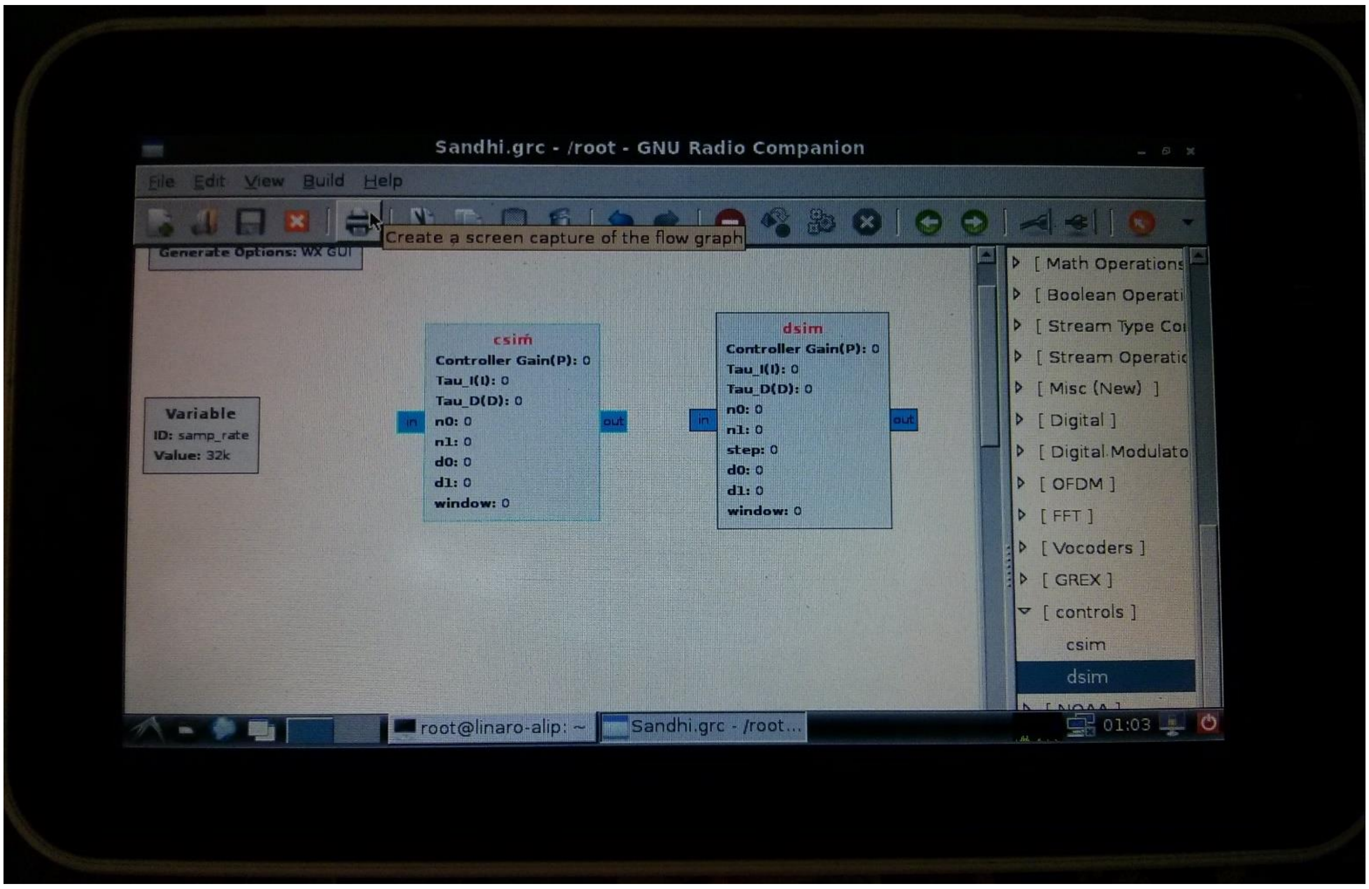


```
[ 75%] Built target filter_swig_swig_doc
[ 75%] Built target _filter_swig_swig_tag
[ 75%] Built target _filter_swig
[ 76%] Built target pygen_gr_filter_swig_4898f
[ 76%] Built target pygen_gr_filter_python_24204
[ 77%] Built target pygen_gr_filter_examples_ba53b
[ 79%] Built target gnuradio-qtgui
[ 79%] Built target _qtgui_swig_doc_tag
[ 79%] Built target qtgui_swig_swig_doc
[ 79%] Built target _qtgui_swig_swig_tag
[ 80%] Built target _qtgui_swig
[ 80%] Built target pygen_gr_qtgui_swig_7dff1
[ 80%] Built target pygen_gr_qtgui_python_ff0d3
[ 80%] Built target pygen_gr_qtgui_examples_c2756
[ 80%] Built target pygen_gr_utils_src_python_43d3a
[ 81%] Built target pygen_gr_utils_src_python_7a0f3
[ 81%] Built target pygen_gr_utils_src_python_modtool_3226e
[ 82%] Built target pygen_gr_wxgui_grc_782c7
[ 82%] Built target pygen_gr_wxgui_src_python_92de1
[ 82%] Built target pygen_gr_wxgui_src_python_c99bf
[ 82%] Built target pygen_gr_wxgui_src_python_de830
[ 83%] Built target blocks_generated_includes
[ 94%] Built target gnuradio-blocks
[ 94%] Built target pygen_gr_blocks_python_d447e
[ 94%] Built target _blocks_swig_doc_tag
[ 94%] Built target blocks_swig_swig_doc
[ 94%] Built target _blocks_swig_swig_tag
[ 94%] Building CXX object gr36/gr-blocks/swig/CMakeFiles/_blocks_swig.dir/blocks_swigPYTHON_wrap.cx
x.o
_
```

Figure 23: Compiling Sandhi on Aakash

Aakash tablet is an arm device, so Sandhi has to be compiled and installed in its environment. The normal compile and install time of Sandhi for arm devices stands at around 8 hours. The screenshot shown below shows Sandhi on Aakash.

Compiling Sandhi for Aakash completes the development cycle full circle. It enables the user to execute all the previously shown flow graphs from Aakash.



Generate Options: WX GUI

Variable
ID: samp_rate
Value: 32k

csim
Controller Gain(P): 0
Tau_I(I): 0
Tau_D(D): 0
n0: 0
n1: 0
d0: 0
d1: 0
window: 0

dsim
Controller Gain(P): 0
Tau_I(I): 0
Tau_D(D): 0
n0: 0
n1: 0
step: 0
d0: 0
d1: 0
window: 0

Figure 24: Sandhi on Aakash

6 Customizing Sandhi

6.1 How can one write a block on their own?

One can start writing a block in Sandhi using the gr-modtool. It serves as a great guideline to implement the blocks in GRC (GNU Radio Companion – the UI of GNU Radio). It is very important to look at the steps given in the chapters 2 and 3 to understand the development cycle. It captures a development cycle of around 20 weeks.

This documentation is also aimed at cutting short the development time of a new developer as the limitations of GNU Radio from control standpoint got exposed at a much later stage.

It is highly recommended that a new developer should stick to the protocol followed in APPENDIX 5 and Chapter 3.

The GUI can be modified and further improved by the developer if the developer takes keen interest in design aspects of User Interface

6.2 Methods of implementation

There are two ways of implementing blocks in Sandhi. The two techniques mentioned below have their own pros and cons.

6.2.1 C++ based implementation

If the user is striving for performance the blocks should be implemented in C++. C++ is a compiled language, thereby it converts the code directly into a native code of the particular machine. This makes C++ faster.

However with C++ the development time increases multifold. In the C++ implementation the swig wrapper generate python objects at runtime to be used by the python-cheetah script used to pass values to the functions from the GRC.

6.2.2 Python based implementation

Python is an interpreted language. At runtime byte code are generated which are converted to native machine code by some other language.

However python has a very rapid development cycle. With a little performance tradeoff, it gives the user a very rapid development cycle to prototype their ideas. However it is highly recommended that performance critical systems should be written in C++.

7 Importance of Sandhi to Chemical Engineers

The use of feedback is to improve the performance of scientific and industrial equipment. The fundamental premise of feedback loops is to take into account the actual measurements and hence compute the actuations in order to meet the operational specifications. This finds a lot of applications in areas like Process Control.

Control solutions require hardware interfacing, such as sending and receiving data from sensors, actuators. The transmission and reception of information is carried out using DAQ tools like LabVIEW.

The real-time control the output is made available to the system and the deviation is calculated from the desired o/p specifications. This error is systematically reduced and hence the output specifications are achieved.

Appropriate actuators and amplifiers allow the user to control the physical systems. A flow diagram of one such control system is given below

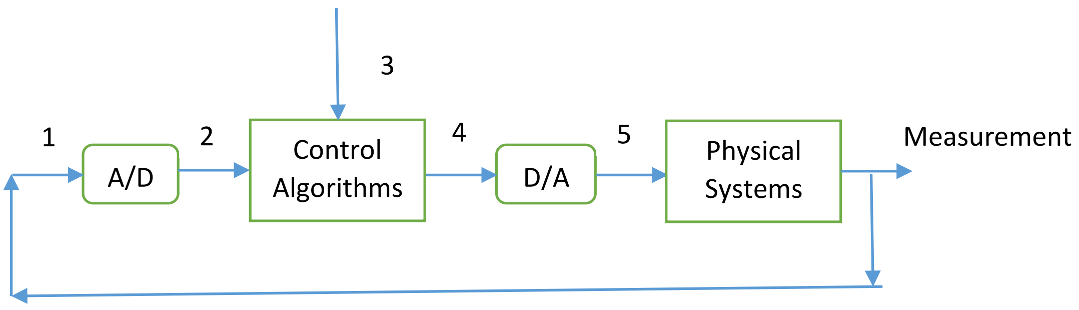


Figure 25: Adapted feeback structure from a paper[6] in references

(1)The measurement is converted into a digital signal a number here (2) it is then compared with a reference value (3) in the control algorithm block. The resulting command (4) is converted to an analog signal (5) and then applied to the physical system

Instrumentation and control are a very important part of setting up any chemical unit. The installation cost of these systems could be quite high.

Sandhi is being projected as an open source alternative to LabVIEW, and has the potential to substantially reduce the cost of setting up a remote lab for process control. A very holistic development approach is being taken and by the next phase a few LabVIEW applications would be functionally replaced by LabVIEW.

8 Comparison of LabVIEW and Sandhi

LabVIEW	Sandhi
LAbVIEW or Laboratory Virtual Instrument Engineering Workbench is a proprietary software for visual programming from National Instruments(NI)	Sandhi is an open source software built on GNU Radio; Sandhi can be used primarily for system simulation.
Provides built in hardware interfacing and DAQ tools, drivers for all NI DAQ cards(which are also proprietary in nature)	Inherits drivers from GNU Radio’s UHD(Universal Software Radio Peripheral Hardware Driver) module and COMEDI
Contains signal processing blocks, controller blocks, blocks for solving linear algebraic equation, advanced calculus etc. which are abstracted by simple intuitive blocks	Inherits advanced signal processing blocks from GNU Radio, contains basic control system simulation blocks as of yet, along with basic mathematics blocks.
Only LabVIEW Full and Professional Development systems can be interfaced with MATLAB using ActiveX technology	Sandhi can harness various computational engines; currently it can be interfaced with Scilab (using Sciscipy), GNU Octave, and Python libraries.
LabVIEW runs only on MS Windows as well as Linux for X86 and X64 architecture	Sandhi can be compiled on Windows and Linux for X86, X64 as well as ARM devices.
It offers graphical programming for many of microcontroller and FPGA kits(Field Programmable Gate Arrays).	Sandhi currently cannot program embedded devices.

Table 1: A comparison table between LabVIEW and Sandhi

9 Roadmap

9.1 Where do we stand? And Proposed Future Work

Sandhi has taken the shape of a visual programming editor. A rapid development cycle has been developed and it can easily be adapted to use many other control libraries available in Scilab. It’s the combined responsibility of the control community and our team to start functionally replacing LabVIEW applications by Sandhi.

Sandhi’s device support is explained by the fact that it supports COMEDI drivers. Moreover it’s based on a python framework, which makes all the free and open drivers available in python available to Sandhi.

The α version is ready and is to be released by November 1st 2013.

The future development revolves around functionally replacing a few LabVIEW implementation in Sandhi. The plan is also to try and functionally port LabVIEW based Virtual Lab applications by Sandhi as possible. It could be a great testing ground for our software.

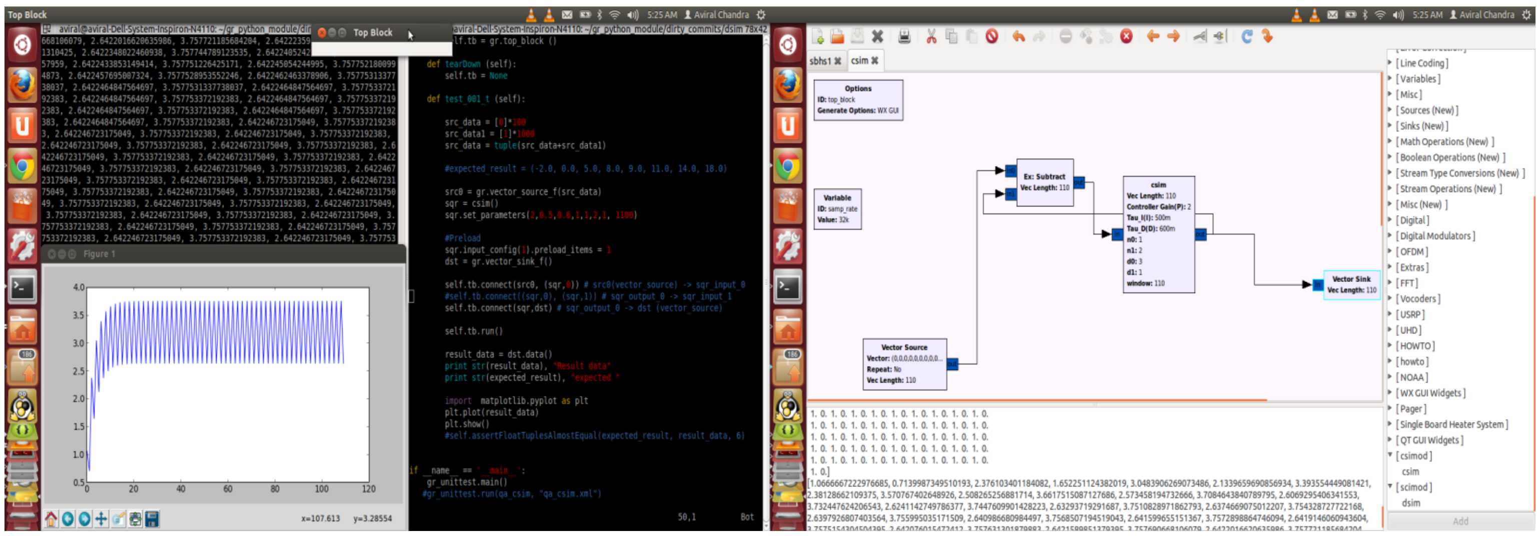
9.2 Plotting library

A plotting library is already under-way and would be fixed by the time we release the α version of our software, which is around November 1st 2013. The conventional plot blocks are not working with our control blocks, they are giving erroneous plots as they are taking some erroneous pre-load condition and re-initializing after some vector length. The debugging has already started and it would be fixed in a matter of week.

9.3 Developing the network communication protocols and remote lab for the users

The APPENDIX 1 shows implementation of a network protocol to exchange data to and from a server. It’s a lightweight URL based communication protocol implemented entirely in python.

Given the premise of remote labs, it’s important for the client and the server to exchange data. Such a data exchange has been shown and it’s implementation explained. The next phase would be about connecting these blocks and coming up with a novel remote lab.



```
aviral@aviral-Dell-System-Inspiron-N4110: ~/gr_python_module/dir
668106079, 2.6422016620635986, 3.757721185684204, 2.64222359
1310425, 2.6422348022460938, 3.757744789123535, 2.6422405242
57959, 2.6422433853149414, 3.757751226425171, 2.642245054244995, 3.757752180099
4873, 2.6422457695007324, 3.7577528953552246, 2.6422462463378906, 3.75775313377
38037, 2.6422464847564697, 3.7577531337738037, 2.6422464847564697, 3.7577533721
92383, 2.6422464847564697, 3.757753372192383, 2.6422464847564697, 3.75775337219
2383, 2.6422464847564697, 3.757753372192383, 2.6422464847564697, 3.757753372192
383, 2.6422464847564697, 3.757753372192383, 2.642246723175049, 3.75775337219238
3, 2.642246723175049, 3.757753372192383, 2.642246723175049, 3.757753372192383,
2.642246723175049, 3.757753372192383, 2.642246723175049, 3.757753372192383, 2.6
42246723175049, 3.757753372192383, 2.642246723175049, 3.757753372192383, 2.6422
46723175049, 3.757753372192383, 2.642246723175049, 3.757753372192383, 2.6422467
23175049, 3.757753372192383, 2.642246723175049, 3.757753372192383, 2.6422467231
75049, 3.757753372192383, 2.642246723175049, 3.757753372192383, 2.6422467231750
49, 3.757753372192383, 2.642246723175049, 3.757753372192383, 2.642246723175049,
 3.757753372192383, 2.642246723175049, 3.757753372192383, 2.642246723175049, 3.
757753372192383, 2.642246723175049, 3.757753372192383, 2.642246723175049, 3.757
753372192383, 2.642246723175049, 3.757753372192383, 2.642246723175049, 3.757753
```

```
aviral-Dell-System-Inspiron-N4110: ~/gr_python_module/dirty_commits/dsim 78x42
lf.tb = gr.top_block ()

    def tearDown (self):
        self.tb = None

    def test_001_t (self):

        src_data = [0]*100
        src_data1 = [1]*1000
        src_data = tuple(src_data+src_data1)

        #expected_result = (-2.0, 0.0, 5.0, 8.0, 9.0, 11.0, 14.0, 18.0)

        src0 = gr.vector_source_f(src_data)
        sqr = csim()
        sqr.set_parameters(2,0.5,0.6,1,1,2,1, 1100)

        #Preload
        sqr.input_config(1).preload_items = 1
        dst = gr.vector_sink_f()

        self.tb.connect(src0, (sqr,0)) # src0(vector_source) -> sqr_input_0
        #self.tb.connect((sqr,0), (sqr,1)) # sqr_output_0 -> sqr_input_1
        self.tb.connect(sqr,dst) # sqr_output_0 -> dst (vector_source)

        self.tb.run()

        result_data = dst.data()
        print str(result_data), "Result data"
        print str(expected_result), "expected "

        import  matplotlib.pyplot as plt
        plt.plot(result_data)
        plt.show()
        #self.assertFloatTuplesAlmostEqual(expected_result, result_data, 6)


if __name__ == '__main__':
    gr_unittest.main()
   #gr_unittest.run(qa_csim, "qa_csim.xml")
                                                              50,1          Bot
```

Options
ID: top_block
Generate Options: WX GUI

Variable
ID: samp_rate
Value: 32k

Vector Source
Vector: (0,0,0,0,0,0,0,0,0...
Repeat: No
Vec Length: 110

Ex: Subtract
Vec Length: 110

csim
Vec Length: 110
Controller Gain(P): 2
Tau_I(I): 500m
Tau_D(D): 600m
n0: 1
n1: 2
d0: 3
d1: 1
window: 110

Vector Sink
Vec Length: 110

```
1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0.
1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0.
1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0.
1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0.
1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0.
1. 0.]
[1.0666667222976685, 0.7139987349510193, 2.376103401184082, 1.652251124382019, 3.0483906269073486, 2.1339659690856934, 3.393554449081421,
2.38128662109375, 3.570767402648926, 2.508265256881714, 3.6617515087127686, 2.573458194732666, 3.7084643840789795, 2.6069295406341553,
3.732447624206543, 2.6241142749786377, 3.7447609901428223, 2.63293719291687, 3.7510828971862793, 2.6374669075012207, 3.754328727722168,
2.6397926807403564, 3.755995035171509, 2.640986680984497, 3.7568507194519043, 2.641599655151367, 3.7572898864746094, 2.6419146060943604,
```

Figure 26: A static plot generated by dumping data in python workspace. This would be fixed by the time Sandhi is released

10 References

- [1]Arquimedes Barrios , Stifen Panche , Mauricio Duque , Victor H. Grisales , Flavio Prieto, José L. Villa, Philippe Chevrel, Michael Canu: A multi-user remote academic laboratory system, Article, Elseveir: Computers and Education,

  Article history:
  Received 28 April 2012
  Accepted 17 October 2012

- [2]Dictino Chaos[1];*, Jes ´us Chac´on[1], Jose Antonio Lopez-Orozco[2] and Sebasti´an Dormido[1] : Virtual and Remote Robotic Laboratory Using EJS, MATLAB and Lab-VIEW

  1)Department of Computer Science and Automatic Control, UNED, Juan del Rosal 16, Madrid 28040, Spain; E-Mails: jchacon@bec.uned.es (J.C.); sdormido@dia.uned.es (S.D.)

  2)Department of Computers Architecture and Automatic Control, Complutense University, Ciudad Universitaria, Madrid 28040, Spain; E-Mail: jalo@dacya.ucm.es

  Published in Sensors — Open Access Journal *Sensors* (ISSN 1424-8220; CODEN: SENSC9)

  Article History:
  Received: 28 December 2012; in revised form: 1 February 2013
  Accepted: 16 February 2013
  Published: 21 February 2013

- [3]Carla Martin-Villalba ∗, Alfonso Urquia, Sebastian Dormido : Object-oriented modelling of virtual-labs for education in chemical process control, *Dept. Inform´atica y Autom´atica, UNED, Juan del Rosal 16, 28040 Madrid, Spain*

  Article history:
  Received 27 January 2006
  Received in revised form 9 September 2007
  Accepted: 19 may 2008
  Available online: 7 July 2008

- [4]KLEIN_ and G. WOZNY: WEB BASED REMOTE EXPERIMENTS FOR CHEMICAL ENGINEERING EDUCATION, Paper in IChem E, TU Berlin, Institute of Process and Plant Technology, Berlin, Germany

- [5]Jagdish Y. Patil, Balashish Dubey, Kannan M. Moudgalya, Rakesh Peter: GNURadio, Scilab, Xcos and COMEDI for Data Acquisition and Control: An Open Source Alternative to LabVIEW, Article, IIT Bombay, Mumbai 400076,

Article History:
Preprints of the 8th IFAC Symposium on Advanced Control of Chemical Processes
The International Federation of Automatic Control Furama Riverfront, Singapore, July 10-13, 2012

- [6]Introduction to Real-time Control using LabVIEW^{TM} with an Application to Distance Learning*
Authors:
Ch. SALZMANN, D. GILLET, and P. HUGUENIN, Swiss Federal Institute of Technology Lausanne, Switzerland. E-mail: christophe.salzmann:epfi.ch

- [7]COMEDI. http://www.comedi.org/
- [8]Internship Report on Graphical Programming Language LabVIEW & Xcos, GNU Radio or Blockly as an Open Source Alternative to LabVIEW for Data Acquisition and Control
Saruch Rathore

INTERNET RESOURCES:

- [1]Scilab Anywhere C/S – a Client/Server system to provide remote Scilab services
See: http://scilabanywhere.sourceforge.net/

- [2]Virtual Labs (India)
See: http://en.wikipedia.org/wiki/Virtual_Labs_(India)

- [3[Labshare
See: http://www.labshare.edu.au/

- [4]Remote Laboratory
See: http://en.wikipedia.org/wiki/Remote_laboratory

- [5]Aakash (tablet)
See: http://en.wikipedia.org/wiki/Aakash_(tablet)

- [6]Aakash 2
See: http://en.wikipedia.org/wiki/Aakash_2

- [7]GNU Radio
See: http://gnuradio.org/redmine/projects/gnuradio/wiki

- [8]Sciscipy: A Scilab API for Python
See: http://forge.scilab.org/index.php/p/sciscipy/

- [9]Abstraction (computer science)
See: http://en.wikipedia.org/wiki/Abstraction_(computer_science)

- [10]GNU Radio: Out-of-tree modules. Extending GNU radio with own functionality and blocks
  See: http://gnuradio.org/redmine/projects/gnuradio/wiki/OutOfTreeModules

- [11]Blockly: A visual programming editor
  See: https://code.google.com/p/blockly/

- [12]Google app-engine
  See: https://cloud.google.com/console#/project/apps~remote-cloudlabs

- [13]NI LabVIEW Full Development System for Linux,
  See: http://sine.ni.com/nips/cds/view/p/lang/en/nid/2541

- [14]Executing MATLAB Scripts in LabVIEW
  See: http://zone.ni.com/reference/en-XX/help/371361J-01/gmath/matlab_script_node/

- [15]LabVIEW support for Linux
  See: http://sine.ni.com/nips/cds/view/p/lang/en/nid/2541

VERSION CONTROL SYSTEM:

Repositories created, forked and contributed to:

- GIMP Tool Kit Widgets and experiment with out of tree modules
  https://github.com/manojgudi/gnu_lc/commits/master

- First python module successfully implemented
  https://github.com/manojgudi/gr_python_module

- Writing python blocks
  https://github.com/aviralchandra/gr-py_block

- Experiments with xcos UI widgets, knob implementation
  https://github.com/aviralchandra/xcos_UI

- Complied binaries of Sandhi made available
  https://github.com/manojgudi/sandhi

- GRExtras - Advanced GNU Radio Blocks forked from Josh Blum
  https://github.com/aviralchandra/grextras

- GRAS: forked from Josh Blum
  https://github.com/aviralchandra/gras

APPENDICES

APPENDIX 1

Implementing a URL based communicating protocol

Developing a communication protocol for communicating data from a remote location. This Appendix implements an URL based communication protocol.

The domain cloudlabs was registered at google appengine. It has been used for all the communications

One should type the following URL in their respective browser to see the results:

www.remote-cloudlabs.appspot.com/hello?temp=100&heat=100&fan=100

The results are available on the following URL:

www.remote-cloudlabs.appspot.com/display

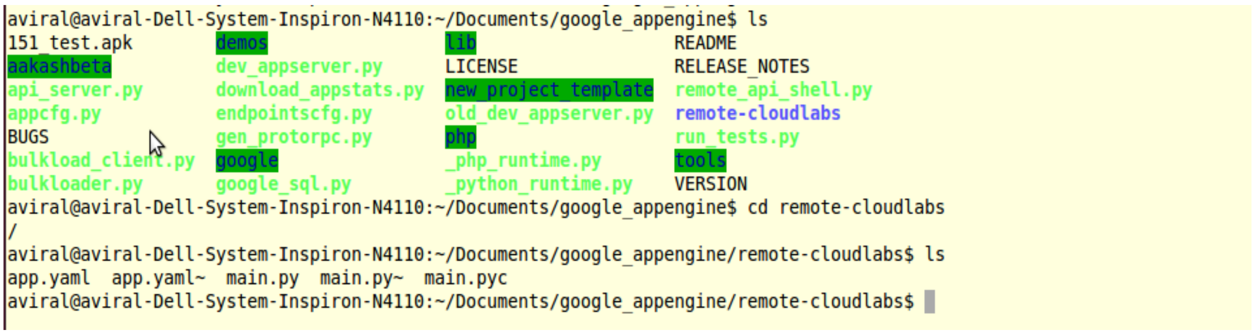
This URL can be parsed from any programming language. JSON and Python urllib are among popular libraries to parse this URL. The first URL is used to send values to the database registered on the google's free cloud-service app-engine.

The second URL is used to retrieve this data.

The implementation and the exact code is given here.

Code:

The google app-engine for Ubuntu 12.04 was downloaded and installed and the main.py file found under the remote-cloudlabs was written. The following snippet shows the exact file path to find the main.py file to be edited.



```
aviral@aviral-Dell-System-Inspiron-N4110:~/Documents/google_appengine$ ls
151_test.apk        demos               lib                 README
aakashbeta          dev_appserver.py    LICENSE             RELEASE_NOTES
api_server.py       download_appstats.py new_project_template remote_api_shell.py
appcfg.py           endpointscfg.py     old_dev_appserver.py remote-cloudlabs
BUGS                gen_protorpc.py     php                 run_tests.py
bulkload_client.py  google              _php_runtime.py     tools
bulkloader.py       google_sql.py       _python_runtime.py  VERSION
aviral@aviral-Dell-System-Inspiron-N4110:~/Documents/google_appengine$ cd remote-cloudlabs
/
aviral@aviral-Dell-System-Inspiron-N4110:~/Documents/google_appengine/remote-cloudlabs$ ls
app.yaml  app.yaml~  main.py  main.py~  main.pyc
aviral@aviral-Dell-System-Inspiron-N4110:~/Documents/google_appengine/remote-cloudlabs$
```

APPENDIX Figure 1:finding the main.py file

The ***main.py*** file. The comments start with a '#' and are self-explanatory

import webapp3

from google.appengine.ext import db

```
#creating a class instance of the database, google app engine allows us to do so. Whar t we
have done here is very conveniently created a db template and specified the data types it
can have for the 3 columns(temp/heat/fan)

class MyData(db.Model):
    temp = db.IntegerProperty()
    heat = db.IntegerProperty()
    fan = db.IntegerProperty()

#/hello redirects to the class Hello world as can be seen in the URL container below
class HelloWorld(webapp2.RequestHandler):
    def get(self):
        temp = self.request.get('temp')
        heat = self.request.get('heat')
        fan = self.request.get('fan')
        data = MyData()
        data.temp = int(temp)
        data.heat = int(heat)
        data.fan = int(fan)
        data.put()

#/display redirects to the class Fetcher as can be seen in the URL container below
class Fetcher(webapp2.RequestHandler):
    def get(self):
        data = db.GqlQuery("select * from MyData")

        for x in data:
            str = "TEMP= %d, HEAT=%d, FAN=%d <br>" %(x.temp,x.heat,x.fan)
            self.response.write(str)

#URL container
app = webapp2.WSGIApplication([
    ('/hello', HelloWorld),
    ('/display', Fetcher)
], debug=True)
```

APPENDIX 2

The Back End

Reusing of the driver modules to implement a backend that responds to the app-engine and reads/writes values from the SBHS(Single Board Heater System) is implemented here.

The forked sbhs driver module on top of which the development took place (Source: https://github.com/prashants/sbhs)

All comments begin with '#'. However the first line is known as shebang and is mandatory if we want to execute the script by a command like ./pythonfilename.py

main2.py

#!/usr/bin/python -tt

# the serial module is already available with python's distributed package. It is used for establishing data connection with the SBHS device, the time module is used to re-iterate the code automatically by using the sleep function.

```
import serial
import time
from time import sleep
```

# A data channel is opened and the communication with the device established

```
ser = serial.Serial('/dev/ttyUSB0', baudrate=9600, timeout=1)
ser.open()
```

# these libraries are imported from python-dist-packages to parse/read URL

```
import urllib
import urllib2
```

# this line imports everything from the sbhs.py & scan_machines.py file. It can be found on the URL https://github.com/prashants/sbhs. It is much more convenient to clone the git repository using the command ***git clone https://github.com/prashants/sbhs.git.*** *in terminal in Ubuntu.* The sbhs.py and scan_machines.py files should be located and put in the same directory as the **main2.py** file being edited.

```
from sbhs import *
from scan_machines import *


new_device = Sbhs()
new_device.connect(80)
new_device.connect_device(0)
while True:

        new_device.setHeat(10)
        new_device.setFan(80)
        f=new_device.getTemp()
        print f
```

#writing to google app-engine, we are using the same protocol to read/write values to the google-app engine.

```
        data = {}
        data['temp'] = '30'#float value not int
        data['heat'] = '200'
        data['fan'] = '100'
        url_val = urllib.urlencode(data)
        print url_val


        url = 'http://remote-cloudlabs.appspot.com/hello'
        full_url = url + '?' + url_val
        data = urllib2.urlopen(full_url)
```

#reading from url

```
        req =urllib2.Request('http://remotecloudlabs.appspot.com/display')
        response = urllib2.urlopen(req)
        data = response.read()
        print data
       time.sleep(10)
```

Note:

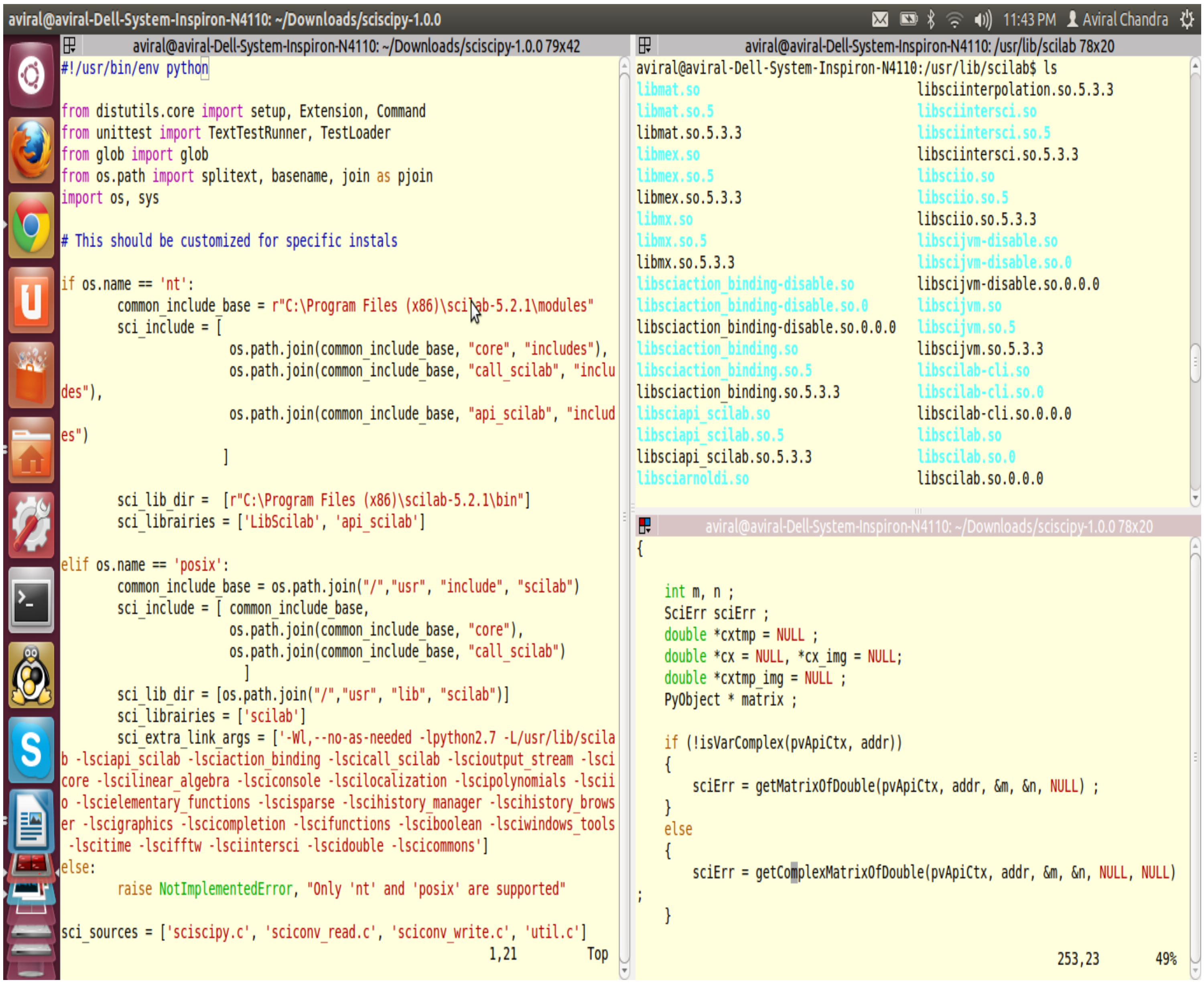
What has been implemented in the above two sections is the middle-ware and the backend that can respond to any front-end. The client end just has to parse the URL and exchange values with the middle-ware which is the google app-engine in order to interact with the back end

APPENDIX 3

Sciscipy changes that made it usable

All the shared object files commonly known as .so files are listed in the top-right screen partition. They are intended to be used by executable files.

The linker error as shown in the lower right window, was sorted out by including these so files in the sci_extra_link_args. The configuration was done by running the setup.py file included with sciscipy.



```
aviral@aviral-Dell-System-Inspiron-N4110: ~/Downloads/sciscipy-1.0.0                11:43 PM  Aviral Chandra
```

```
aviral@aviral-Dell-System-Inspiron-N4110: ~/Downloads/sciscipy-1.0.0 79x42
#!/usr/bin/env python

from distutils.core import setup, Extension, Command
from unittest import TextTestRunner, TestLoader
from glob import glob
from os.path import splitext, basename, join as pjoin
import os, sys

# This should be customized for specific instals

if os.name == 'nt':
        common_include_base = r"C:\Program Files (x86)\scilab-5.2.1\modules"
        sci_include = [
                        os.path.join(common_include_base, "core", "includes"),
                        os.path.join(common_include_base, "call_scilab", "inclu
des"),
                        os.path.join(common_include_base, "api_scilab", "includ
es")
                      ]

        sci_lib_dir =  [r"C:\Program Files (x86)\scilab-5.2.1\bin"]
        sci_librairies = ['LibScilab', 'api_scilab']

elif os.name == 'posix':
        common_include_base = os.path.join("/","usr", "include", "scilab")
        sci_include = [ common_include_base,
                        os.path.join(common_include_base, "core"),
                        os.path.join(common_include_base, "call_scilab")
                          ]
        sci_lib_dir = [os.path.join("/","usr", "lib", "scilab")]
        sci_librairies = ['scilab']
        sci_extra_link_args = ['-Wl,--no-as-needed -lpython2.7 -L/usr/lib/scila
b -lsciapi_scilab -lsciaction_binding -lscicall_scilab -lscioutput_stream -lsci
core -lscilinear_algebra -lsciconsole -lscilocalization -lscipolynomials -lscii
o -lscielementary_functions -lscisparse -lscihistory_manager -lscihistory_brows
er -lscigraphics -lscicompletion -lscifunctions -lsciboolean -lsciwindows_tools
 -lscitime -lscifftw -lsciintersci -lscidouble -lscicommons']
else:
        raise NotImplementedError, "Only 'nt' and 'posix' are supported"

sci_sources = ['sciscipy.c', 'sciconv_read.c', 'sciconv_write.c', 'util.c']
                                                            1,21          Top
```

```
aviral@aviral-Dell-System-Inspiron-N4110: /usr/lib/scilab 78x20
aviral@aviral-Dell-System-Inspiron-N4110:/usr/lib/scilab$ ls
libmat.so                                libsciinterpolation.so.5.3.3
libmat.so.5                              libsciintersci.so
libmat.so.5.3.3                          libsciintersci.so.5
libmex.so                                libsciintersci.so.5.3.3
libmex.so.5                              libsciio.so
libmex.so.5.3.3                          libsciio.so.5
libmx.so                                 libsciio.so.5.3.3
libmx.so.5                               libscijvm-disable.so
libmx.so.5.3.3                           libscijvm-disable.so.0
libsciaction_binding-disable.so          libscijvm-disable.so.0.0.0
libsciaction_binding-disable.so.0        libscijvm.so
libsciaction_binding-disable.so.0.0.0    libscijvm.so.5
libsciaction_binding.so                  libscijvm.so.5.3.3
libsciaction_binding.so.5                libscilab-cli.so
libsciaction_binding.so.5.3.3            libscilab-cli.so.0
libsciapi_scilab.so                      libscilab-cli.so.0.0.0
libsciapi_scilab.so.5                    libscilab.so
libsciapi_scilab.so.5.3.3                libscilab.so.0
libsciarnoldi.so                         libscilab.so.0.0.0
```

```
aviral@aviral-Dell-System-Inspiron-N4110: ~/Downloads/sciscipy-1.0.0 78x20
{

    int m, n ;
    SciErr sciErr ;
    double *cxtmp = NULL ;
    double *cx = NULL, *cx_img = NULL;
    double *cxtmp_img = NULL ;
    PyObject * matrix ;

    if (!isVarComplex(pvApiCtx, addr))
    {
        sciErr = getMatrixOfDouble(pvApiCtx, addr, &m, &n, NULL) ;
    }
    else
    {
        sciErr = getComplexMatrixOfDouble(pvApiCtx, addr, &m, &n, NULL, NULL)
;
    }
                                                            253,23        49%
```

APPENDIX Figure 2: scscipy changes

Configuration was done by typing ./setup.py on terminal

The compilation was done by ./setup.py build

The installation was completion using ./setup.py build install

APPENDIX 4

Building GRAS from source

The steps and the command snippet are given as under.

Steps:

1) Go to the gras directory.

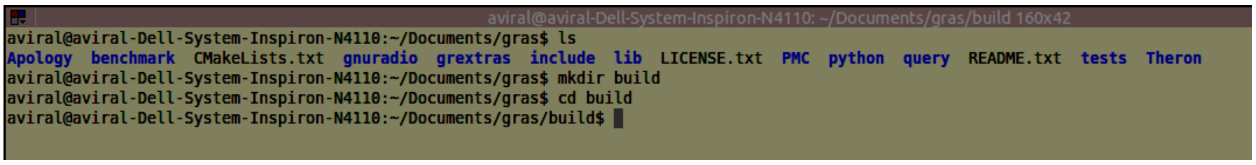
2) Make a build directory using mkdir build

3) Go to the build directory using cd build

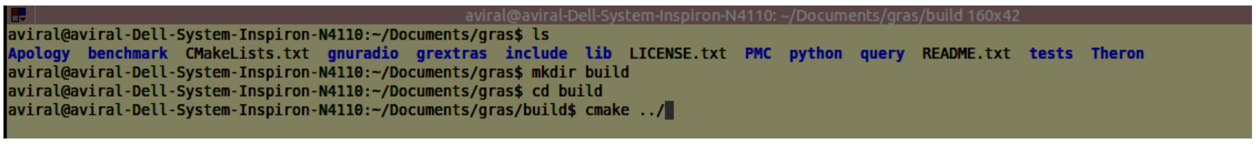
4) Type cmake ../ to start the configuration process

5) Type sudo make to complete the compilation

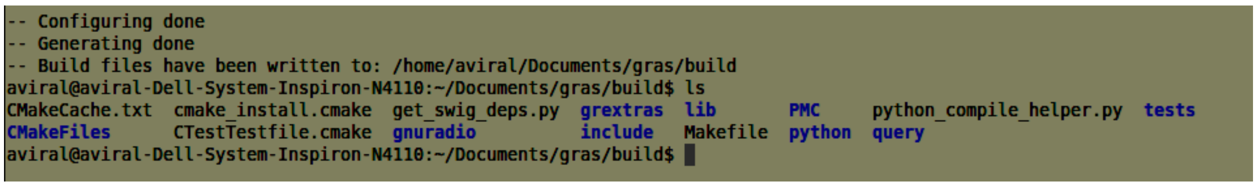
6) Type make install to make it available to the user



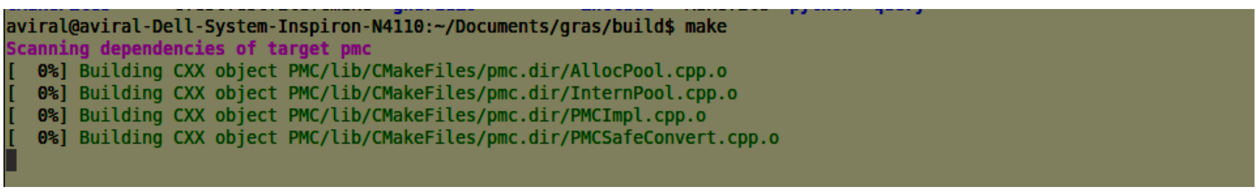
```
aviral@aviral-Dell-System-Inspiron-N4110: ~/Documents/gras/build 160x42
aviral@aviral-Dell-System-Inspiron-N4110:~/Documents/gras$ ls
Apology  benchmark  CMakeLists.txt  gnuradio  grextras  include  lib  LICENSE.txt  PMC  python  query  README.txt  tests  Theron
aviral@aviral-Dell-System-Inspiron-N4110:~/Documents/gras$ mkdir build
aviral@aviral-Dell-System-Inspiron-N4110:~/Documents/gras$ cd build
aviral@aviral-Dell-System-Inspiron-N4110:~/Documents/gras/build$ 
```



```
aviral@aviral-Dell-System-Inspiron-N4110: ~/Documents/gras/build 160x42
aviral@aviral-Dell-System-Inspiron-N4110:~/Documents/gras$ ls
Apology  benchmark  CMakeLists.txt  gnuradio  grextras  include  lib  LICENSE.txt  PMC  python  query  README.txt  tests  Theron
aviral@aviral-Dell-System-Inspiron-N4110:~/Documents/gras$ mkdir build
aviral@aviral-Dell-System-Inspiron-N4110:~/Documents/gras$ cd build
aviral@aviral-Dell-System-Inspiron-N4110:~/Documents/gras/build$ cmake ../
```



```
-- Configuring done
-- Generating done
-- Build files have been written to: /home/aviral/Documents/gras/build
aviral@aviral-Dell-System-Inspiron-N4110:~/Documents/gras/build$ ls
CMakeCache.txt  cmake_install.cmake  get_swig_deps.py  grextras  lib       PMC     python_compile_helper.py  tests
CMakeFiles      CTestTestfile.cmake  gnuradio          include   Makefile  python  query
aviral@aviral-Dell-System-Inspiron-N4110:~/Documents/gras/build$ 
```



```
aviral@aviral-Dell-System-Inspiron-N4110:~/Documents/gras/build$ make
Scanning dependencies of target pmc
[  0%] Building CXX object PMC/lib/CMakeFiles/pmc.dir/AllocPool.cpp.o
[  0%] Building CXX object PMC/lib/CMakeFiles/pmc.dir/InternPool.cpp.o
[  0%] Building CXX object PMC/lib/CMakeFiles/pmc.dir/PMCImpl.cpp.o
[  0%] Building CXX object PMC/lib/CMakeFiles/pmc.dir/PMCSafeConvert.cpp.o
```

APPENDIX Figure 3: Series of screenshots illustrating GRAS build

The last step after this is make install. This completes the installation process and the Advanced Scheduler is ready to use

APPENDIX 5

dsimul development logic was the same as csim. Only a single python file had changes. Only the file which had changes is given here.

```
#!/usr/bin/python

import sciscipy

# u is a TUPLE vector of width w

def discrete_sim(P,I,D,n0,n1,st,d0,d1,u):
        code_string1 = "s=%s;"
        code_string2 = "Gc=sys-
lin("+str(st)+",("+str(P*I+D)+"*s)"+","+str(I)+"*s);"
        code_string3 = "G=syslin("
        code_string4 = str(st) +","+ str(n0)+"*s"+ "+"+str(n1)+","+
str(d0)+"*s"+"+"+str(d1)+");"
        code_string5 = "r=tf2ss(G*Gc);"
        code_string6 = "u="+str((u))+";"
        code_string7 = "y=dsimul(r,u)"
        code_string =code_string1 + code_string2+ code_string3+
code_string4 + code_string5+code_string6+code_string7

        # Check complete_code_string
        #print code_string

        import sciscipy
        sciscipy.eval(code_string)
        y = sciscipy.read("y")
        return y

#print discrete_sim(1,1,0.1,2,1,"u=zeros(1,50);u(10)=1")

if __name__ == "__main__":
        u = [0]*100
        u[50] = 1
        out = discrete_sim(2,0.5,0.6,1,1,0.1,2,1,u)
        print out

        #import matplotlib.pyplot as plt
        #plt.plot(out)
        #plt.show()
```

APPENDIX 6

Copyright terms of GNU Radio