Tutorial
Creating a C Function Block in Scicos

Phil Schmidt

March 7, 2009

Table of Contents

1 Introduction

This document is a tutorial that describes the process of creating a custom Scicos function block in C. Scicos is a graphical dynamic system modeling tool that is contained within Scilab, a Matlab-like numerical analysis and simulation package.

I have created this tutorial for a couple reasons. First, this is my way to leave breadcrumbs for myself, so that whenever I create my own function blocks, I have an example to follow. Second, I found that the available Scicos documentation is sorely lacking in complete examples with thorough descriptions of the steps to take. I spent many hours poring over what documentation I could find, posting questions to the Scilab newsgroup, and engaging in plain trial-and-error, before I was able to successfully build my own function blocks. I hope that by creating this tutorial, I can compress your learning curve and help you to quickly become productive.

Regarding the development environment, I have used ScicosLab 4.3 on Ubuntu Linux to create the example presented in this tutorial. If you are using Windows, I can't say whether this stuff will work exactly the same, though most of it should.

So, let's get started!

2 Defining the Model

2.1 Equations for The Example Block

For this tutorial, we will create a custom integrator block that has constant high and low limits, and variable gain. The equations of the block are as follows:

$$\begin{array}{ll} \dot{x}=g_p u & L_{hi}>x>0 \\ \dot{x}=g_p u & x=L_{hi}, u\le 0 \\ \dot{x}=g_n u & 0\ge x>L_{lo} \\ \dot{x}=g_n u & x=L_{lo}, u\ge 0 \\ \dot{x}=0 & x\ge L_{hi}, u>0 \\ \dot{x}=0 & x\le L_{lo}, u<0 \end{array} \tag{1}$$

where u, g_p and g_n are inputs to the block and L_{hi} and L_{lo} are block parameters.

In addition to defining the block's state, we will also want to define its output. We will choose the state variable as one output, and the block's current gain (g_p, g_n or zero) as a second output.

Notice in equations (1) the careful definition of the discontinuities in the derivative at L_{hi} and L_{lo}. This careful definition is important for the numerical integrator in Scicos, which needs to know where the discontinuities are in order to properly simulate the system. This leads us to the next section...

2.2 Modes, Surfaces and Zero-Crossings

Generally, Scicos needs to know where there are discontinuities in the derivatives of the system in order to properly simulate the system. To define these, Scicos makes use of modes and surfaces, which you must define in your block code, to determine where those discontinuties are and what to do on either side of them.

A surface is simply a variable in your system that has values that cross discontinuities in the system. Scicos detects that a surface is crossing a discontinuity by detecting that the surface variable crosses zero.

A mode is a variable that holds values representing each distinct region that exist in a surface. So, if a surface has a single zero-crossing, then there would be two mode values to represent that the system is on one or the other side of the zero crossing.

In the system in equations (1), there are three surfaces, defined as $s_0 = x - L_{hi}$, $s_1 = x - L_{lo}$ and $s_2 = x$, The three surfaces therefore each have a zero crossing; one at L_{hi}, one at zero and one at $L_{lo.}$ There is a single mode variable that can have four discrete values representing each of the four zero-crossing regions. We will make the following definitions for mode:

$$\begin{aligned} mode&=1 & x&\geq L_{hi} \\ mode&=2 & 0<x&<L_{hi} \\ mode&=3 & L_{lo}<x&\leq 0 \\ mode&=4 & x&\leq L_{lo} \end{aligned} \qquad (2)$$

3 Creating the Computational Function

Once you've defined the equtions for the block, you are ready to write the C code that implements it. In the code will be a function that Scicos calls to compute the behavior of the block. This is called the computational function.

3.1 The Code

For our example block, I will simply present the code here, and then comment on key parts of the code. So, here is the code, saved in the file **`lim_int_comp.c`**:

```
// This is the computational function for a Scicos model block.
// The model is of a variable-gain integrator with hard high and low limits.


#include "scicos/scicos_block4.h"
#include <math.h>
#include <stdio.h>
#include <stdlib.h>

#define r_IN(n, i)    ((GetRealInPortPtrs(blk, n+1))[(i)])
#define r_OUT(n, i)   ((GetRealOutPortPtrs(blk, n+1))[(i)])

// parameters
#define Lhi     (GetRparPtrs(blk)[0])      // integrator high limit
#define Llo     (GetRparPtrs(blk)[1])      // integrator low limit

// inputs
#define in      (r_IN(0,0))         // integrator input
#define gainp   (r_IN(1,0))         // integrator gain when X > 0
#define gainn   (r_IN(2,0))         // integrator gain when X <= 0

// states
#define X       (GetState(blk)[0])          // integrator state
#define Xdot    (GetDerState(blk)[0])       // derivative of the integrator output

// outputs
#define out     (r_OUT(0, 0))       // integrator output
#define Igain   (r_OUT(1, 0))       // integrator gain

// other constants
#define surf0   (GetGPtrs(blk)[0])
#define surf1   (GetGPtrs(blk)[1])
#define surf2   (GetGPtrs(blk)[2])
#define mode0   (GetModePtrs(blk)[0])


// if X is greater than Lhi, then mode is 1
// if X is between Lhi and zero, then mode is 2
// if X is between zero and Llo, then mode is 3
// if X is less than Llo, then mode is 4
#define mode_xhzl    1
#define mode_hxzl    2
#define mode_hzxl    3
#define mode_hzlx    4

void lim_int(scicos_block *blk, int flag)
  {
  double gain = 0;

  switch (flag)
    {
    case 0:
      // compute the derivative of the continuous time state
      if ((mode0 == mode_xhzl && in < 0) || mode0 == mode_hxzl)
        gain = gainp;
      else if ((mode0 == mode_hzlx && in > 0) || mode0 == mode_hzxl)
        gain = gainn;
      Xdot = gain * in;
      break;

    case 1:
```

```
62       // compute the outputs of the block
63       if (X >= Lhi || X <= Llo)
64         Igain = 0;
65       else if (X > 0)
66         Igain = gainp;
67       else
68         Igain = gainn;
69       out = X;
70       break;
71
72     case 9:
73       // compute zero crossing surfaces and set modes
74       surf0 = X - Lhi;
75       surf1 = X;
76       surf2 = X - Llo;
77
78       if (get_phase_simulation() == 1)
79         {
80         if (surf0 >= 0)
81           mode0 = mode_xhzl;
82         else if (surf2 <= 0)
83           mode0 = mode_hzlx;
84         else if (surf1 > 0)
85           mode0 = mode_hxzl;
86         else
87           mode0 = mode_hzxl;
88         }
89       break;
90     }
91   }
```

3.2 Commentary on the Code

Line 5: Be sure to #include “scicos_block4.h”. This header defines macros that are used to access data elements in the *scicos_block* data structure. The macros used in this example are *GetRealInPortPtrs* (line 10), *GetRealOutPortPtrs* (line 11), *GetRparPtrs* (lines 14-15), *GetState* (line 23), *GetDerState* (line 24), *GetGPtrs* (lines 31-33) and *GetModePtrs* (line 34). (More information on these and other macros is available from the Scilab online help. Type *help* at the Scilab command prompt, then go to **Scicos: Bloc diagram editor and simulator | whatis scicos | C Macros**.)

Lines 10-44 define a bunch of macros that make it easier to access elements of the Scicos block data structure. I find it much easier to use names that correspond to each of my block’s parameters, states, inputs and outputs, rather than to be constantly working with anonymous pointers and array elements.

Lines 10-11 are helper macros for accessing block inputs and outputs.

Lines 14-15 name the upper and lower limit of the integrator. The limits are passed to the block as real parameters.

Lines 18-20 name the inputs to the integrator block. For this block, all the inputs are real-valued values.

Lines 23-24 name the block’s state and derivative.

Lines 27-28 name the outputs of the integrator block. For this block, all the outputs are real-valued values.

Lines 31-33 name the three surfaces in the block.

Line 34 names the single mode variable of the block.

Lines 41-44 name the four mode values that the mode variable may hold.

Lines 46-91 is the function that implements the behavior of the new block.

Line 46: The function name, *lim_int*, will be used later when linking the code into Scicos.

Line 52-59: When flag is zero, the Scicos integrator is requesting the block to compute its derivative. The derivative equation to be used depends on the mode, which can be seen in lines 57 and 59. Compare the equations in this section of the code to the equations given in sections 2 and 3.

Lines 61-70: When flag is one, the Scicos integrator is requesting the block to compute its outputs. As you can see, the mode is not used to determine which gain value to use as output. I could not find documentation explaining whether mode should be used, but in all the example code I looked at, it was not used, so my assumption is that it should not be used here.

Lines 72-89: When flag is nine, the Scicos integrator is requesting information about modes and zero crossings. Surface information is needed at every call, while mode information is only needed during simulation phase 1.

Lines 74-76: The values of the three surfaces are assigned here.

Line 78: This is the test for simulation phase one.

Lines 80-87: Modes are calculated here based on the positions of the surface values.

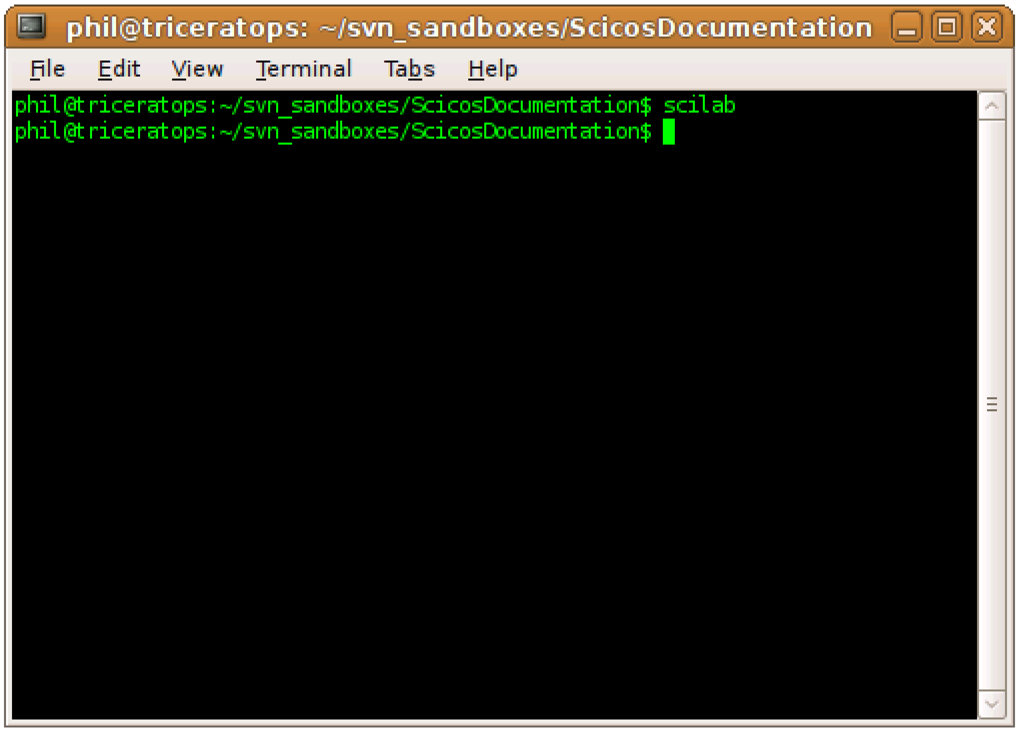
4 Compiling and Linking

With the C code in hand, you are now ready to compile and link it into Scilab.

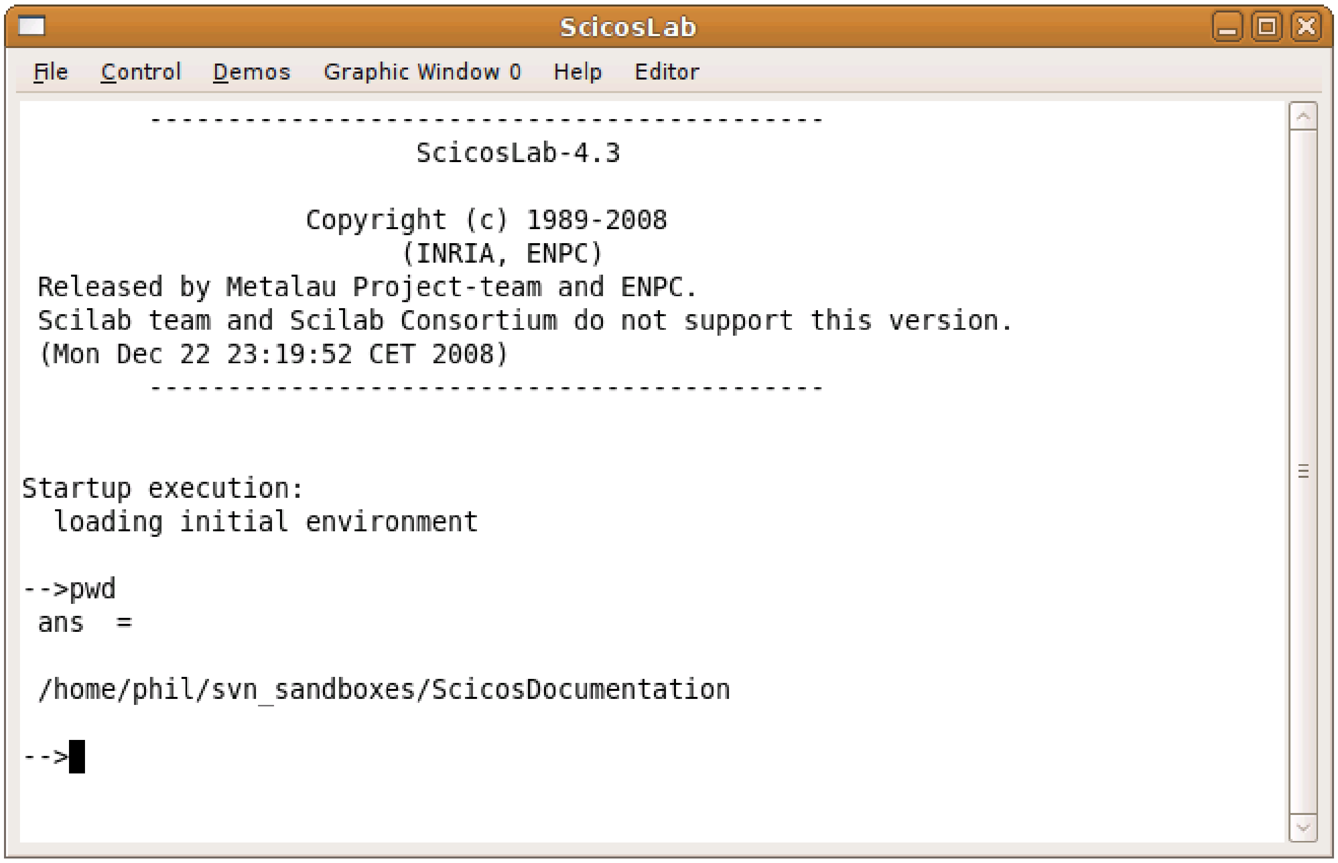
4.1 Starting Scilab

First, run Scilab. You will need to have Scilab running in the directory that contains your source file. The easiest way to do that is to just open a terminal in your working directory, and invoke Scilab from there. Otherwise, use the ***chdir()*** function inside the Scilab console to change to your working directory.

The screenshots below show how I invoke Scilab for this tutorial:


phil@triceratops: ~/svn_sandboxes/ScicosDocumentation
File Edit View Terminal Tabs Help

```
phil@triceratops:~/svn_sandboxes/ScicosDocumentation$ scilab
phil@triceratops:~/svn_sandboxes/ScicosDocumentation$ 
```



```
        -------------------------------------------
                         ScicosLab-4.3

                  Copyright (c) 1989-2008
                        (INRIA, ENPC)
 Released by Metalau Project-team and ENPC.
 Scilab team and Scilab Consortium do not support this version.
 (Mon Dec 22 23:19:52 CET 2008)
        -------------------------------------------


Startup execution:
  loading initial environment

-->pwd
 ans  =

 /home/phil/svn_sandboxes/ScicosDocumentation

-->
```

4.2 Invoking the Compiler and Linker

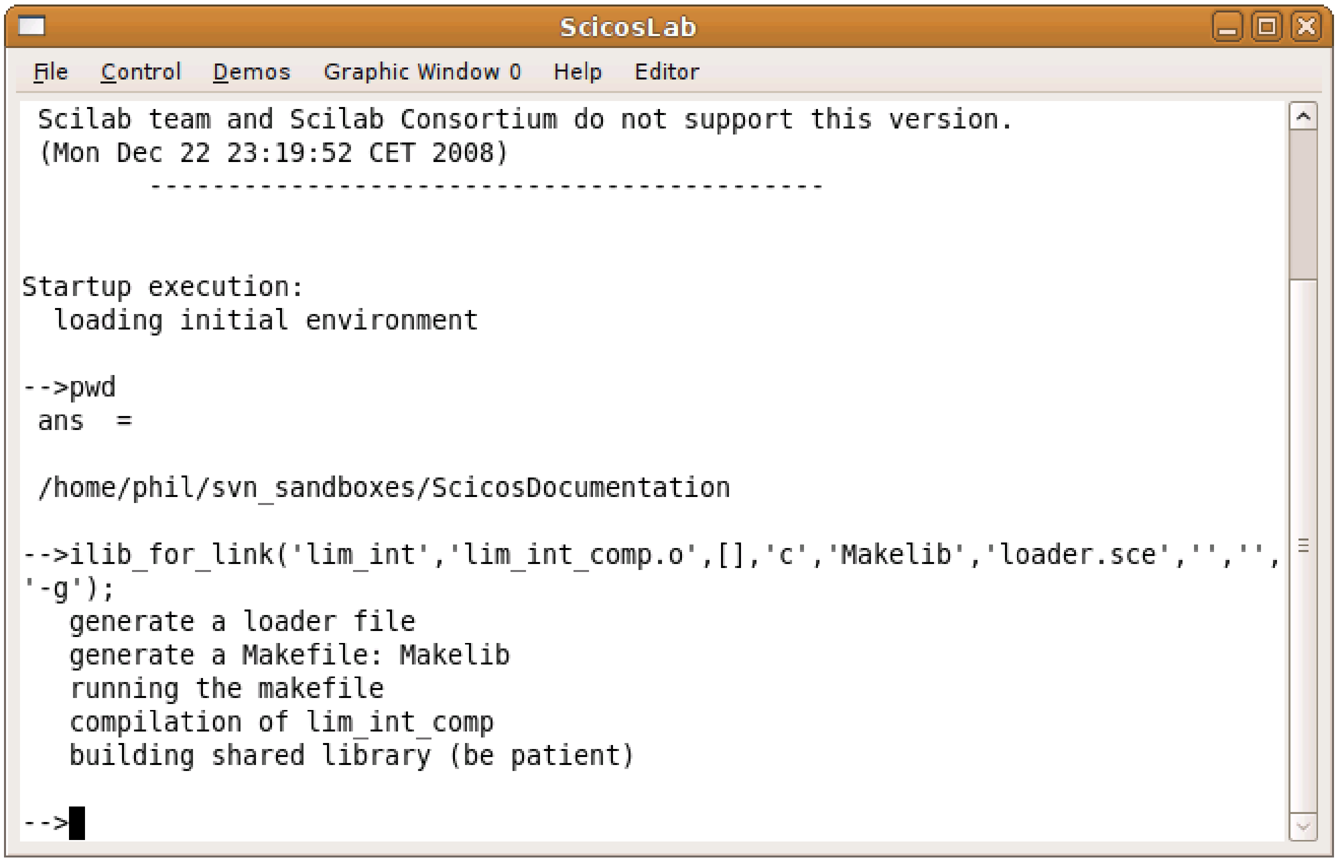
Ok, so Scilab is running and is in the right directory. Compiling and linking is done simply by invoking the following command in Scilab:

```
ilib_for_link('lim_int','lim_int_comp.o',[],'c','Makelib','loader.sce','','','-g');
```

This is kind of a scary looking command, but it's pretty simple once you break it down (and note that you can get help on this command from Scilab's built-in help browser in the *Utilities* category).

- The first parameter, `'lim_int'`, is the name of the function we want linked into Scilab.
- The 2nd parameter, `'lim_int_comp.o'`, is the name of the object file that will be created when our source code is compiled.
- The 3rd parameter, `[]`, is a list of extra libraries needed for linking. Our example does not need any, so an empty matrix is passed.
- The 4th parameter, `'c'`, tells Scilab that this is a C function (as opposed to a Fortran function).
- Parameters 5 and 6, `'Makelib'` and `'loader.sce'`, are the names of the makefile and loader file respectively. Both of the values I used are also the default values.
- Parameters 7 and 8 are the libname and ldflags. For this example, I just pass empty strings, which also happen to be the defaults.
- Finally, parameter 8 is a list of the cflags to pass to the C compiler. I pass in a `'-g'` to make the compiler generate debugging information. This is absolutely necessary if your C code has run-time errors, so that you can use gdb to debug it.

When you run the command, you should get something that looks like the following picture:



```
 Scilab team and Scilab Consortium do not support this version.
 (Mon Dec 22 23:19:52 CET 2008)
        -------------------------------------------


Startup execution:
  loading initial environment

-->pwd
 ans  =

 /home/phil/svn_sandboxes/ScicosDocumentation

-->ilib_for_link('lim_int','lim_int_comp.o',[],'c','Makelib','loader.sce','','',
'-g');
   generate a loader file
   generate a Makefile: Makelib
   running the makefile
   compilation of lim_int_comp
   building shared library (be patient)

-->
```

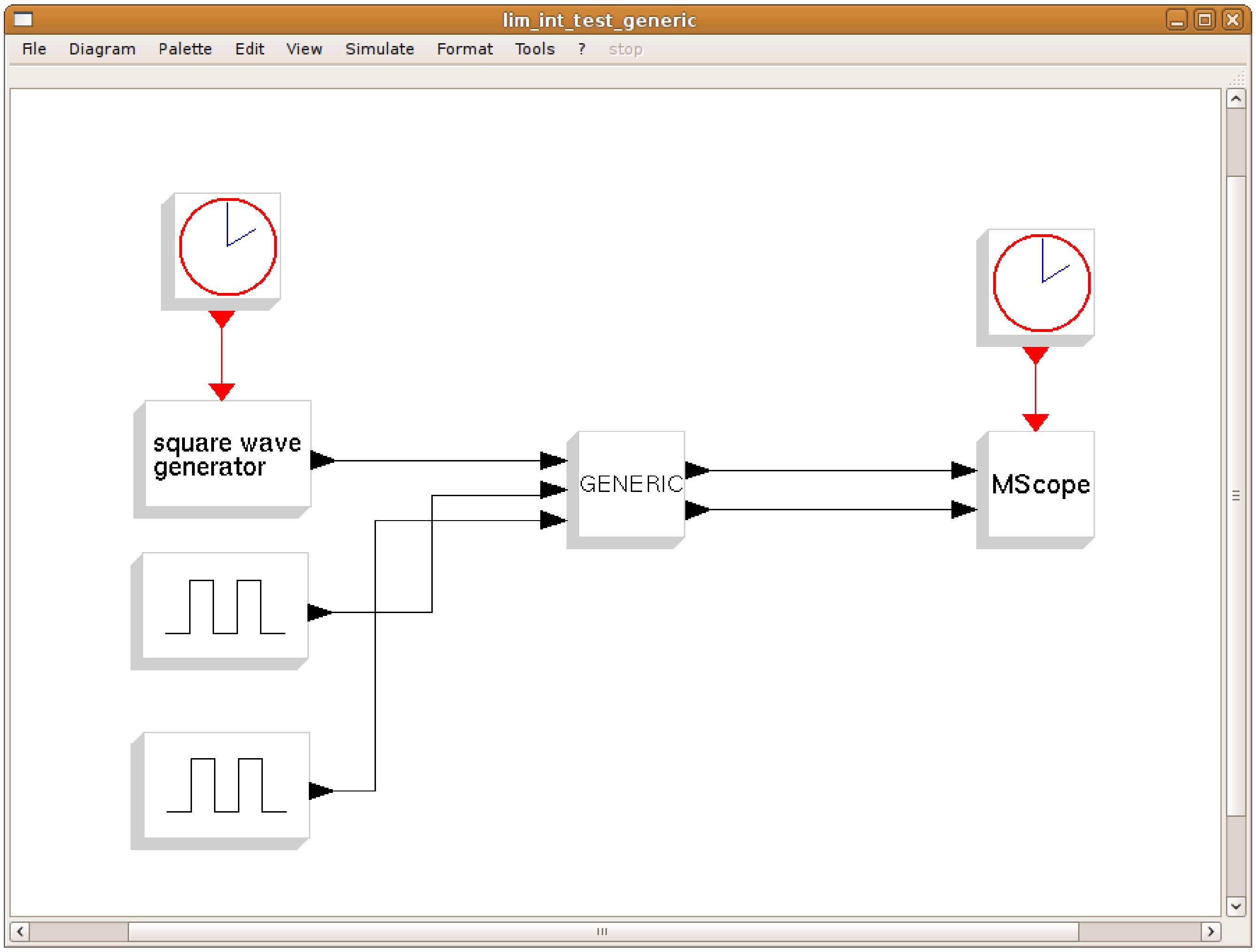
That's it! The C file is compiled and linked!

5 Testing the New Block

Now that we've linked the code, it would be nice to run it to try it out. We can do a quick-and-dirty test using the GENERIC block in Scicos. Create a new Scicos diagram, and follow the instructions below.

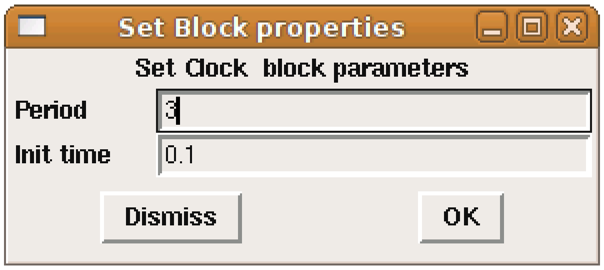
5.1 Block Diagram

The following picture shows the block diagram for the system. Open the Palette Tree and place the necessary blocks onto the diagram and connect them as shown. You will need two **CLOCK_c** blocks, one **GENSQR_f** block, two **GEN_SQR** blocks, one **generic_block3** block and a **CMSCOPE** block.



5.2 Block Configurations

5.2.1 LeftHand CLOCK_c Block

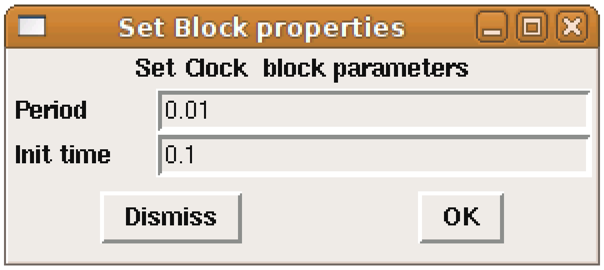


Set Clock block parameters

Period: 3

Init time: 0.1

5.2.2 Righthand CLOCK_c Block

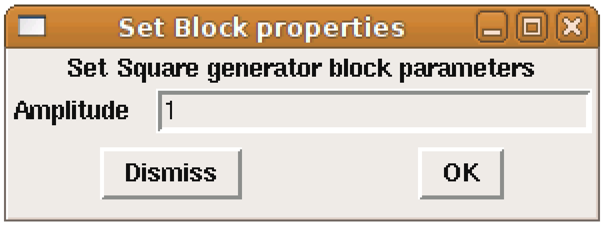


Set Clock block parameters

Period 0.01

Init time 0.1

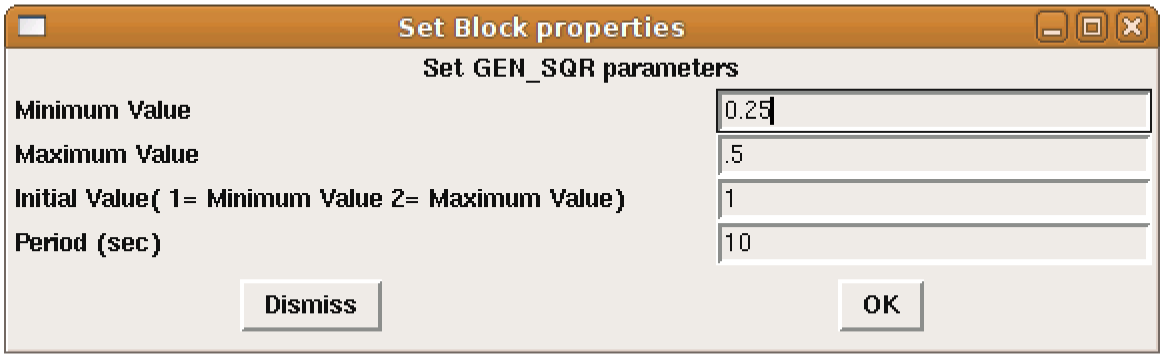
5.2.3 GENSQR_f Block



Set Square generator block parameters

Amplitude 1

5.2.4 Middle GEN_SQR Block



Set GEN_SQR parameters

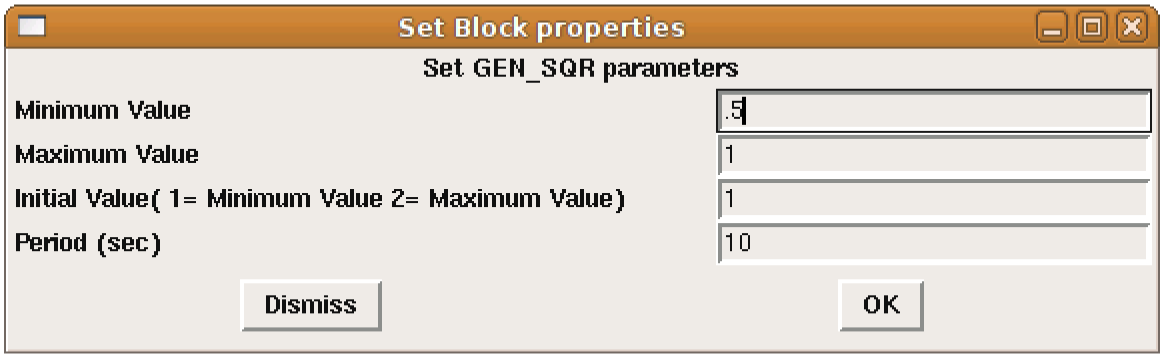
Minimum Value 0.25

Maximum Value .5

Initial Value(1= Minimum Value 2= Maximum Value) 1

Period (sec) 10

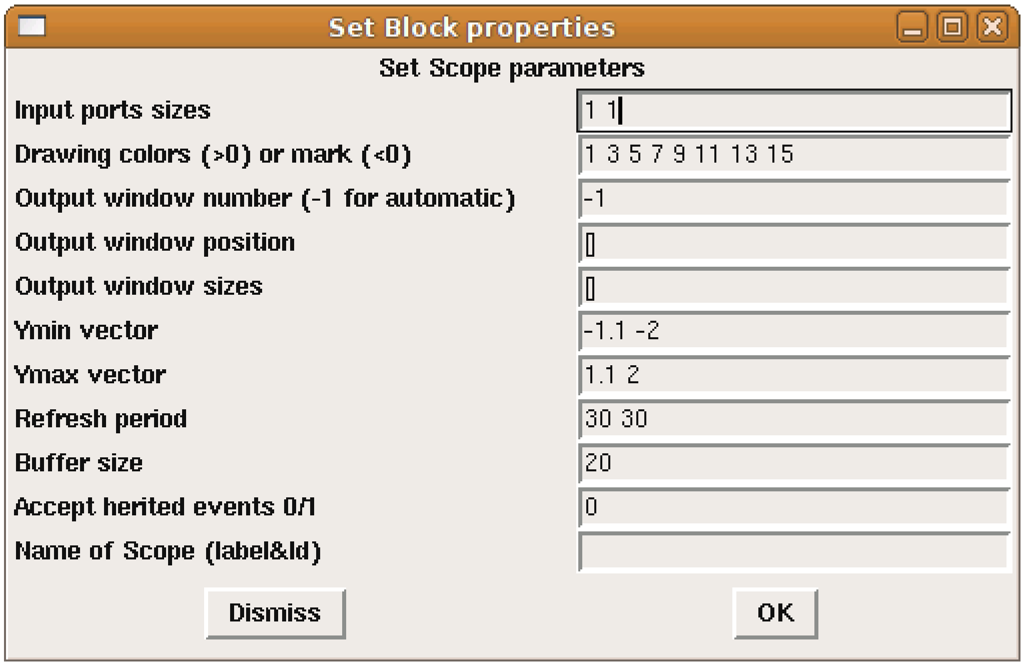
5.2.5 Bottom GEN_SQR Block



Set GEN_SQR parameters

Minimum Value	.5
Maximum Value	1
Initial Value(1= Minimum Value 2= Maximum Value)	1
Period (sec)	10

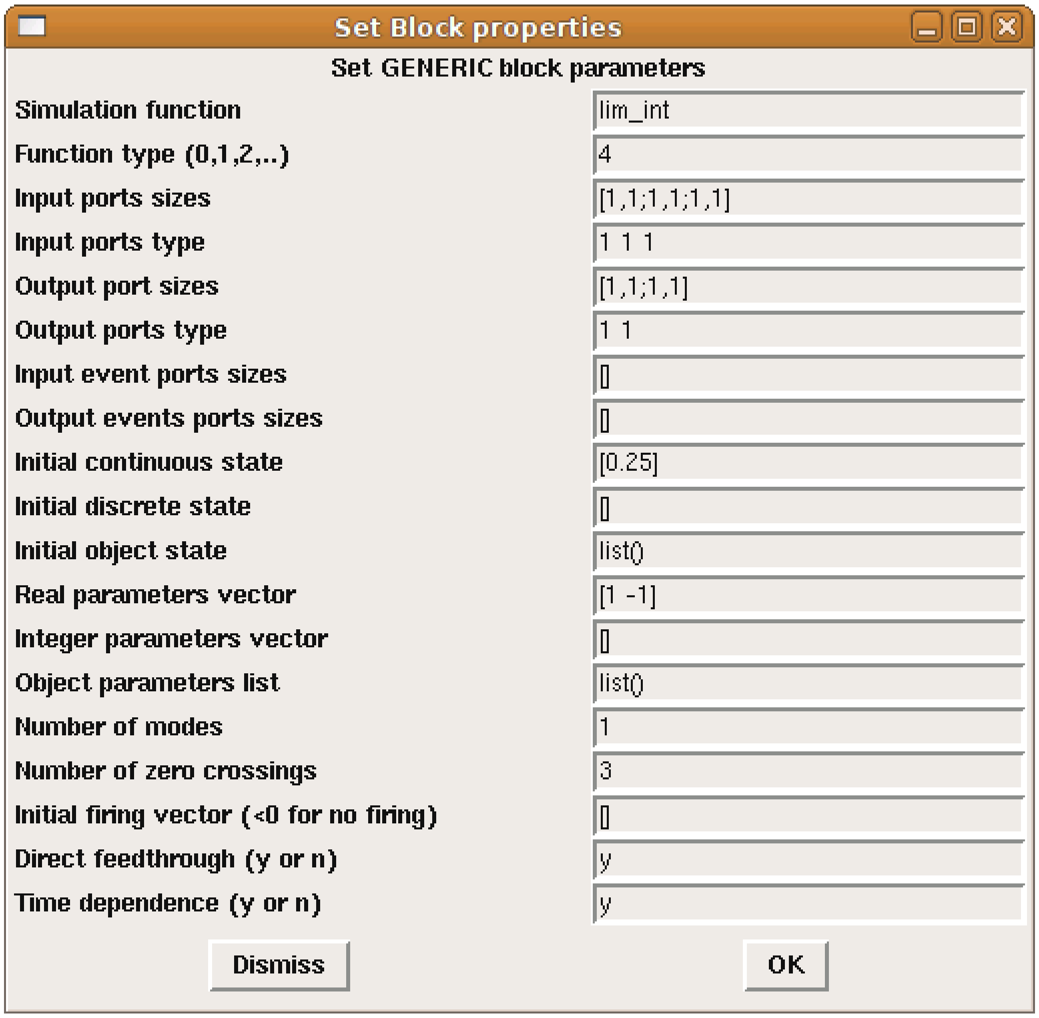
5.2.6 CMSCOPE Block



Set Scope parameters

Input ports sizes	1 1
Drawing colors (>0) or mark (<0)	1 3 5 7 9 11 13 15
Output window number (-1 for automatic)	-1
Output window position	[]
Output window sizes	[]
Ymin vector	-1.1 -2
Ymax vector	1.1 2
Refresh period	30 30
Buffer size	20
Accept herited events 0/1	0
Name of Scope (label&Id)	

5.2.7 generic_block3 Block



Set Block properties

Set GENERIC block parameters

Simulation function	lim_int
Function type (0,1,2,..)	4
Input ports sizes	[1,1;1,1;1,1]
Input ports type	1 1 1
Output port sizes	[1,1;1,1]
Output ports type	1 1
Input event ports sizes	[]
Output events ports sizes	[]
Initial continuous state	[0.25]
Initial discrete state	[]
Initial object state	list()
Real parameters vector	[1 -1]
Integer parameters vector	[]
Object parameters list	list()
Number of modes	1
Number of zero crossings	3
Initial firing vector (<0 for no firing)	[]
Direct feedthrough (y or n)	y
Time dependence (y or n)	y

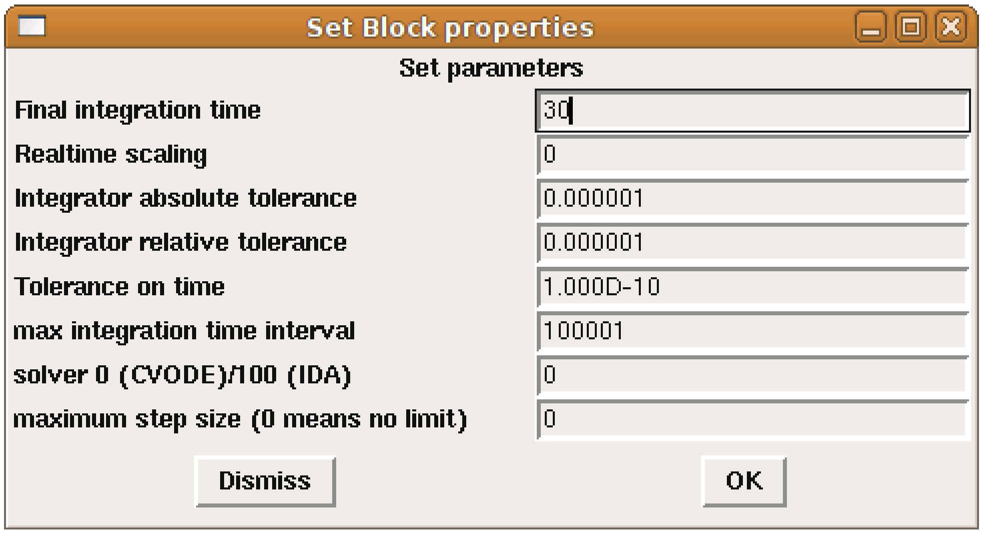
The configuration for this block deserves some explanation:

- Simulation function: This is the name of the linked C computational function that implements the block.
- Function type: A C block is type 4.
- Input ports sizes: The block has three 1x1 (scalar) inputs.
- Input ports type: The three inputs are of type *real*.
- Output ports sizes: The block has two 1x1 (scalar) outputs.
- Output ports type: The two outputs are of type *real*.
- The block has no input or output events.
- The initial condition for the blck's state is 0.25.
- There are no discrete states, thus no initial condition.

- Initial object state: *I have not figured out what this is yet!*
- The block's parameters (upper and lower limit) are real parameters.
- There are no integer parameters.
- There is one mode variable.
- There are three zero crossing surfaces.
- The block has direct feedthrough, i.e., it can form part of an algebraic loop.
- The block is time dependent, as it has a continuous state.

5.3 Running the Simulation

Now that our model is built, we can run it. First, set up the simulation parameters from the Simulate | Setup menu in the Scicos editor:

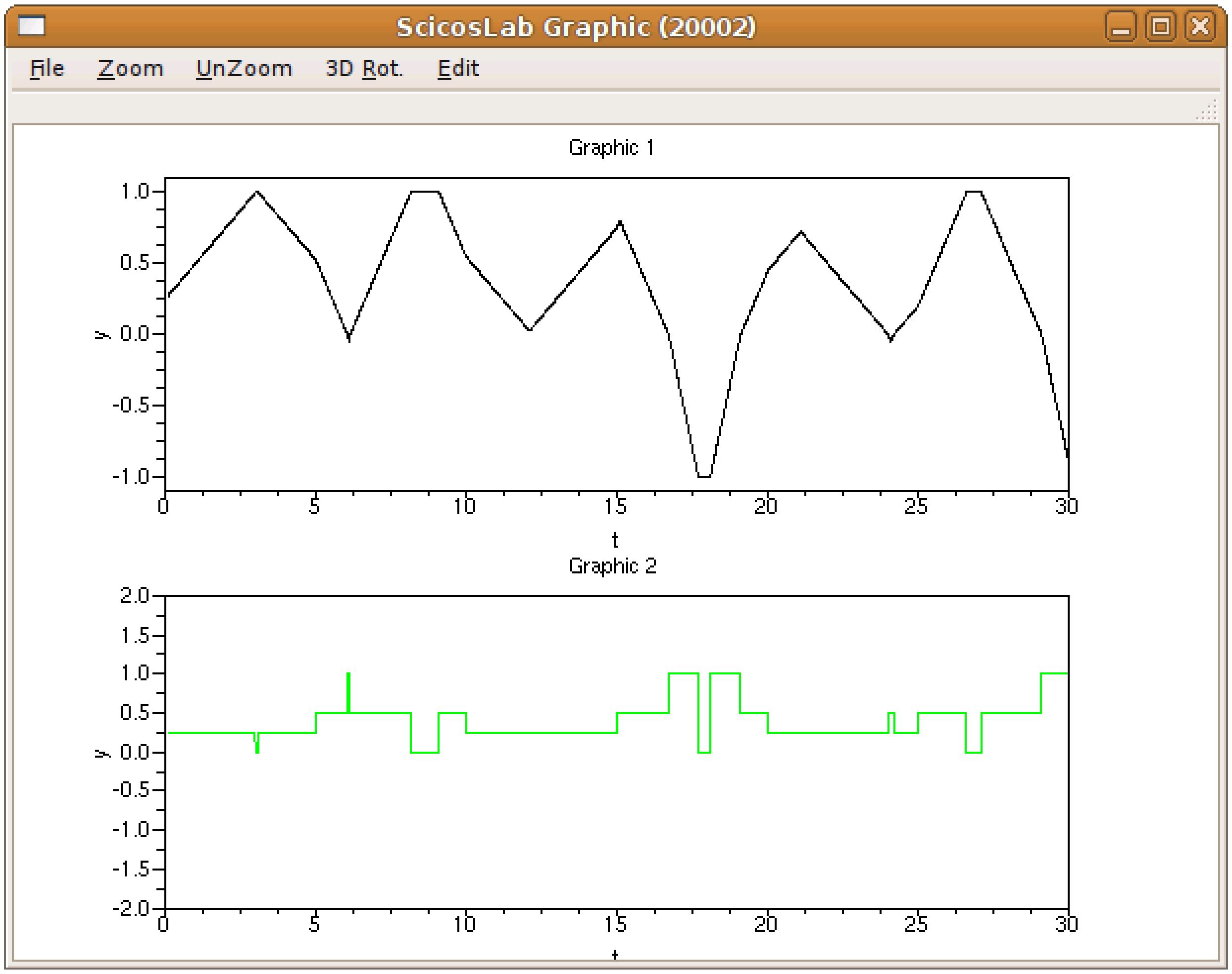


Set Block properties

Set parameters

Final integration time	30
Realtime scaling	0
Integrator absolute tolerance	0.000001
Integrator relative tolerance	0.000001
Tolerance on time	1.000D-10
max integration time interval	100001
solver 0 (CVODE)/100 (IDA)	0
maximum step size (0 means no limit)	0

Then run the simulation (Simulate | Run), to get the following plot:



And there you have it, a custom block written in C.

This is all great, but it would be nice to have a customized graphical block to go with the custom computational function. The next section shows how to do this.

6 Creating the Interfacing Function

The interfacing function is written in Scilab language, and is used to define the appearance and behavior of the graphical elements of the block in the Scicos editor. As I did with the computational function earlier, I will simply present the code, and then provide commentary afterwards.

6.1 The Code

Save the following code into the file **lim_int_intf.sci**.

```
1 function [x,y,typ] = LIMINT(job, arg1, arg2)
2
3   x = [];
4   y = [];
5   typ = [];
```

```

//disp(job)

select job
  case 'plot' then
    standard_draw(arg1)

  case 'getinputs' then
    [x,y,typ] = standard_inputs(arg1)
    //disp(sci2exp(x))

  case 'getoutputs' then
    [x,y,typ] = standard_outputs(arg1)
    //disp(sci2exp(x))

  case 'getorigin' then
    [x,y] = standard_origin(arg1)
    //disp(sci2exp(x))

  case 'set' then
    //message(sci2exp(arg1))
    x = arg1
    graphics = arg1.graphics
    exprs = graphics.exprs
    model = arg1.model
    while %t do
      [ok,Lhi,Llo,Xinitial,exprs] = getvalue('Set funny integrator parameters',..
        ['high limit';'low limit';'Xinitial'],..
        list('vec',1,'vec',1,'vec',1),..
        exprs)
      if ~ok then break,end
      model.state = [Xinitial]
      model.rpar = [Lhi;Llo]
      graphics.exprs = exprs
      x.graphics = graphics
      x.model = model
      break
    end

  case 'define' then
    //message('in define')
    Lhi = 1.0
    Llo = -1.0
    Xinitial = 0
    model = scicos_model()
    model.sim = list('lim_int',4)
    model.in = [1;1;1]
    model.out = [1;1]
    model.state = [Xinitial]
    model.dstate = [0]
    model.rpar = [Lhi;Llo]
    model.blocktype = 'c'
    model.nmode = 1
    model.nzcross = 3
    model.dep_ut = [%t %t]

    exprs = [string([Lhi;Llo;Xinitial])]
    gr_i = ['x=orig(1),y=orig(2),w=sz(1),h=sz(2)';
            'txt=[''Funny'';''Integrator'']';
            'xstringb(x+0.25*w, y+0.20*h, txt, 0.50*w, 0.60*h, ''fill'')';
            'txt=[''in'';'''';''gainp'';'''';''gainn'']';
            'xstringb(x+0.02*w, y+0.08*h, txt, 0.25*w, 0.80*h, ''fill'')';
            'txt=['''';''out'';'''';''gain'';'''']';
            'xstringb(x+0.73*w, y+0.08*h, txt, 0.25*w, 0.80*h, ''fill'')';
            ]
    x = standard_define([4 2],model,exprs,gr_i)

end

```

```
75 endfunction
```

6.2 Commentary on the Code

Before I begin with the commentary, let me point out that much of what's in the interfacing function I don't fully understand. I have started with known working examples and the available documentation to come up with this function. However, documentation is slim (which is one reason I'm writing this tutorial!). So, I will explain only what I understand.

Line 1: The name of the function, LIMINT, will be used inside the Scicos editor to access the function block. Keep this in mind for later.

Lines 7, 15, 19, 23, 26, 46: These are debugging lines. If you uncomment them, you will be able to see debugging information in the Scilab console and in dialog boxes while you construct and run the model.

Lines 10, 13, 17, 21: These cases just use the "standard" functions that are provided with Scilab. Good luck finding documentation on how these functions work (and if you do find it, please let me know!)

Lines 25-43: This case defines the behavior of the dialog box that appears when you double-click the block. It uses the getvalue() function in lines 32-35 to display the dialog and return the values entered. The initial condition of the model state is assigned in line 37, while the parameters, Lhi and Llo, are assigned to the model in line 38. All the return values are stored in the structure variable x in lines 40 and 41.

Lines 45-71: This case defines the appearance of the function block and initializes all the model variables when the block is first placed into the Scicos diagram.

Lines 50-60 define the block's model properties. Compare closely the assignments in lines 51-60 with the initializers in the GENERIC (generic_block3) block shown earlier:

- Line 51 corresponds to **Simulation function** and **Function type**.
- Line 52 corresponds to **Input ports sizes**.
- Line 53 corresponds to **Output ports sizes**.
- Line 54 corresponds to **Initial continuous state**.
- Line 55 corresponds to **Initial discrete state**. I found that this cannot be an empty vector, thus I assigned a value (which is unused by the computational function) to make Scilab happy.
- Line 56 corresponds to **Real parameters vector**.
- Line 58 corresponds to **Number of modes**.
- Line 59 corresponds to **Number of zero crossings**.
- Line 60 corresponds to **Direct feedthrough** and **Time dependence**.

Line 62 defines the initial values that will be displayed in the block's configuration dialog.

Lines 63-70 define the Scilab code that will be executed to draw the block's graphical and text elements. The code must be captured as a list of strings. Be careful about quoting!

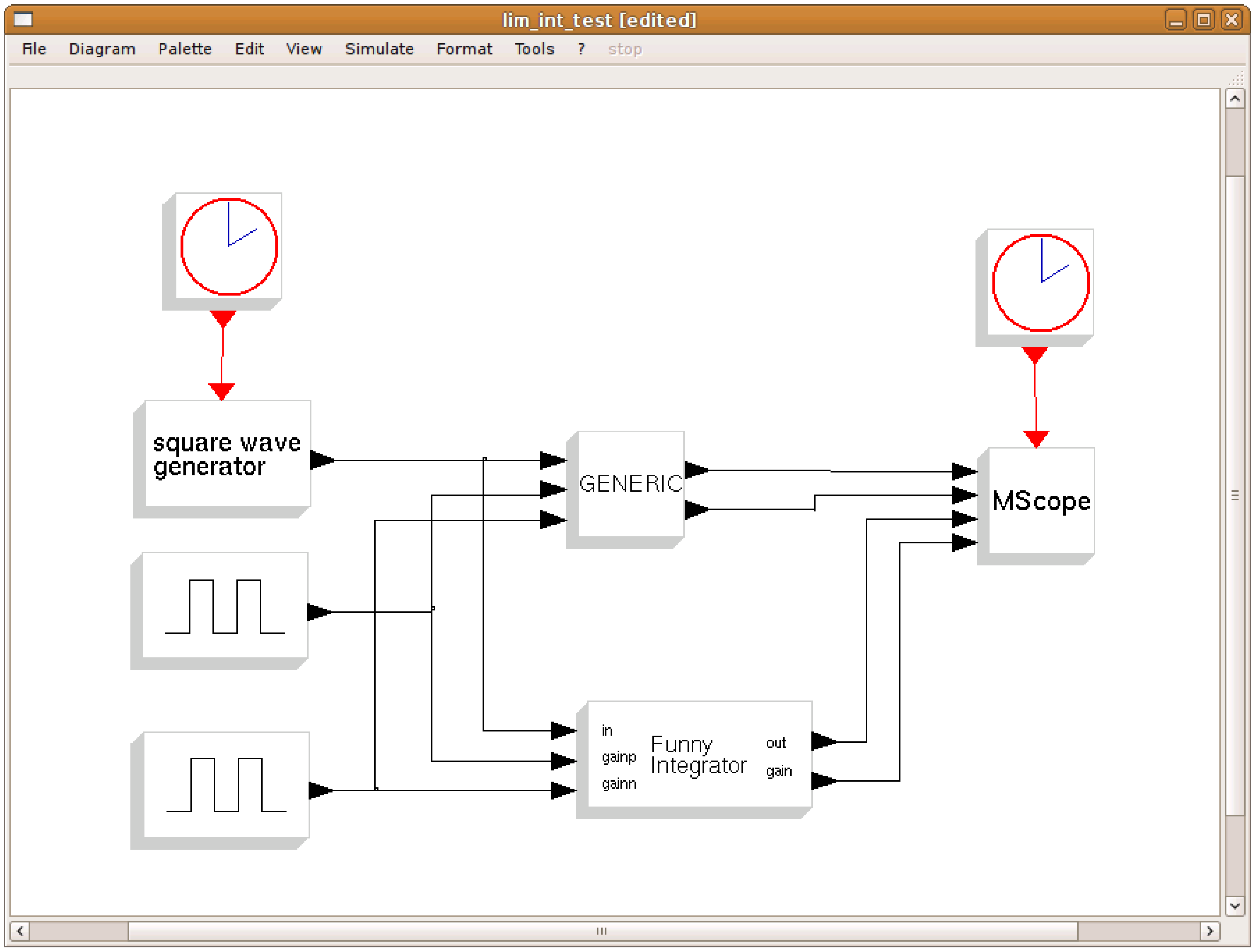
Line 71 creates the block, including both its graphical and model elements. The size of the block is specified by the first parameter of the standard_define() function.

7 Testing the Interfacing Function

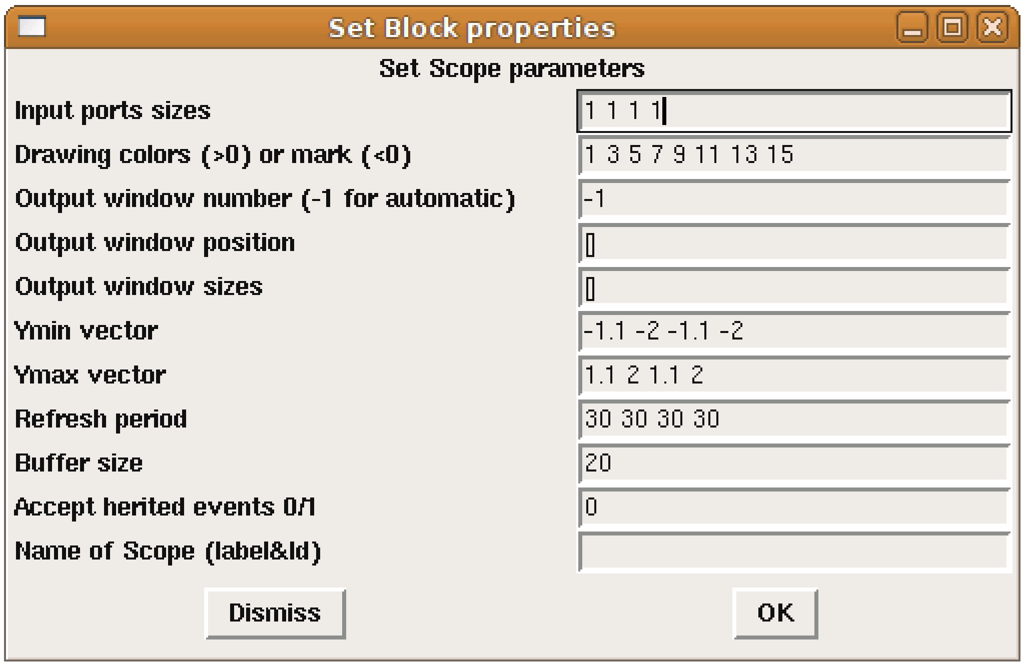
Now that we have written the interfacing function, it is a simple matter to use it. Assuming you still have Scicos open, activate the Scilab window (from the Scicos Tools menu) and enter the following command to create the interfacing function in Scilab:

exec lim_int_intf.sci;

Go back to Scicos by clicking the mouse in the Scicos window, open the dialog from the **Edit | Add new block** menu, and enter LIMINT into the dialog. Then simply place the new block into the diagram, and connect it up. The picture below shows what this looks like:



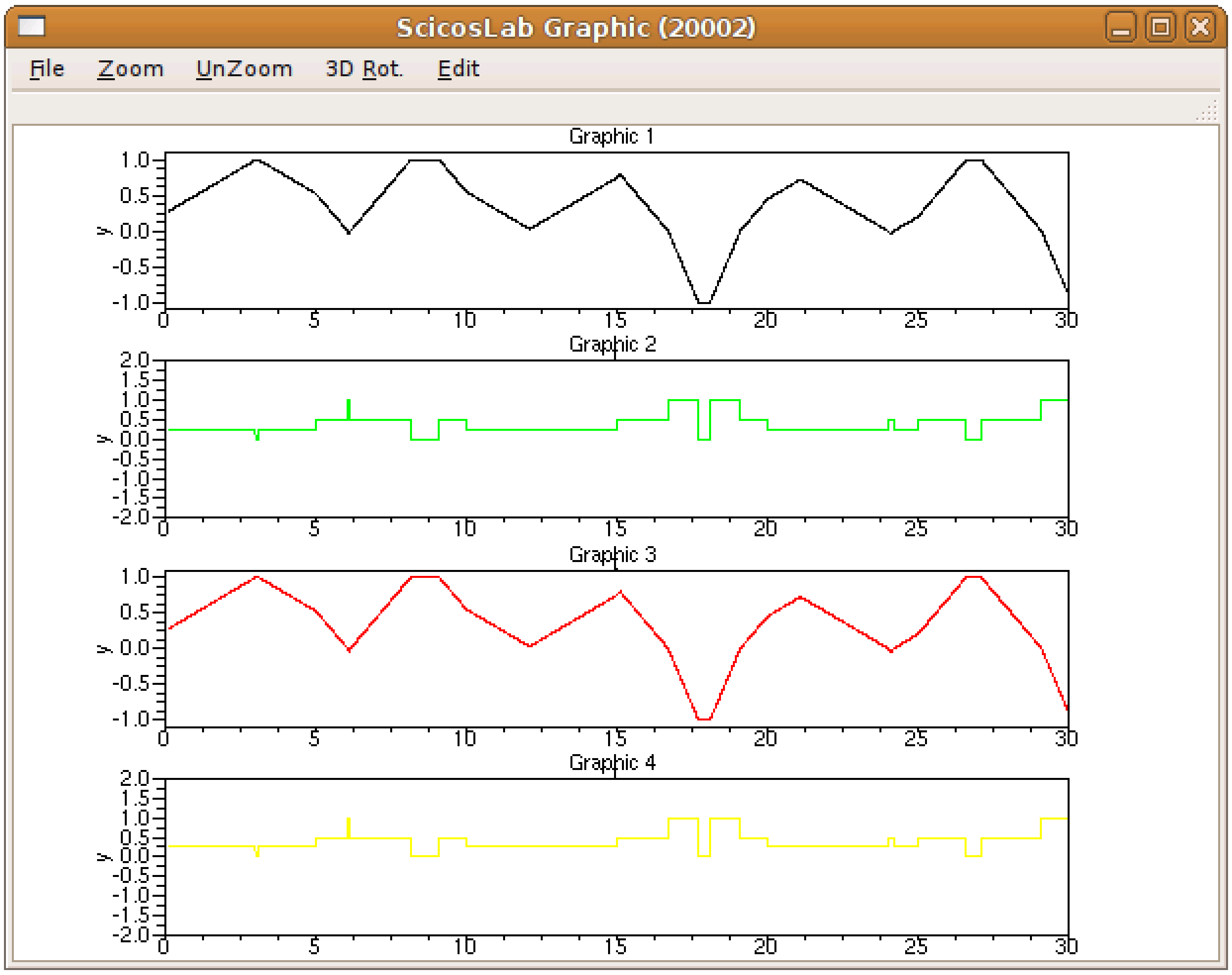
Here's the configuration dialog for the CMSCOPE block:



Set Scope parameters

Input ports sizes	1 1 1 1
Drawing colors (>0) or mark (<0)	1 3 5 7 9 11 13 15
Output window number (-1 for automatic)	-1
Output window position	[]
Output window sizes	[]
Ymin vector	-1.1 -2 -1.1 -2
Ymax vector	1.1 2 1.1 2
Refresh period	30 30 30 30
Buffer size	20
Accept herited events 0/1	0
Name of Scope (label&Id)	

Once again, run the model. You should get the following plot:



As expected, the outputs of the two blocks are the same because they use the same computational function. Cool!

8 Debugging

All the above is well and good, however, when you’re writing your own functions, it is likely that you will have errors, and will need to debug them. I showed in the interfacing function how to drop in simple debugging lines, so I won’t discuss that any further. However, for C code, sprinkling printf() calls into the code may not be sufficient or may not work at all, especially if your Scilab instance is running as a separate process, as stdout may not go to a console. So, you will need to use gdb. In this section, I’ll give you a (very) high level overview of the process. And remember, what I describe here is what I do on my Ubuntu system. It will likely work the same on most *nix systems, but is liable not to work the same on Windows systems.

8.1 First Steps

In order to make your life much easier, we will start by making a macro file for gdb that will display Scicos model values. Create a file in your working directory called gdb_cmd, and put the following content into it:

```
define showInputs
    printf "Inputs:  "
    set $i = 0
    while $i < $arg0 -> nin
        printf "in%i = %g   ", $i, *(double*)$arg0 -> inptr[$i]
        set $i = $i + 1
    end
    printf "\n"
end

define showOutputs
    printf "Outputs: "
    set $i = 0
    while $i < $arg0 -> nout
        printf "out%i = %g   ", $i, *(double*)$arg0 -> outptr[$i]
        set $i = $i + 1
    end
    printf "\n"
end

define showParms
    printf "rParms:  "
    set $i = 0
    while $i < $arg0 -> nrpar
        printf "rp%i = %g   ", $i, $arg0 -> rpar[$i]
        set $i = $i + 1
    end
    printf "\n"
    printf "iParms:  "
    set $i = 0
    while $i < $arg0 -> nipar
        printf "ip%i = %i   ", $i, $arg0 -> ipar[$i]
        set $i = $i + 1
    end
    printf "\n"
end

define showState
    printf "State:   "
    set $i = 0
    while $i < $arg0 -> nx
        printf "x%i = %g   ", $i, $arg0 -> x[$i]
        set $i = $i + 1
    end
    printf "\n"
    printf "Deriv:   "
    set $i = 0
    while $i < $arg0 -> nx
        printf "xd%i = %i   ", $i, $arg0 -> xd[$i]
        set $i = $i + 1
    end
    printf "\n"
end

define showMZ
    printf "zCross:  "
    set $i = 0
    while $i < $arg0 -> ng
        printf "g%i = %g   ", $i, $arg0 -> g[$i]
        set $i = $i + 1
    end
    printf "\n"
    printf "jRoot:   "
    set $i = 0
    while $i < $arg0 -> ng
        printf "j%i = %g   ", $i, $arg0 -> jroot[$i]
        set $i = $i + 1
    end
    printf "\n"
    printf "Mode:    "
    set $i = 0
    while $i < $arg0 -> nmode
        printf "m%i = %i   ", $i, $arg0 -> mode[$i]
        set $i = $i + 1
    end
    printf "\n"
end

define Show
    showInputs(blk)
    showOutputs(blk)
    showParms(blk)
    showMZ(blk)
    showState(blk)
end

define hook-stop
    Show
end

break lim_int
continue
```

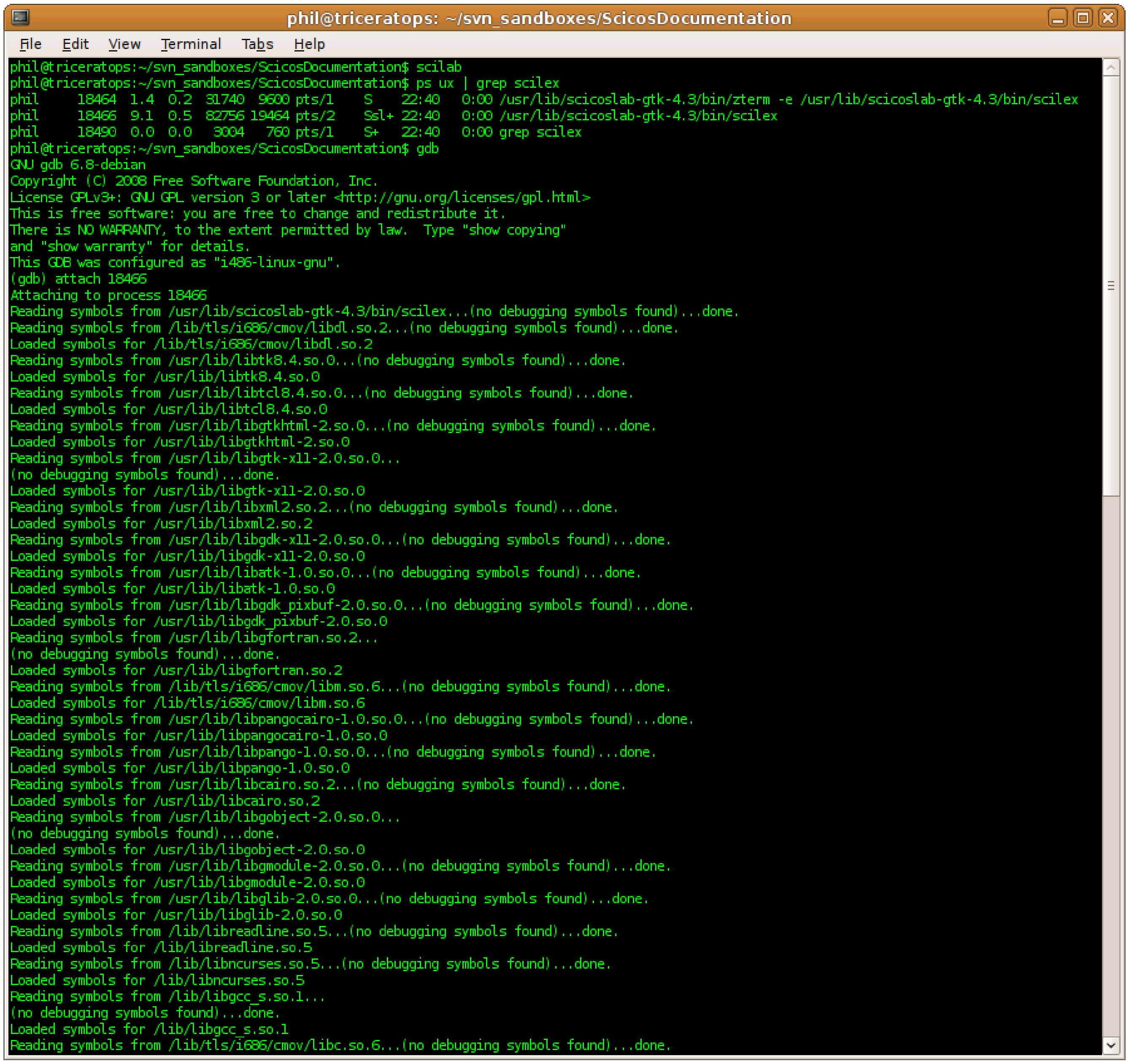
Just a quick comment on the script above: Notice that the *Show* function passes the name ***blk*** to the various *showXXX* functions; ***blk*** is the very same ***blk*** parameter in the *lim_int()* C computational function.

8.2 Starting a Debugging Session

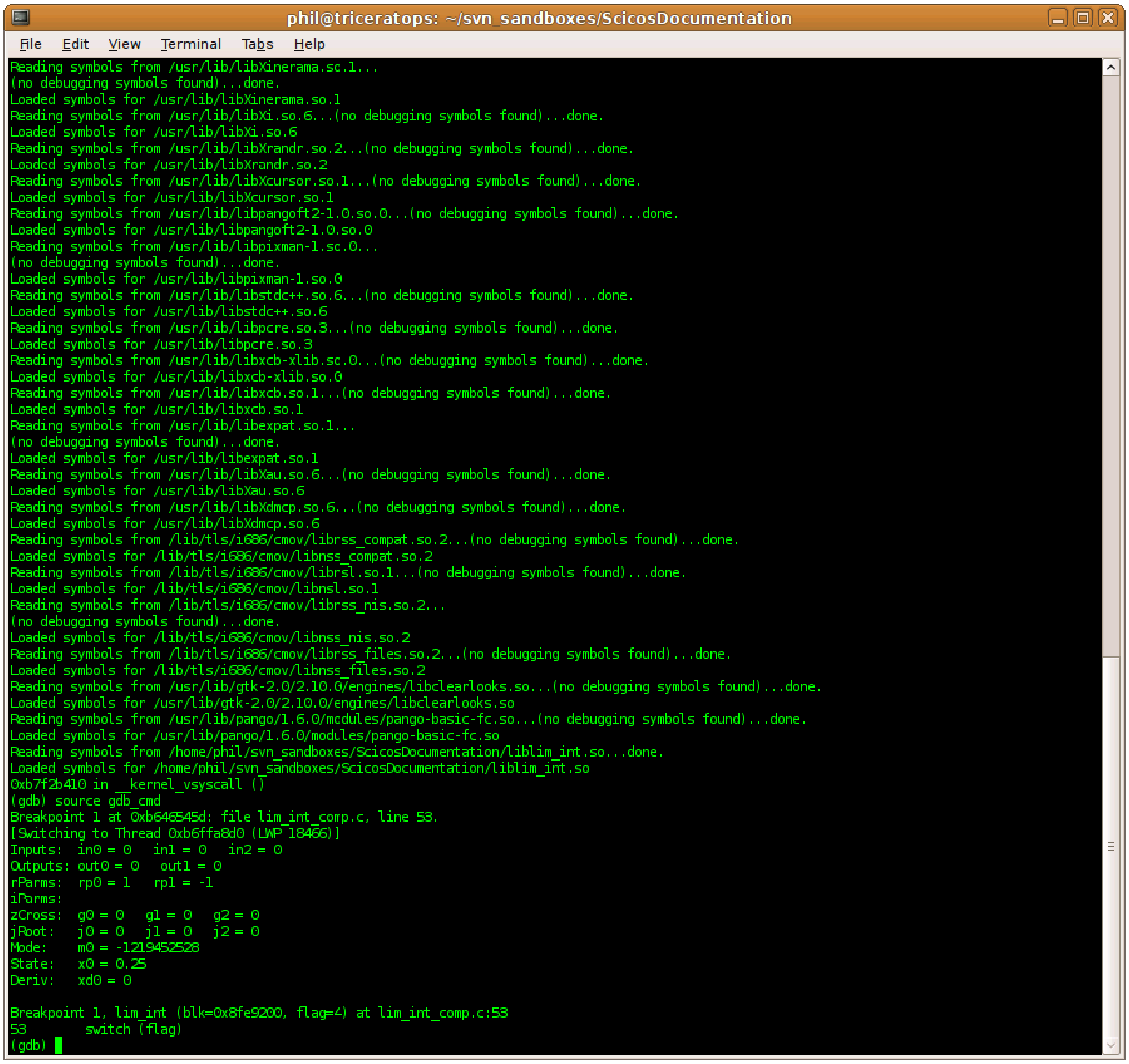
Using the example block we created in this tutorial, follow these steps to debug the model:

1. Start Scilab as you did earlier. Once in Scilab, start Scicos, and load the example model file lim_int_test.
2. In the console, enter the command **ps ux | grep scilex**. Scilex is the name of the Scilab executable.
3. Run gdb.
4. Attach gdb to the scilex process.
5. Source the gdb_cmd file into gdb.
6. Now go to Scicos and run the model.

At this point, Scicos has been stopped at a breakpoint at the computational function lim_int(). This whole interaction will look like the following screenshots:


phil@triceratops: ~/svn_sandboxes/ScicosDocumentation
File Edit View Terminal Tabs Help

```
phil@triceratops:~/svn_sandboxes/ScicosDocumentation$ scilab
phil@triceratops:~/svn_sandboxes/ScicosDocumentation$ ps ux | grep scilex
phil     18464  1.4  0.2  31740  9600 pts/1    S    22:40   0:00 /usr/lib/scicoslab-gtk-4.3/bin/zterm -e /usr/lib/scicoslab-gtk-4.3/bin/scilex
phil     18466  9.1  0.5  82756 19464 pts/2    Ssl+ 22:40   0:00 /usr/lib/scicoslab-gtk-4.3/bin/scilex
phil     18490  0.0  0.0   3004   760 pts/1    S+   22:40   0:00 grep scilex
phil@triceratops:~/svn_sandboxes/ScicosDocumentation$ gdb
GNU gdb 6.8-debian
Copyright (C) 2008 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "i486-linux-gnu".
(gdb) attach 18466
Attaching to process 18466
Reading symbols from /usr/lib/scicoslab-gtk-4.3/bin/scilex...(no debugging symbols found)...done.
Reading symbols from /lib/tls/i686/cmov/libdl.so.2...(no debugging symbols found)...done.
Loaded symbols for /lib/tls/i686/cmov/libdl.so.2
Reading symbols from /usr/lib/libtk8.4.so.0...(no debugging symbols found)...done.
Loaded symbols for /usr/lib/libtk8.4.so.0
Reading symbols from /usr/lib/libtcl8.4.so.0...(no debugging symbols found)...done.
Loaded symbols for /usr/lib/libtcl8.4.so.0
Reading symbols from /usr/lib/libgtkhtml-2.so.0...(no debugging symbols found)...done.
Loaded symbols for /usr/lib/libgtkhtml-2.so.0
Reading symbols from /usr/lib/libgtk-x11-2.0.so.0...
(no debugging symbols found)...done.
Loaded symbols for /usr/lib/libgtk-x11-2.0.so.0
Reading symbols from /usr/lib/libxml2.so.2...(no debugging symbols found)...done.
Loaded symbols for /usr/lib/libxml2.so.2
Reading symbols from /usr/lib/libgdk-x11-2.0.so.0...(no debugging symbols found)...done.
Loaded symbols for /usr/lib/libgdk-x11-2.0.so.0
Reading symbols from /usr/lib/libatk-1.0.so.0...(no debugging symbols found)...done.
Loaded symbols for /usr/lib/libatk-1.0.so.0
Reading symbols from /usr/lib/libgdk_pixbuf-2.0.so.0...(no debugging symbols found)...done.
Loaded symbols for /usr/lib/libgdk_pixbuf-2.0.so.0
Reading symbols from /usr/lib/libgfortran.so.2...
(no debugging symbols found)...done.
Loaded symbols for /usr/lib/libgfortran.so.2
Reading symbols from /lib/tls/i686/cmov/libm.so.6...(no debugging symbols found)...done.
Loaded symbols for /lib/tls/i686/cmov/libm.so.6
Reading symbols from /usr/lib/libpangocairo-1.0.so.0...(no debugging symbols found)...done.
Loaded symbols for /usr/lib/libpangocairo-1.0.so.0
Reading symbols from /usr/lib/libpango-1.0.so.0...(no debugging symbols found)...done.
Loaded symbols for /usr/lib/libpango-1.0.so.0
Reading symbols from /usr/lib/libcairo.so.2...(no debugging symbols found)...done.
Loaded symbols for /usr/lib/libcairo.so.2
Reading symbols from /usr/lib/libgobject-2.0.so.0...
(no debugging symbols found)...done.
Loaded symbols for /usr/lib/libgobject-2.0.so.0
Reading symbols from /usr/lib/libgmodule-2.0.so.0...(no debugging symbols found)...done.
Loaded symbols for /usr/lib/libgmodule-2.0.so.0
Reading symbols from /usr/lib/libglib-2.0.so.0...(no debugging symbols found)...done.
Loaded symbols for /usr/lib/libglib-2.0.so.0
Reading symbols from /lib/libreadline.so.5...(no debugging symbols found)...done.
Loaded symbols for /lib/libreadline.so.5
Reading symbols from /lib/libncurses.so.5...(no debugging symbols found)...done.
Loaded symbols for /lib/libncurses.so.5
Reading symbols from /lib/libgcc_s.so.1...
(no debugging symbols found)...done.
Loaded symbols for /lib/libgcc_s.so.1
Reading symbols from /lib/tls/i686/cmov/libc.so.6...(no debugging symbols found)...done.
```


phil@triceratops: ~/svn_sandboxes/ScicosDocumentation
File Edit View Terminal Tabs Help

```
Reading symbols from /usr/lib/libXinerama.so.1...
(no debugging symbols found)...done.
Loaded symbols for /usr/lib/libXinerama.so.1
Reading symbols from /usr/lib/libXi.so.6...(no debugging symbols found)...done.
Loaded symbols for /usr/lib/libXi.so.6
Reading symbols from /usr/lib/libXrandr.so.2...(no debugging symbols found)...done.
Loaded symbols for /usr/lib/libXrandr.so.2
Reading symbols from /usr/lib/libXcursor.so.1...(no debugging symbols found)...done.
Loaded symbols for /usr/lib/libXcursor.so.1
Reading symbols from /usr/lib/libpangoft2-1.0.so.0...(no debugging symbols found)...done.
Loaded symbols for /usr/lib/libpangoft2-1.0.so.0
Reading symbols from /usr/lib/libpixman-1.so.0...
(no debugging symbols found)...done.
Loaded symbols for /usr/lib/libpixman-1.so.0
Reading symbols from /usr/lib/libstdc++.so.6...(no debugging symbols found)...done.
Loaded symbols for /usr/lib/libstdc++.so.6
Reading symbols from /usr/lib/libpcre.so.3...(no debugging symbols found)...done.
Loaded symbols for /usr/lib/libpcre.so.3
Reading symbols from /usr/lib/libxcb-xlib.so.0...(no debugging symbols found)...done.
Loaded symbols for /usr/lib/libxcb-xlib.so.0
Reading symbols from /usr/lib/libxcb.so.1...(no debugging symbols found)...done.
Loaded symbols for /usr/lib/libxcb.so.1
Reading symbols from /usr/lib/libexpat.so.1...
(no debugging symbols found)...done.
Loaded symbols for /usr/lib/libexpat.so.1
Reading symbols from /usr/lib/libXau.so.6...(no debugging symbols found)...done.
Loaded symbols for /usr/lib/libXau.so.6
Reading symbols from /usr/lib/libXdmcp.so.6...(no debugging symbols found)...done.
Loaded symbols for /usr/lib/libXdmcp.so.6
Reading symbols from /lib/tls/i686/cmov/libnss_compat.so.2...(no debugging symbols found)...done.
Loaded symbols for /lib/tls/i686/cmov/libnss_compat.so.2
Reading symbols from /lib/tls/i686/cmov/libnsl.so.1...(no debugging symbols found)...done.
Loaded symbols for /lib/tls/i686/cmov/libnsl.so.1
Reading symbols from /lib/tls/i686/cmov/libnss_nis.so.2...
(no debugging symbols found)...done.
Loaded symbols for /lib/tls/i686/cmov/libnss_nis.so.2
Reading symbols from /lib/tls/i686/cmov/libnss_files.so.2...(no debugging symbols found)...done.
Loaded symbols for /lib/tls/i686/cmov/libnss_files.so.2
Reading symbols from /usr/lib/gtk-2.0/2.10.0/engines/libclearlooks.so...(no debugging symbols found)...done.
Loaded symbols for /usr/lib/gtk-2.0/2.10.0/engines/libclearlooks.so
Reading symbols from /usr/lib/pango/1.6.0/modules/pango-basic-fc.so...(no debugging symbols found)...done.
Loaded symbols for /usr/lib/pango/1.6.0/modules/pango-basic-fc.so
Reading symbols from /home/phil/svn_sandboxes/ScicosDocumentation/liblim_int.so...done.
Loaded symbols for /home/phil/svn_sandboxes/ScicosDocumentation/liblim_int.so
0xb7f2b410 in __kernel_vsyscall ()
(gdb) source gdb_cmd
Breakpoint 1 at 0xb646545d: file lim_int_comp.c, line 53.
[Switching to Thread 0xb6ffa8d0 (LWP 18466)]
Inputs:  in0 = 0   in1 = 0   in2 = 0
Outputs: out0 = 0   out1 = 0
rParms:  rp0 = 1   rp1 = -1
iParms:
zCross:  g0 = 0   g1 = 0   g2 = 0
jRoot:   j0 = 0   j1 = 0   j2 = 0
Mode:    m0 = -1219452528
State:   x0 = 0.25
Deriv:   xd0 = 0

Breakpoint 1, lim_int (blk=0x8fe9200, flag=4) at lim_int_comp.c:53
53        switch (flag)
(gdb) 
```

Notice in the above screenshot that the values of the block data structure elements are displayed. They will appear each time that gdb breaks program execution.

From here, you will use the standard gdb commands to inspect code and step through it. The **help**, **step** and **continue** commands will be very useful. For further details on gdb debugging, use its online help or consult the copious documentation that is available on the web.

9 Conclusion

This concludes this tutorial. I hope it has been helpful!