

OPEN SOURCE ENGINEERING



A SCILAB PROFESSIONAL PARTNER



HOW TO DEVELOP A GRAPHICAL USER INTERFACE (GUI) IN SCILAB.

In this tutorial we show how to create a GUI in Scilab for an ODE problem. The adopted problem is the LHY model already used in other tutorials.

Level



This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License.

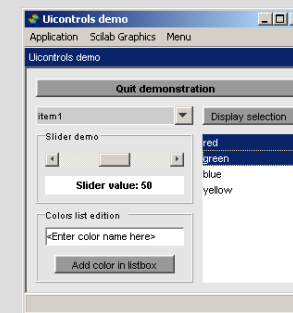
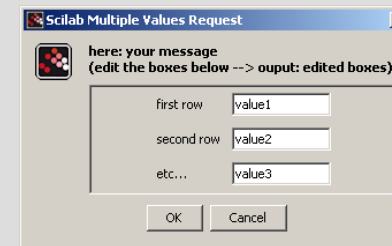
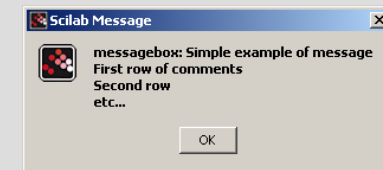


www.openeering.com

Step 1: The purpose of this tutorial

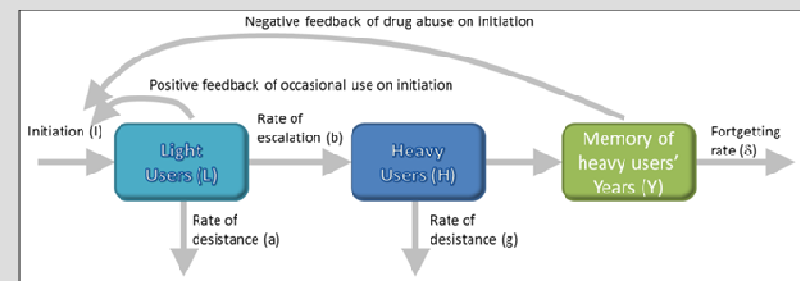
In this tutorial we show, step by step, how to create a Graphical User Interface (*GUI*) in Scilab.

Examples of Scilab GUIs are reported on the right. These images are taken from the GUI menu of the *Scilab Demonstrations*.



Step 2: Model description

In this tutorial we refer to the LHY model. A description of the *LHY model* and its solution using *ode* command can be found in the basic tutorial of modeling differential equations in Scilab.



Step 3: Roadmap

In this tutorial we describe how to construct a GUI for the LHY. We implement the system in the following way:

- Provide a description of GUI programming in Scilab;
- Implement the GUI for the LHY model;
- Test the program and visualize the results.

Descriptions	Steps
GUI programming in Scilab	4-5
GUI for the LHY model	6-17
Running and testing	18-19

Step 4: Scilab GUI programming

A GUI is a *human-computer interface* that uses graphical objects like windows, menus and icons to interact with users through the use of mouse and keyboard (often in a limited way).

The main advantage of using GUIs is the ability to make computer operations more intuitive, and thus easier to learn and use especially for new users. For example, it is much easier to move files by dragging icons with the mouse than by having to remember the command that performs the same task.

The function that creates a graphical user interface object in Scilab is "*uicontrol*".

The syntax of the GUI function is

```
h = uicontrol(PropertyName,PropertyValue,...)
h = uicontrol(parent,PropertyName,PropertyValue,...)
h = uicontrol(uich)
```

For a detailed description see "help uicontrol".

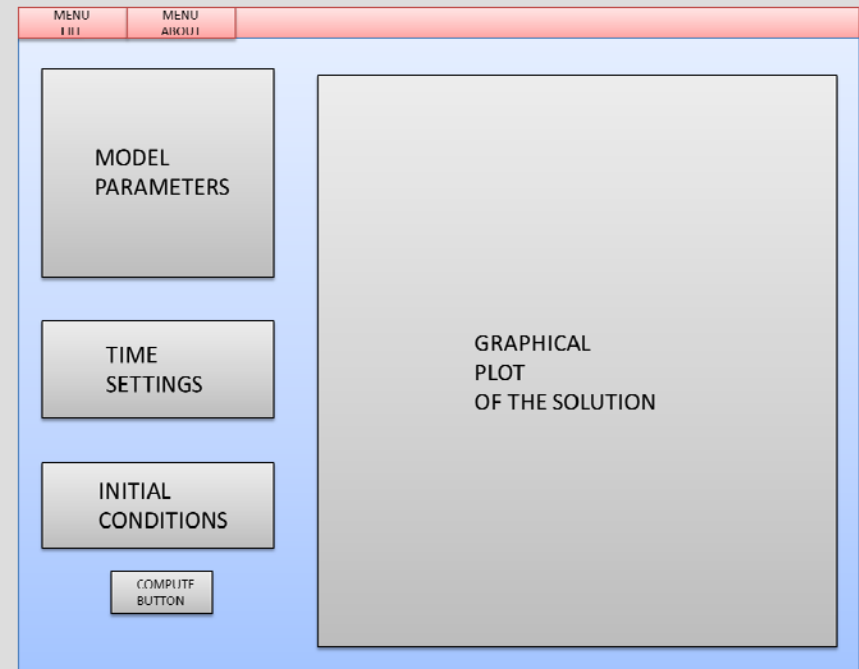
Step 5: Design of the GUI interface

The first step is to *design* the skeleton of *the GUI interface* for the LHY model.

Designing a “correct” GUI is not a trivial task. Many rules exist and can be found in textbooks and web sites dedicated to this subject.

On the right we report our scheme for the LHY graphical user interface where:

- "Model parameters" contains the list of all LHY model parameters that the user can change;
- "Time Setting" contains the list of all time simulation variables;
- "Initial conditions" contains the list of all initial conditions for the ODE problem;
- "graphical plot " contains the plot of the simulation's results;
- "Compute button" updates the solution in the "graphical plot ";
- The menu "file" contains the entry "close" that is used to close the program;
- The menu "About" contains the entry "About" that is used to display some info about program and author.



Step 6: Storing default parameters

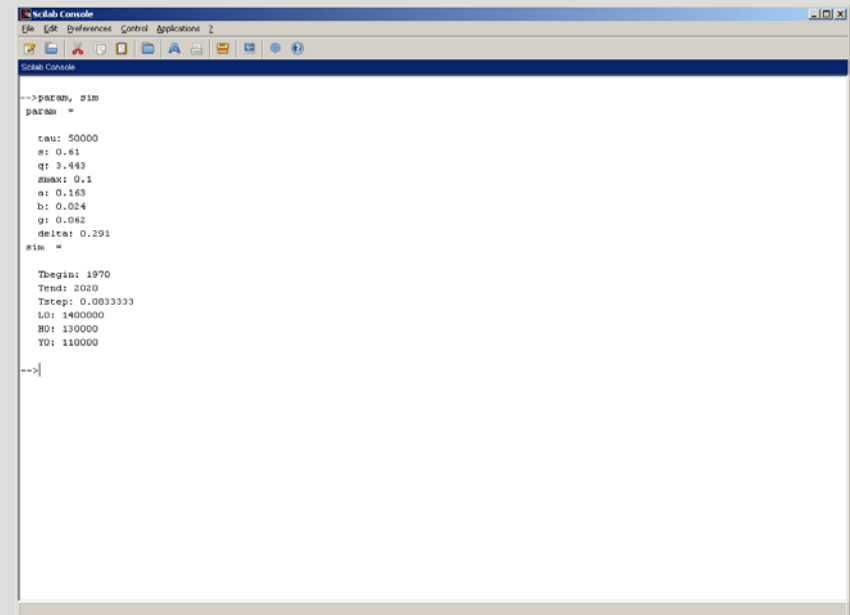
The first step is to store all the default LHY model parameters into `mlist` Scilab objects. The parameters are subdivided into two groups:

- the "param" group for the LHY model parameters;
- the "sim" group for the simulation time parameters and initial conditions.

```
// Default simulation parameters
param = [];
param.tau = 5e4; // Number of innovators per year (initiation)
param.s = 0.61; // Annual rate at which light users attract non-
users (initiation)
param.q = 3.443; // Constant which measures the deterrent effect
of heavy users (initiation)
param.smax = 0.1; // Lower bound for s effective (initiation)
param.a = 0.163; // Annual rate at which light users quit
param.b = 0.024; // Annual rate at which light users escalate to
heavy use
param.g = 0.062; // Annual rate at which heavy users quit
param.delta = 0.291; // Forgetting rate

// Simulation data
// Time variables
sim = [];
sim.Tbegin = 1970; // Initial time
sim.Tend = 2020; // Final time
sim.Tstep = 1/12; // Time step
// Initial conditions
sim.L0 = 1.4e6; // Light users at the initial time
sim.H0 = 0.13e6; // Heavy users at the initial time
sim.Y0 = 0.11e6; // Decaying heavy user at the initial time
```

The name of the developed function is "LHY_Gui".



```
Scilab Console
-->param, sim
param =
tau: 50000
s: 0.61
q: 3.443
smax: 0.1
a: 0.163
b: 0.024
g: 0.062
delta: 0.291
sim =
Tbegin: 1970
Tend: 2020
Tstep: 0.08333333
L0: 1400000
H0: 130000
Y0: 110000
-->
```

Step 7: Create an initial window

In the next step we create, with the "scf" Scilab function, an initial window where graphical object can be added.

The function "gettext" is used for internationalization (a.k.a. i18n, i.e. it translates the text into the current local language if available).

```
// Global window parameters
global margin_x margin_y;
global frame_w frame_h plot_w plot_h;

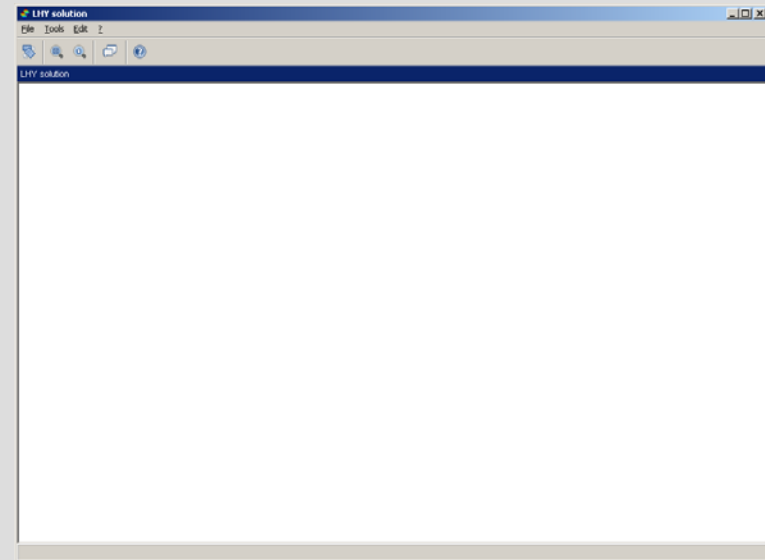
// Window Parameters initialization
frame_w = 300; frame_h = 550; // Frame width and height
plot_w = 600; plot_h = frame_h; // Plot width and height
margin_x = 15; margin_y = 15; // Horizontal and vertical margin
for elements
defaultfont = "arial"; // Default Font
axes_w = 3*margin_x + frame_w + plot_w; // axes width
axes_h = 2*margin_y + frame_h; // axes height (100 =>
toolbar height)

demo_lhy = scf(100001); // Create window with id=100001 and make
it the current one

// Background and text
demo_lhy.background = -2;
demo_lhy.figure_position = [100 100];
demo_lhy.figure_name = gettext("LHY solution");

// Change dimensions of the figure
demo_lhy.axes_size = [axes_w axes_h];
```

Type "help scf" for more details on the scf command.



Step 8: Set menu and toolbar

After creating the window, we create the menu. The used function are:

- "addmenu"/"delmenu": for adding or removing a menu from the menu bar;
- "uimenu": for creating a menu or a submenu in a figure.

Associate to each menu entry there is a **callback**, a reference to a executable function that is activated when the user select the entry.

```
// Remove Scilab graphics menus & toolbar
delmenu(demo_lhy.figure_id,gettext("&File"));
delmenu(demo_lhy.figure_id,gettext("&Tools"));
delmenu(demo_lhy.figure_id,gettext("&Edit"));
delmenu(demo_lhy.figure_id,gettext("&?"));
toolbar(demo_lhy.figure_id,"off");

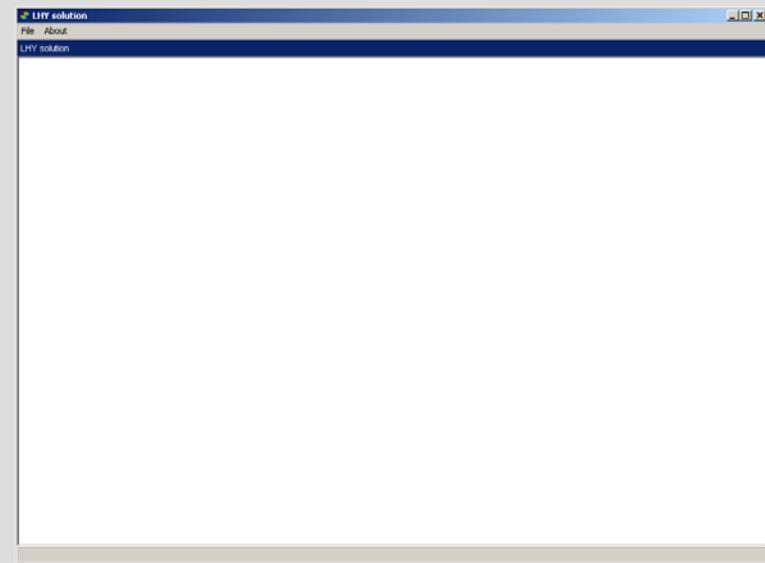
// New menu
h1 = uimenu("parent",demo_lhy, "label",gettext("File"));
h2 = uimenu("parent",demo_lhy, "label",gettext("About"));

// Populate menu: file
uimenu("parent",h1, "label",gettext("Close"), "callback",
"demo_lhy=get_figure_handle(100001);delete(demo_lhy);",
"tag","close_menu");

// Populate menu: about
uimenu("parent",h2, "label",gettext("About"),
"callback","LHY_About();");
// Sleep to guarantee a better display (avoiding to see a
sequential display)
sleep(500);
```

An import field for the graphical function is the **"parent"** field i.e. the handle where the graphical object should be attached, in our case "demo_lhy". Changing this value allows to move a control from a figure to another.

See "uimenu" help file for more details.

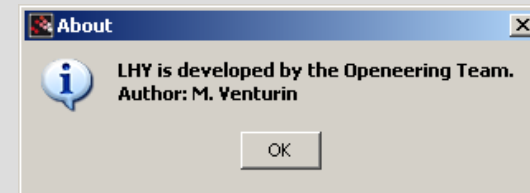


Step 9: About dialog

The "About" dialog is a modal dialog created using the "messagebox" function. Two types of windows are available:

- **"modal dialog"**: in a modal window users are required to interact with it before it returns to the parent application.
- **"modeless dialog"**: in modeless or non-modal window the request information is not essential to continue with the program. A non-modal window can be left open while work continues elsewhere.

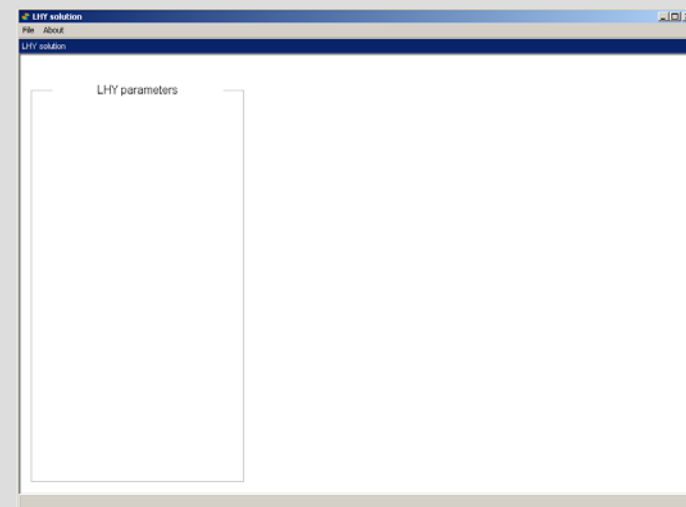
```
function LHY_About()  
    msg = sprintf(gettext("LHY is developed by the Openeering  
Team.\nAuthor: M. Venturin"));  
    messagebox(msg, gettext("About"), "info", "modal");  
endfunction
```



Step 10: Creating a frame

A frame is graphical object used to group other objects.

```
// Frames creation [LHY parameters]  
my_frame = uicontrol("parent",demo_lhy, "relief","groove", ...  
    "style","frame", "units","pixels", ...  
    "position",[ margin_x margin_y frame_w frame_h], ...  
    "horizontalalignment","center", "background",[1 1 1], ...  
    "tag","frame_control");  
  
// Frame title  
my_frame_title = uicontrol("parent",demo_lhy, "style","text", ...  
    "string","LHY parameters", "units","pixels", ...  
    "position",[30+margin_x margin_y+frame_h-10 frame_w-60 20],  
    ...  
    "fontname",defaultfont, "fontunits","points", ...  
    "fontsize",16, "horizontalalignment","center", ...  
    "background",[1 1 1], "tag","title_frame_control");
```



Step 11: Populate the frame

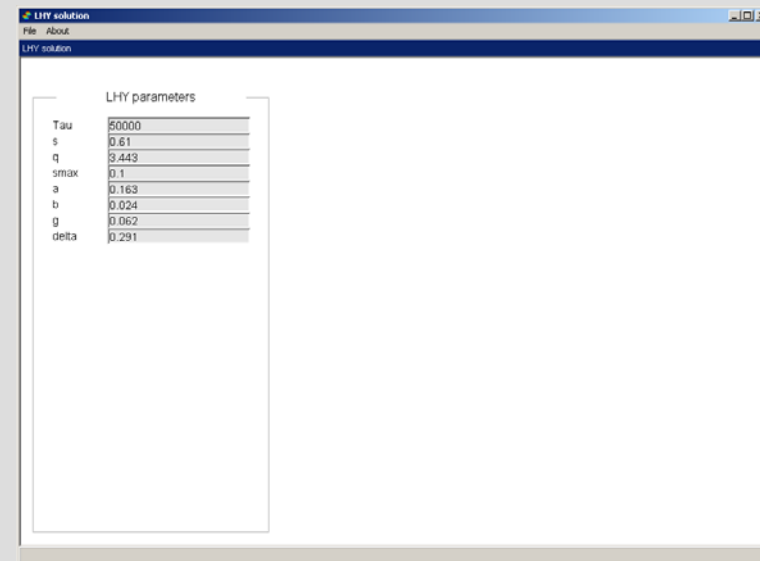
Each parameters is composed of the following elements:

- a string that denote the parameter (style "text");
- an edit box (style "edit") that is used as an input/output mask (the input is recovering by searching all graphical objects having that tag, and hence the tag must be unique among the elements)

```
// Adding model parameters
guihl = frame_w;
guihlo = 240;

// ordered list of labels
labels1 = ["Tau", "s", "q", "smax", "a", "b", "g", "delta"];
// ordered list of default values
values1 = [5e4, 0.61, 3.443, 0.1, 0.163, 0.024, 0.062, 0.291];
// positioning
l1 = 40; l2 = 100; l3 = 110;
for k=1:size(labels1,2)
    uicontrol("parent",demo_lhy, "style","text",...
        "string",labels1(k), "position",[l1,guihl-
k*20+guihlo,l2,20], ...
        "horizontalalignment","left", "fontsize",14, ...
        "background",[1 1 1]);

    guientry1(k) = uicontrol("parent",demo_lhy, "style","edit",
    ...
        "string",string(values1(k)), "position",[l3,guihl-
k*20+guihlo,l80,20], ...
        "horizontalalignment","left", "fontsize",14, ...
        "background",[.9 .9 .9], "tag",labels1(k));
end
```



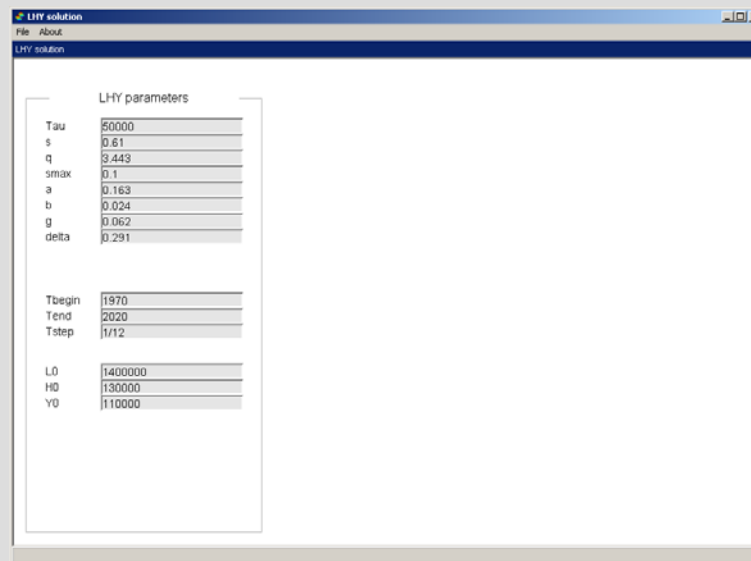
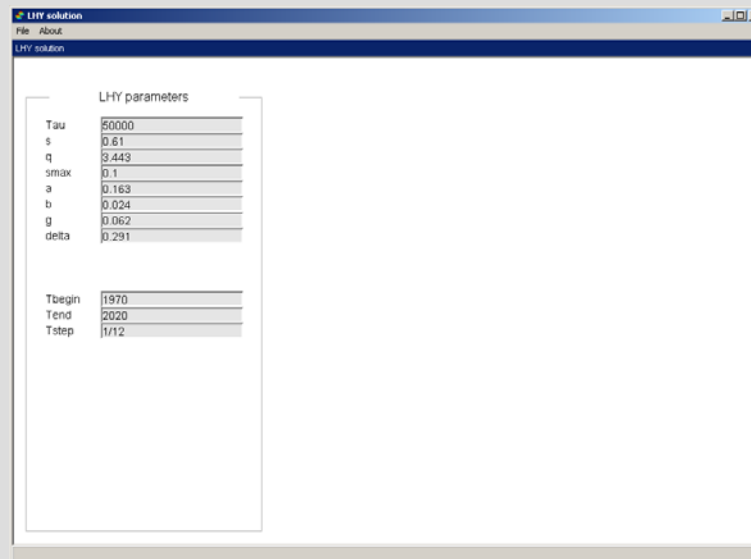
Step 12: Continue with the frame

As previously done, we continue to add graphical components for simulations parameters and initial conditions.

```
// Adding simulation parameters
guih2 = 240;
guih2o = 80;
labels2 = ["Tbegin", "Tend", "Tstep"];
values2 = ["1970", "2020", "1/12"];
l1 = 40; l2 = 100; l3 = 110;
for k=1:size(labels2,2)
    uicontrol("parent",demo_lhy, "style","text", ...
        "string",labels2(k), "position",[l1,guih2-
k*20+guih2o,l2,20], ...
        "horizontalalignment","left", "fontsize",14, ...
        "background",[1 1 1]);

    guientry2(k) = uicontrol("parent",demo_lhy, "style","edit",
    ...
        "string",values2(k), "position",[l3,guih2-
k*20+guih2o,l80,20], ...
        "horizontalalignment","left", "fontsize",14, ...
        "background",[.9 .9 .9], "tag",labels2(k));
end
// Adding initial conditions
guih3 = 150;
guih3o = 80;
labels3 = ["L0", "H0", "Y0"];
values3 = [1.4e6, 0.13e6, 0.11e6];
l1 = 40; l2 = 100; l3 = 110;
for k=1:size(labels3,2)
    uicontrol("parent",demo_lhy, "style","text", ...
        "string",labels3(k), "position",[l1,guih3-
k*20+guih3o,l2,20], ...
        "horizontalalignment","left", "fontsize",14, ...
        "background",[1 1 1]);

    guientry3(k) = uicontrol("parent",demo_lhy, "style","edit",
    ...
        "string",string(values3(k)), "position",[l3,guih3-
k*20+guih3o,l80,20], ...
        "horizontalalignment","left", "fontsize",14,...
        "background",[.9 .9 .9], "tag",labels3(k));
end
```

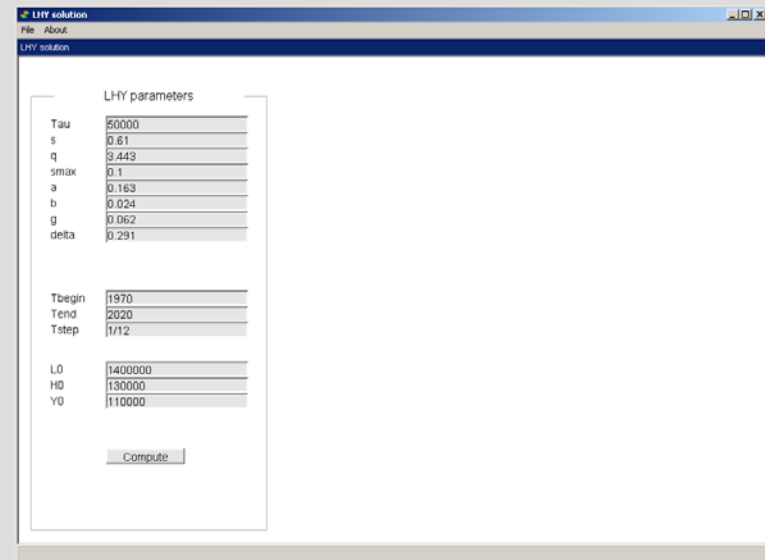


Step 13: Add a button

Next, using the same command "uicontrol", we add the "compute button" (style "pushbutton"):

```
// Adding button
huibutton = uicontrol(demo_lhy, "style","pushbutton", ...
    "Position",[110 100 100 20], "String","Compute", ...
    "BackgroundColor",[.9 .9 .9], "fontsize",14, ...
    "Callback","syscompute");
```

The button is associated to the callback "syscompute". This callback will be explained and created at step 17.



Step 14: Solving function

The solving function is based on the code already developed in the tutorial of the LHY model solved using "ode" function. See this tutorial for more details.

```
function sol=solvingProblem(param, sim)
    // Compute the solution
    sol = struct();

    // Assign ODE solver data
    y0 = [sim.L0;sim.H0;sim.Y0];
    t0 = sim.Tbegin;
    t = sim.Tbegin:sim.Tstep:(sim.Tend+100*%eps);
    f = LHY_System;

    // Solving the system
    sol.LHY = ode(y0, t0, t, f);
    sol.t = t;
endfunction
```

Step 15: Display function

The display function is derived from the "LHY_Plot" function of the tutorial on LHY solved using "ode" function. See this tutorial for more details.

```
function displayProblem(solh)
// Fetching solution
LHY = sol.LHY;
t = sol.t;
L = LHY(1,:); H = LHY(2,:); Y = LHY(3,:);

// Evaluate initiation
I = LHY_Initiation(L,H,Y, param);

// maximum values for nice plot
[Lmax, Lindmax] = max(L); tL = t(Lindmax);
[Hmax, Hindmax] = max(H); tH = t(Hindmax);
[Ymax, Yindmax] = max(Y); tY = t(Yindmax);
[Imax, Iindmax] = max(I); tI = t(Iindmax);

// Text of the maximum point
Ltext = sprintf(' (%4.1f , %7.0f)', tL, Lmax);
Htext = sprintf(' (%4.1f , %7.0f)', tH, Hmax);
Ytext = sprintf(' (%4.1f , %7.0f)', tY, Ymax);
Itext = sprintf(' (%4.1f , %7.0f)', tI, Imax);

// Plotting of model data
delete(gca());
plot(t,[LHY;I]);
legend(['Light Users'; 'Heavy users'; 'Memory'; 'Initiation']);

// Vertical line
set(gca(), "auto_clear", "off");

xpolys([tL,tH,tY,tI;tL,tH,tY,tI],[0,0,0,0;Lmax,Hmax,Ymax,Imax]);

// Text of maximum point
xstring(tL,Lmax,Ltext); xstring(tH,Hmax,Htext);
xstring(tY,Ymax,Ytext); xstring(tI,Imax,Itext);
xlabel('Year');
set(gca(), "auto_clear", "on");

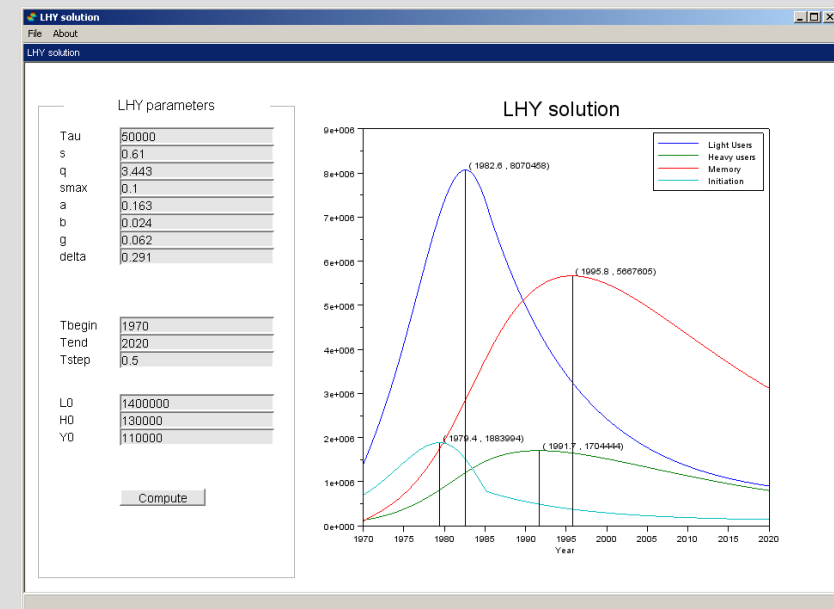
my_plot_axes = gca();
my_plot_axes.title.font_size = 5;
my_plot_axes.axes_bounds = [1/3,0,2/3,1];
my_plot_axes.title.text = "LHY solution";
endfunction
```

Step 16: Default initial plot

The last step for "LHY_Gui" is to compute the solution with the default data and to re-draw the window.

```
// Compute solution
sol = solvingProblem(param, sim);

// Redraw window
drawlater();
newaxes();
displayProblem(sol);
drawnow();
```



Step 17: Linking the computational function to the button

The "syscompute" function is associated to the "Compute button" (see step 13) and performs the following steps:

- it retrieves data from the object framework: this process is done by the function "findobj" that searches from all graphical objects those having the prefixed target tag;
- it evaluates the available data in the field converting these data from string into real using the function "evstr";
- it performs the simulation with the "solvingProblem" function;
- it redraws the window.

```
function syscompute
// retrieve data
param = [];
tau = findobj("tag", "Tau"); param.tau = evstr(tau.string);
s = findobj("tag", "s"); param.s = evstr(s.string);
q = findobj("tag", "q"); param.q = evstr(q.string);
smax = findobj("tag", "smax"); param.smax =
evstr(smax.string);
a = findobj("tag", "a"); param.a = evstr(a.string);
b = findobj("tag", "b"); param.b = evstr(b.string);
g = findobj("tag", "g"); param.g = evstr(g.string);
delta = findobj("tag", "delta"); param.delta =
evstr(delta.string);

sim = [];
Tbegin = findobj("tag", "Tbegin"); sim.Tbegin =
evstr(Tbegin.string);
Tend = findobj("tag", "Tend"); sim.Tend = evstr(Tend.string);
Tstep = findobj("tag", "Tstep"); sim.Tstep =
evstr(Tstep.string);
L0 = findobj("tag", "L0"); sim.L0 = evstr(L0.string);
H0 = findobj("tag", "H0"); sim.H0 = evstr(H0.string);
Y0 = findobj("tag", "Y0"); sim.Y0 = evstr(Y0.string);

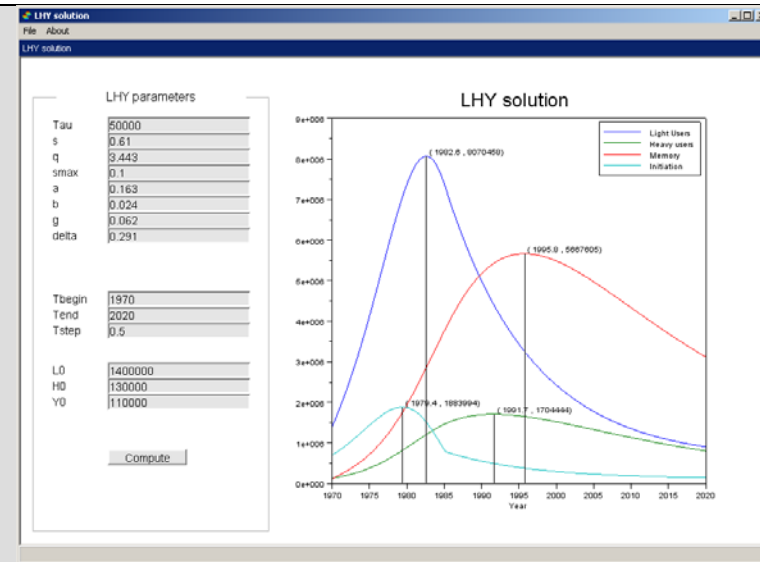
// Compute solution
sol = solvingProblem(param, sim);

// Redraw window
drawlater();
my_plot_axes = gca();
my_plot_axes.title.text = "LHY solution";
newaxes();
displayProblem(sol);
drawnow();
endfunction
```

Step 18: Running and testing #1

Change the Scilab working directory to the current project working directory and run the command `exec('LHY_MainGui.sce',-1)`.

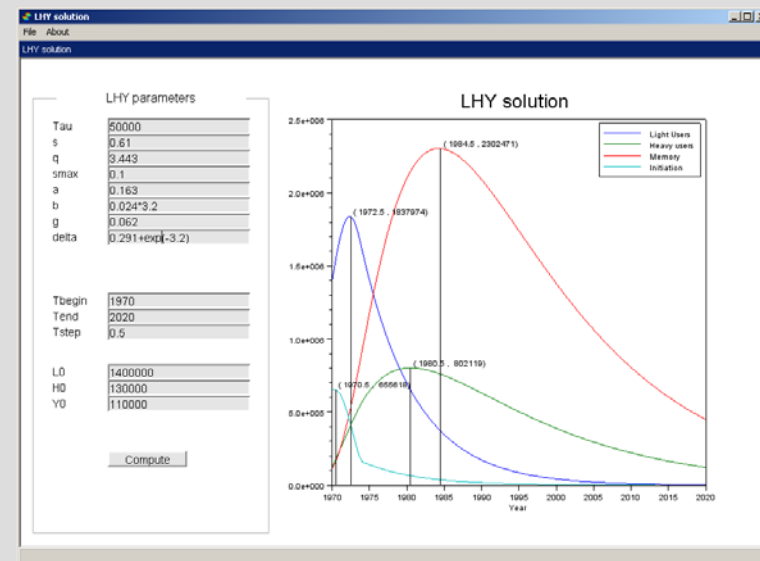
This command runs the main program and plots a chart similar to the one shown on the right.



Step 19: Running and testing #2

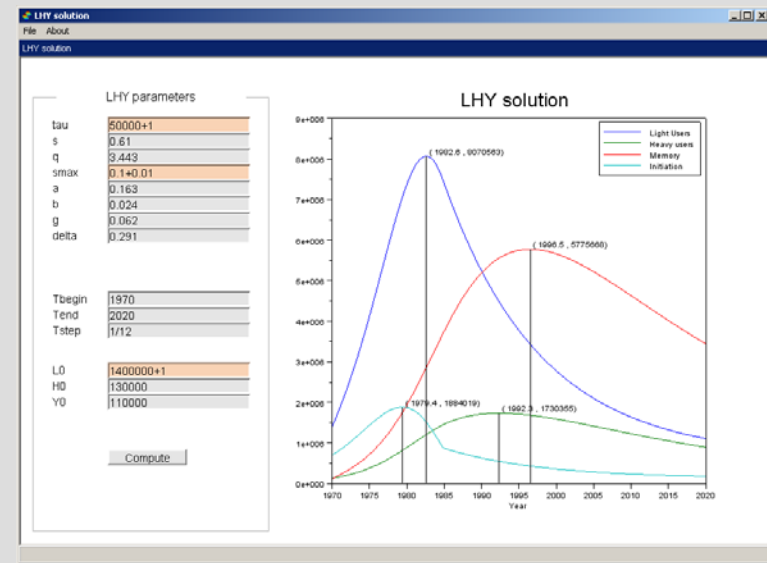
Since we have used the function "evstr" it is possible to insert expression in the input masks and see, in an easy way, the effects of parameters changing in the solution.

For example, on the right we have reported the simulation where the parameters $b = 0.024 * 3.2$ and $\delta = 0.291 + \exp(-3.2)$.



Step 20: Exercise #1

Try to modify the program adding the following feature: Every time the "compute button" is pressed the program check if the LHY parameters are changed with respect to the default values and if a parameter is changed then the associated input mask is colored using the orange color as depicted in the figure.



Step 20: Exercise #1 (Hints)

Some hints:

- copy the mlist data "param" and "sim" into a global data structure and save the name of their fields;
- add a "updatedFields" function into the "syscompute" function;
- write the "updatedFields" function.

```
// global default data
global gparam gsim pf sf;
gparam = param;
gsim = sim;
// param and sim fields
pf = fieldnames(gparam); // param fields
sf = fieldnames(gsim); // sim fields
```

```
// Update fields
updateFields(param, sim);
```



```

function updateFields(param, sim)
    global gparam gsim pf sf

    // Loop over gparam/param elements
    for i=1:size(pf,1)
        // Get current field
        cf = pf(i);
        // check if they are different
        checkequal = 0;
        if(abs(gparam(cf)-param(cf))<= 1e-12) then
            checkequal = 1;
        end
        // Find object
        obj = findobj("tag", cf);
        if isempty(obj) then
            // object not found, skip, some problems?
            continue;
        end
        // Colorize
        if(~checkequal) then
            // Object is modified: use new color
            obj.BackgroundColor = [0.98 0.83 0.71];
        else
            // Object is not modified: use default color
            obj.BackgroundColor = [.9 .9 .9];
        end
    end

    // The same code for the field gsim/sim
endfunction

```

Step 21: Concluding remarks and References

In this tutorial we have shown how a GUI can be developed for the LHY model.

On the right-hand column you may find a list of references for further studies.

1. Scilab Web Page: Available: www.scilab.org.
2. Openeering: www.openeering.com.

Step 22: Software content

To report a bug or suggest some improvement please contact Openeering team at the web site www.openeering.com.

Thank you for your attention,

Manolo Venturin

```
-----  
GUI MODEL IN SCILAB  
-----  
  
-----  
Directory: model  
-----  
LHY_Color_Gui.sci      : GUI for the LHY model (ex. 1)  
LHY_Gui.sci           : GUI for the LHY model  
LHY_Initiation.sci    : Initiation function  
LHY_System.sci        : LHY system  
  
-----  
Main directory  
-----  
LHY_MainGui.sce        : Main console GUI program  
LHY_MainGuiColor.sce  : Solution of the exercise  
license.txt            : The license file
```