7

Getting Started

7.1 Construction of a Simple Diagram

Scicos contains a graphical editor that can be used to construct block diagram models of dynamical systems. The blocks can come from various palettes provided in Scicos or can be user-defined. In this section, we describe how the editor can be used to construct simple models and how these models can be simulated.

7.1.1 Running Scicos

Scicos is a Scilab toolbox included in the ScicosLab package. The Scicos editor can be opened by the `scicos` command

```
⟼scicos;
```

This command opens up an empty Scicos diagram named by default `Untitled`. Called with an argument, it can open up an existing diagram:

```
⟼scicos my_diagram.cos;
```

The Scicos main window displaying an empty diagram is illustrated in Figure 7.1. The look of the main window may be slightly different under different window managers and operating systems. Editor functionalities are available through pull-down menus placed at the top of the editor window. The manual page of each functionality is displayed by selecting `Help` in the `Misc` menu and then the item of interest in the menu.

Some of the editor functionalities can also be accessed by clicking the right mouse button. Finally, keyboard shortcuts can be used for various operations. For example, typing an “r” activates the operation `Replot`, which centers and redraws the diagram. Keyboard shortcuts can be defined by the user.

Note that when Scicos is running, the Scilab window is not active. It can be activated with the `Activate Scilab Window` button in the Scicos `Tools` menu. This operation does not close Scicos, it simply deactivates it temporarily. To reactivate Scicos, do a simple mouse click on any Scicos diagram, or execute the `scicos` command in Scilab.

7.1.2 Editing a Model

To construct a model, we need to access Scicos blocks. Scicos provides many elementary blocks organized in different palettes that can be accessed using the operation `Palettes`

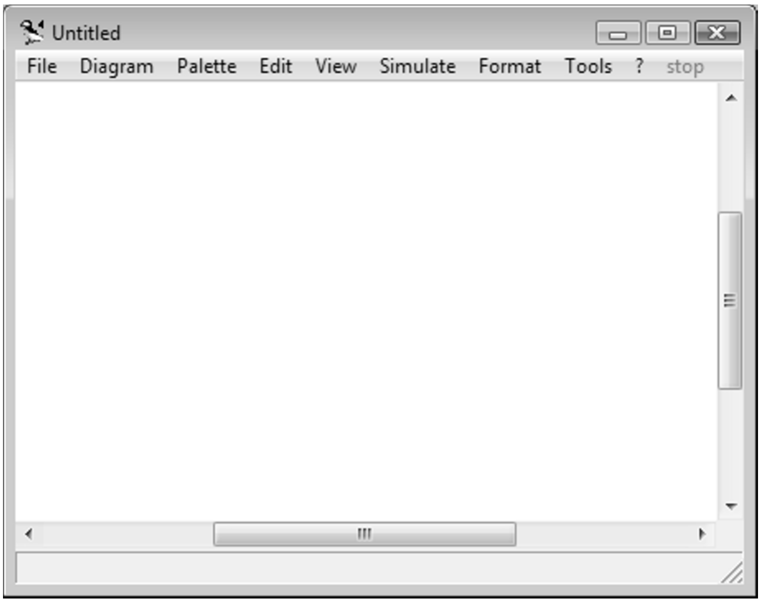


Figure 7.1. Scicos editor main window.

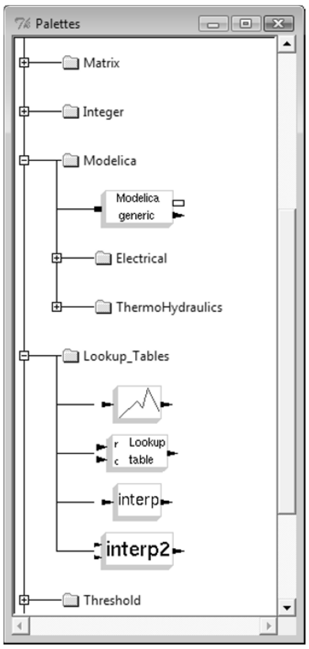


Figure 7.2. Palette browser.

in the `Edit` menu. This operation opens up a dialog box that includes the list of available palettes. By selecting a palette in the list, a new Scicos window opens up displaying the blocks available in this palette. A more direct and convenient way of accessing a block in a palette is via the `PalTree` operation in the `Edit` menu. This operation opens a Palette browser; see Figure 7.2. Blocks from palettes can be copied into the main Scicos window by dragging the desired block and dropping at the location where the block is to be copied in the Scicos window.

The `Sources` and `Sinks` palettes contain respectively blocks generating signals without any inputs and blocks without any outputs such as scopes and Write-to-file blocks. We start by copying from these palettes three blocks as illustrated in Figure 7.3. In this figure we have copied a sinusoid signal generator, a scope, and an event generator (`Event Clock`). The first block generates on its unique output port a sine function. We wish to display this signal using the scope. This can be done by connecting the output of the signal generator to the input of the scope by clicking first on the output port and then near the input port of the scope. The `Align` operation in the `Edit` menu can be used to align the two ports beforehand in order to make sure the link becomes horizontal.

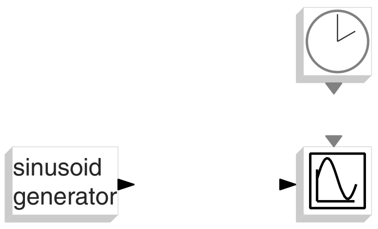


Figure 7.3. Scicos diagram in construction. Blocks are copied from the palettes.

The `Event Clock` is used to activate the scope block periodically with the desired frequency. Every time the scope is activated, it reads the value of the signal on its input port (which is nothing but the value of $\sin(t)$ generated by the signal generator). This value is then used to construct the curve that is displayed in the scope window. To specify that the scope block is activated by the event generator, the activation output of the event generator should be connected to the activation input of the scope. The result then looks like the diagram in Figure 7.4. This diagram is now complete.

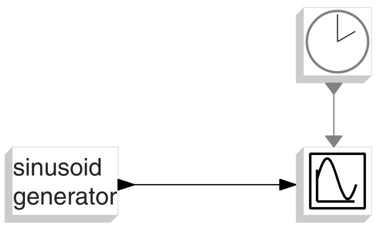


Figure 7.4. Completed Scicos diagram.

Note that Scicos diagrams contain two different types of links. The regular links transmit signals and the activation links transmit activation timing information. By default, the activation links are drawn in red and the regular links in black, but this can be changed by the user. Also note that regular ports are placed on the sides of the blocks whereas activation inputs and outputs are respectively on the top and at the bottom of the blocks. Most blocks follow this convention, but the user can define new blocks with ports arbitrarily placed.

The editor provides, through pull-down menus, many functionalities to change the look of the diagram and that of the blocks such as changing color, size, and font.

7.1.3 Diagram Simulation

To simulate a diagram, it suffices to select the `Run` operation from the `Simulate` menu. Simulation parameters can be set by the `Setup` operation in the same menu. There we can, for example, adjust the final simulation time.

Running the simulation for the system in Figure 7.4 leads to the opening of a graphics window and the display of a sinusoidal signal. This window is opened and updated by the scope block. The simulation result is given in Figure 7.5. In this case, the final simulation time is set to 30. The default value of the final simulation time is very large. A simulation can be stopped using the `stop` button on the main Scicos window, subsequent to which the user has the option of continuing the simulation, ending the simulation, or restarting it.

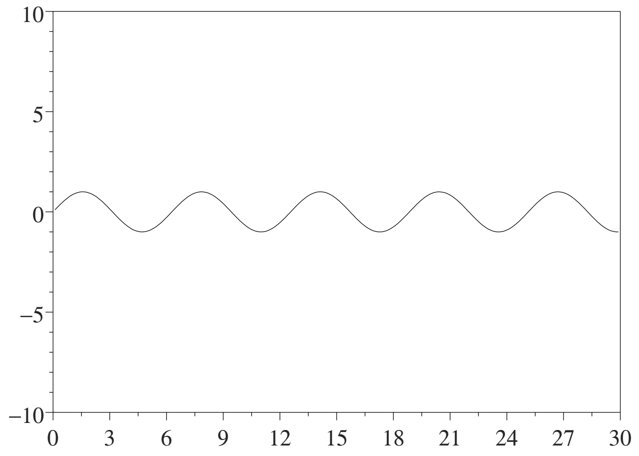


Figure 7.5. Scope window for the system in Figure 7.4.

A Scicos diagram can be modified and simulated again. Let us consider adding an integrator to the diagram in Figure 7.4 as illustrated in Figure 7.6. The integrator block gives as an output the integral of its input. This block comes from the `Linear` palette. We also replace the scope with a multi-input scope. Note that to create a split on a link, in particular here to create a link going to the integrator, off the link connecting the signal generator to the scope, the user must click first on the existing link at the position where the split is to be placed. Also note that to create a broken link, the user can click on intermediary points before clicking on the destination, which is necessarily an input port.

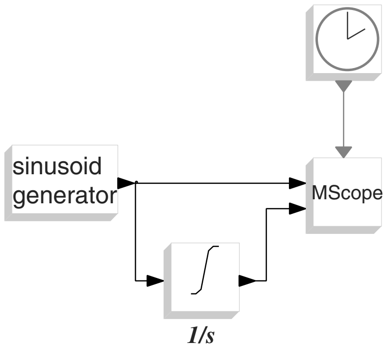


Figure 7.6. Modified Scicos diagram.

The simulation result for this new diagram is given in Figure 7.7.

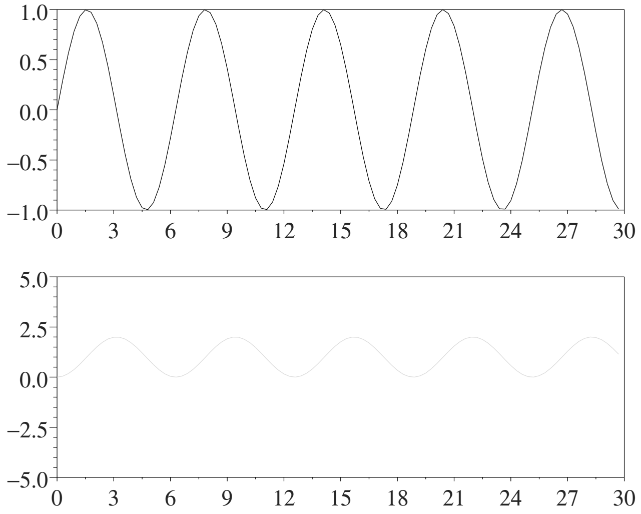
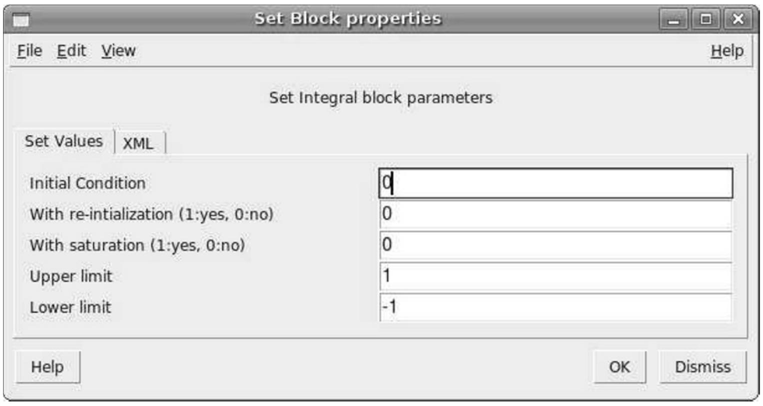


Figure 7.7. Scope window after simulation of the modified diagram.

7.1.4 Changing Block Parameters

The behavior of a Scicos block may depend on parameters that can be modified by the user. These parameters can be changed by clicking on the block. This action opens up a dialog box showing current values of the block parameters and allowing the user to change them. For example, the dialog box associated with the integrator block is illustrated in

Figure 7.8. The other blocks also have parameters. For example, in the sinusoid generator, we can set the frequency, the amplitude, and the phase. In the same way, certain properties of the scope window can be set by the parameters of the scope block.



Set Integral block parameters

Initial Condition	0
With re-intialization (1:yes, 0:no)	0
With saturation (1:yes, 0:no)	0
Upper limit	1
Lower limit	-1

Figure 7.8. Dialog box of the integrator block.

The value of a parameter can be defined by any valid Scilab instruction. For example, the frequency of the sinusoid generator can be set to `2*%pi/10`. The Scilab instruction can also include Scilab variables, but these variables must have been previously defined in the "context" of the diagram. Such variables are called *symbolic parameters*. Symbolic parameters and the context will be discussed in the next section.

Let us now consider more complicated examples in which the systems we want to model are given by differential equations. An integrator block is used to define each state of the system.

SIR Model for Spread of Disease

We consider a simple model of how a disease can spread in a community [5]. Let $s(t)$ be the fraction of the population susceptible to getting infected as a function of time t. Also let $i(t)$ be the infected (and infectious) fraction of the population and $r(t)$ the recovered, and thus immune, fraction of the population. Then the SIR model can be expressed as follows:

$$\dot{s} = -as(t)i(t), \tag{7.1a}$$
$$\dot{i} = as(t)i(t) - bi(t), \tag{7.1b}$$
$$\dot{r} = bi(t), \tag{7.1c}$$

where a and b are positive parameters.

To model this system in Scicos, we set the state of one integrator to s, another one to i, and a third one to r. It is then straightforward to construct the inputs of the integrators from their outputs, as can be seen in Figure 7.9. Here we have set a to 1 and b to 0.3. For the reader less familiar with block diagrams, the $\times$ and Σ blocks multiply and add inputs respectively. Since we are calling the outputs of the integrator blocks s, i, and r, the Scicos implementation of the differential equation in (7.1a) can be thought of as the integral equation

$$s(t) = -\int_{t_0}^{t} s(\tau)i(\tau)d\tau + s(t_0),$$

where $s(t_0)$ is the initial condition of s at time t_0. Similarly (7.1a), (7.1c) are implemented as integral equations in the Scicos diagram. However, they are integrated as differential equations by Scicos using a numerical ODE solver.

The simulation result for this SIR model is given in Figure 7.10. The variable s is initialized to 0.999, and i to 0.001.

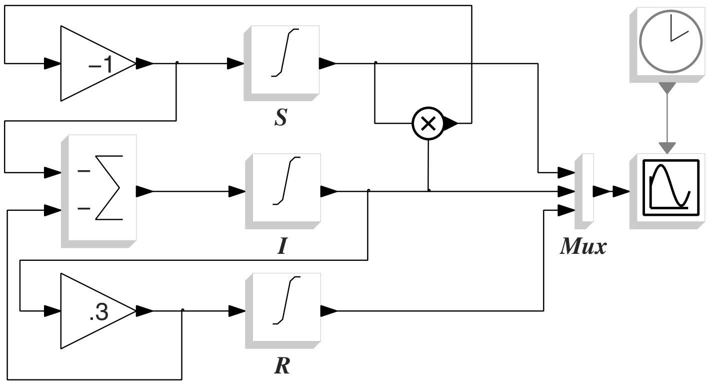


Figure 7.9. Scicos implementation of the SIR model.

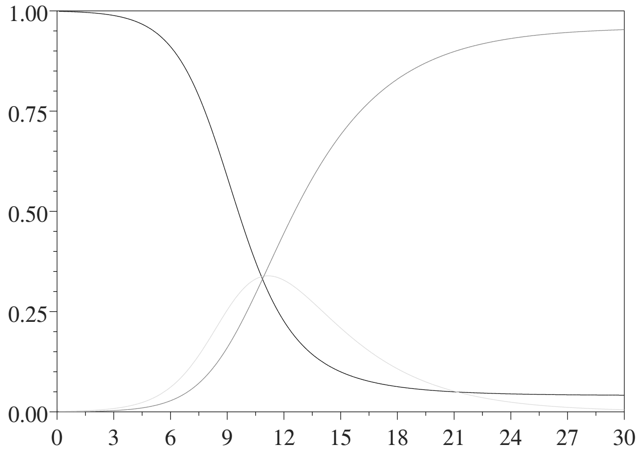


Figure 7.10. The simulation shows, as expected, that the percentage of the recovered population is increasing and that of the susceptible population is decreasing. The percentage of population infected reaches its maximum at a time called the peak of the epidemic.

Chaotic Dynamics of a Rössler Attractor

The Rössler system [54] given below has chaotic behavior for certain values of the parameters a, b and c:

$$\begin{aligned}\dot{x} &= -(y+z),\\ \dot{y} &= x + ay,\\ \dot{z} &= b + z(x-c).\end{aligned}$$

This system is modeled in Figure 7.11 with $a = b = 0.2$ and $c = 5.7$. The initial conditions are set to zero. The 2D scope is used to plot y against x. The result is given in Figure 7.12.

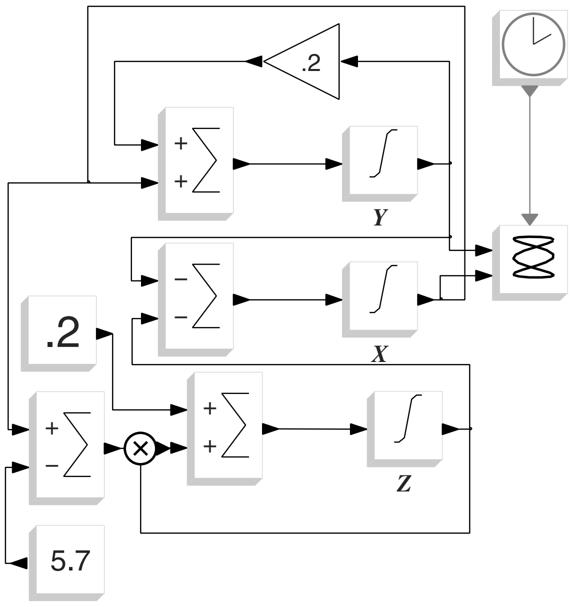


Figure 7.11. Scicos implementation of the Rössler attractor.

7.2 Symbolic Parameters and Context

Often it is useful to use symbolic parameters to define block parameters. This is particularly the case if the same parameter is used in more than one block or if the parameter is computed as a function of other parameters. Symbolic parameters are simply Scilab variables that must be defined in the context of the diagram before being used in the definition of block parameters.

Each Scicos diagram contains a context. The context is simply a Scilab script used to define Scilab variables, which can then be used to define block parameters. To access

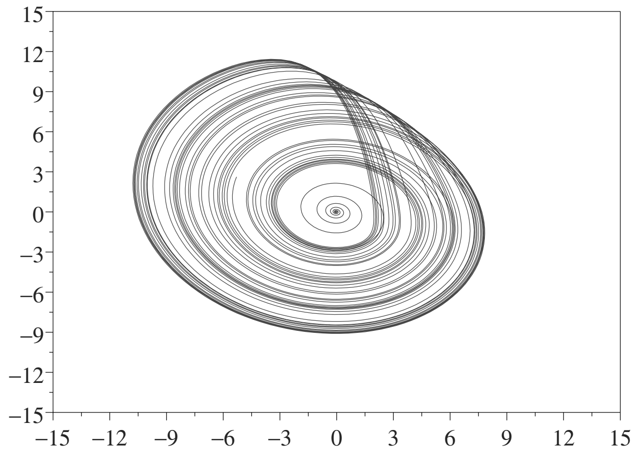


Figure 7.12. Output of the 2D scope for the Rössler model.

the context of the diagram, use the `Context` button in the `Edit` menu. This opens up an editor. The user can use this editor to write a Scilab script, that is, a set of Scilab commands that are executed by Scilab after a click on the `ok` button. For example, if the command `C=1` is placed in the context, then `C` can be used to define a block parameter. If the value of `C` is changed in the context, the parameter of the block is automatically updated accordingly.

Let us now consider an example in which the context is very useful for defining a generic diagram. We consider the construction of a sampled-data observer for a linear system. The system is considered to be modeled as a continuous-time state-space linear system:

$$\dot{x} = Ax + Bu, \tag{7.2}$$

$$y = Cx, \tag{7.3}$$

where A, B, C are constant matrices.

In developing controls for a system like this, it is often easier to develop them in terms of the state x. State feedback controls are one example. But the only thing that may be available in practice is the output y. A state observer is another dynamical system, which accepts as input u and y, and gives as an output $\hat{x}$, which has the property that $x - \hat{x}$ goes to zero independent of the initial conditions of either (7.2) or the observer. Observers have the nice property of being able to return to giving good estimates after disturbances. The speed with which the error goes to zero can be specified to give good estimation but avoid being overly sensitive to disturbances.

A continuous-time observer can be constructed as follows:

$$\dot{\hat{x}} = A\hat{x} + Bu + K(y - C\hat{x}). \tag{7.4}$$

The matrix K must be chosen so that the eigenvalues of $A - KC$ have negative real parts. This ensures that the estimation error $\tilde{x} = \hat{x} - x$ satisfying $\dot{\tilde{x}} = (A - KC)\tilde{x}$ will go to zero.

The discrete-time (sampled data) observer is obtained by first constructing the corresponding continuous-time observer using the method of pole placement, and then discretizing it. We begin by taking fixed dimensions and randomly generated matrices. We will then explain how to make this a generic diagram.

The Scilab script to perform this procedure with random matrices, placed in the context of the diagram, is the following:

```
n=5;m=2;dt=.2;
A=rand(n,n);A=A-max(real(spec(A)))*eye()
B=rand(n,1);C=rand(m,n);D=zeros(m,1);
x0=rand(n,1);
K=ppol(A',C',-ones(x0))';
Ctr=syslin('c',A-K*C,[B,K],eye(A),zeros([B,K]))
Ctrd=dscr(Ctr,dt)
[Ad,Bd,Cd,Dd]=abcd(Ctrd)
```

The original system is constructed with random matrices, and the A matrix is modified to make sure the system is not exponentially unstable. The Scilab pole placement (eigenvalue assignment) function `ppol` is used to obtain the observer gain matrix K, and `dscr` is used to discretize the observer. `dt` is the sampling period.

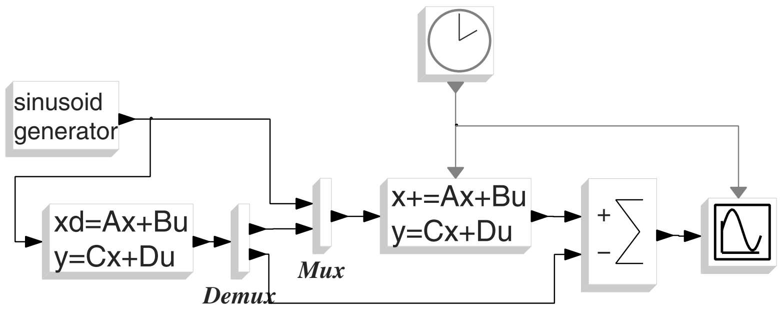
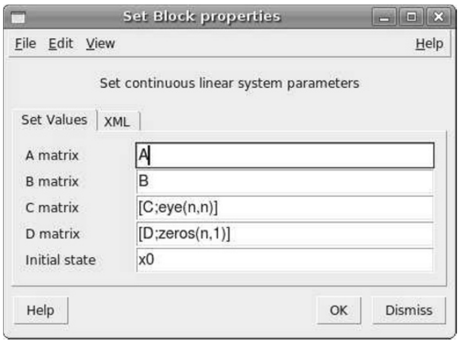


Figure 7.13. Scicos diagram including the system and the discrete-time observer.

The Scicos diagram in Figure 7.13 is used to simulate the observer performance. The input u is set to $\sin(t)$ and the estimation error is displayed using a scope. The continuous-time and discrete-time state-space linear system blocks are used to model the original system and the observer. The parameters of the first are set as illustrated in Figure 7.14 and the second as in Figure 7.15. The first block also outputs its internal state to be used for generating the estimation error displayed by the scope. `dt` is used to define the period of the `Event Clock`.

The use of symbolic parameters is particularly useful in this case because it allows us to construct a generic system observer diagram. To change the system matrices A, B, and C, the size of the system, the size of the outputs, or the discretization time, we only have to modify the definition of the m, n, and system matrices. No change is made to the diagram. In fact, by rewriting the context as follows we can make the diagram completely generic:

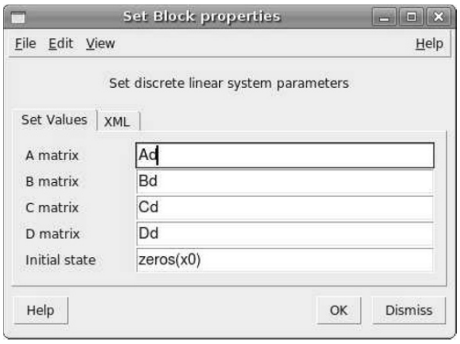
```
load('datafile')
K=ppol(A',C',-ones(x0))';
Ctr=syslin('c',A-K*C,[B,K],eye(A),zeros([B,K]))
Ctrd=dscr(Ctr,dt)
```



Set continuous linear system parameters

A matrix	A
B matrix	B
C matrix	[C;eye(n,n)]
D matrix	[D;zeros(n,1)]
Initial state	x0

Figure 7.14. Dialog box of the block realizing the original system; the output of the block contains also its state.



Set discrete linear system parameters

A matrix	Ad
B matrix	Bd
C matrix	Cd
D matrix	Dd
Initial state	zeros(x0)

Figure 7.15. Dialog box of the discrete-time linear system realizing the observer.

```
[Ad,Bd,Cd,Dd]=abcd(Ctrd)
```

We just have to make sure that the data file `datafile` contains variables `A`, `B`, `C`, `D`, `x0`, and `dt` before launching Scicos. This file can be created in Scilab as follows:

```
⟼ save('datafile'',A,B,C,x0,dt)
```

Remark

If the context contains many lines of Scilab code, it is convenient to place the code in a separate script file and execute it with a single `exec` command in the context. However, if the file being executed by the `exec` command is changed when the diagram is already open, and the user wants the modification to be taken into account, he or she should do an `Eval` because Scicos has no way of knowing whether the file has been changed.

It should also be noted that using a separate script file implies that the Scicos diagram is not self-contained and this script file must always accompany the diagram.

7.3 Virtual blocks and Hierarchy

Some blocks in Scicos palettes are not standard blocks realizing computations during the simulation. These virtual blocks facilitate the modeling task and are often used by the designer to improve the presentation of the Scicos diagram. For example the `Goto` and

`From` blocks can be used to create a link without having to draw it. This is useful when the link is connects two blocks far apart, possibly in different submodels. Similarly, the bus creator (`BUS`) and the bus selector `DEBUS` blocks allow user to group a set of links into a single "link" called bus. But the most important virtual block is a Super block, which will be presented in details here. Note that the virtual blocks are removed in the first phase of compilation.

It is not good modeling practice to place too many blocks in a diagram at the same level because the diagram becomes incomprehensible and difficult to read and debug. For large systems, it is useful to use the Super block facility to construct a hierarchical model. A Super block looks like any other block: it can be moved, copied, resized, etc., but its behavior is defined by the Scicos submodel within it.

7.3.1 Placing a Super Block in a Diagram

There are two ways to place a Super block in a Scicos diagram.

`Region-to-Super-block`

If the submodel the user wants to place in a Super block already exists in the model, then the `Region-to-Super-block` operation of the `Diagram` menu can be used to place it in a Super block. This is done by first selecting the blocks (usually done by specifying the region of the diagram) to be placed in the submodel and then the application of the `Region-to-Super-block` operation. The selected blocks are automatically replaced with a Super block with the appropriate number of input and output ports.

Let us consider a simple example. In the diagram of Figure 7.13, we used a linear system block to model the original system, but since we also needed its internal state, we had to concatenate it to the output, which we then had to split using the `Demux`[1] block in order to generate separately the state `x` and the output `y`. The diagram would have been much clearer if the linear system block had two output ports generating `x` and `y`. Unfortunately, that is not the way this block works.

But the combination of the linear system block and `Demux` does exactly what we want. So it is natural to consider constructing a Super block out of these two blocks. This can be done using the `Region-to-Super-block` functionality and selecting a region containing these two blocks. The result is illustrated in Figure 7.16.

Super Block in Palette

A Super block can also be placed in a diagram by copying the empty Super block provided in the `Others` palette. The desired submodel can then be constructed inside the Super block.

7.3.2 Editing a Super Block

To edit the content of a Super block, it suffices to click on it to "open" it. This opens up a new Scicos editor window resembling the main Scicos window displaying the content of the Super block. For example, by opening the Super block in Figure 7.16, the model illustrated in Figure 7.17 is displayed in the new editor.

[1] Demux is network terminology for demultiplex and it means to separate channels.

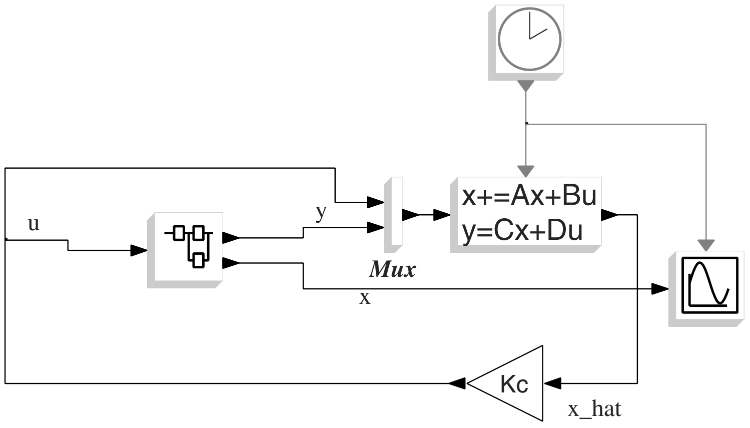


Figure 7.16. The linear system and `Demux` blocks are placed in a Super block.

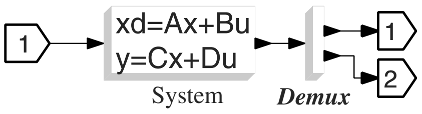


Figure 7.17. Inside the Super block.

Note that new blocks have been added to the submodel that were not present in the original model. These are the input, output port blocks. These blocks define the connection from the inside of a Super block to the outside world. There are exactly as many input (respectively output) port blocks inside the Super block as there are input (respectively output) ports on the Super block.

The Super block can be edited just like the main Scicos diagram. If the number of input or output port blocks is modified, the number of ports of the Super block in the main window is adjusted automatically accordingly. Note that the port blocks must be numbered consecutively and the corresponding ports on the Super block are also numbered counting consecutively from the top to the bottom.

Any number of Super blocks, at various levels of hierarchy can be open and edited in the Scicos editor at any time. Copy and paste operations work across the corresponding editor windows.

There is no need to close any diagram to compile and run simulations; simply use the `Simulation` menu on the top diagram while leaving all other windows open without any modification. There is also no need to close any windows to access the Scilab shell at prompt level. For that use the `Activate Scilab Window` button in the `Tools` menu. This is particularly useful for loading/defining new functions (for example new block interfacing functions), performing computations (post/pre data processing) using From and To Workspace blocks, etc. We can return to Scicos by simply clicking on any Scicos window.

The hierarchical structure of a Scicos diagram can be visulized in a browser using the `Browser` button in inside the `Tools` menu. This browser can also be used to open any diagram simply by double-clicking on its name. See Figure 7.18. The current diagram (the one that has focus) is highlited in the browser.

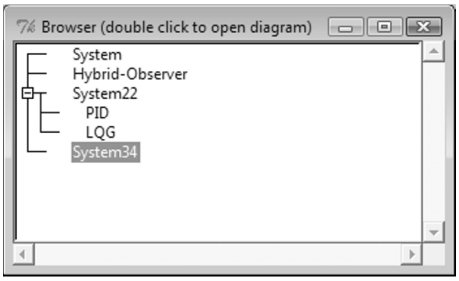


Figure 7.18. Scicos browser.

7.3.3 Scope of Variables in Super Block Contexts and Masking

As we have seen previously, block symbolic parameters are Scilab variables that are defined in the context of the diagram. But for a block inside a Super block, the definition of its symbolic parameters need not necessarily be done in the context of the Super block itself, it may be done in the context of the Super block containing the first Super block (if any), the Super block containg the Super block containing the first Super block (if any), etc., or the context of the top diagram. It can even be done in the Scilab environment. We will discuss this latter mechanism in Chapter 11.

The scoping rules that apply are fully consistent with Scilab's scoping rules: the definition of a variable is first searched in the local Super block context, if it is not found, the one-level up context is searched, and so on. This provides a rigorous methodology for using symbolic parameters avoiding any risk of name conflicts when copying a Super block from one location to another.

To understand the scoping rules, consider the example in Figure 7.19. The context of the main diagram defines the variables `A`, `x`, `B` and `C`. The context of the Super block defines `B` and `y`. Available parameters and their values inside this Super block are: `A=3, B=4, x=%pi/7, y=1, C=1`.

Only the symbolic parameters that are defined as Scilab variables in a context can be used as block parameter. The available symbolic parameters in any diagram can easily be inspected using the `Available parameters` button in the `View` menu; see Figure 7.20 for an example.

The scoping rules in Scicos also provides the proper mechanism for an intuitive masking operation. The masking operation consists of changing the behavior of a Super block in a diagram so that it looks just like a basic block. In particular, when we double click on a masked Super block, Scicos does not open a new editor window but opens up the standard block GUI used to set block parameters. When a masked Super block is created, these parameters, which are simply the parameters that are used inside the Super block but not defined inside the Super block, are identified automatically by Scicos. Let's consider the example in Figure 7.19 and suppose that the blocks inside the Super block use the variables `x`, `y` and `B`; see Figure 7.21.

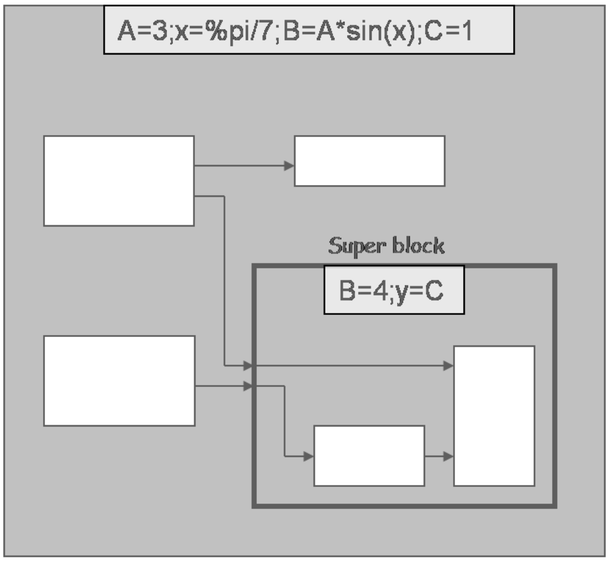
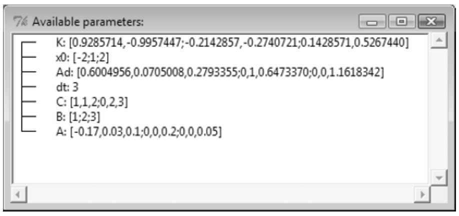


Figure 7.19. Inside the Super block `A=3, B=4, x=%pi/7, y=1, C=1`.



```
K: [0.9285714,-0.9957447;-0.2142857,-0.2740721;0.1428571,0.5267440]
x0: [-2;1;2]
Ad: [0.6004956,0.0705008,0.2793355;0,1,0.6473370;0,0,1.1618342]
dt: 3
C: [1,1,2;0,2,3]
B: [1;2;3]
A: [-0.17,0.03,0.1;0,0,0.2;0,0,0.05]
```

Figure 7.20. Available Scilab variables for use as block parameters.

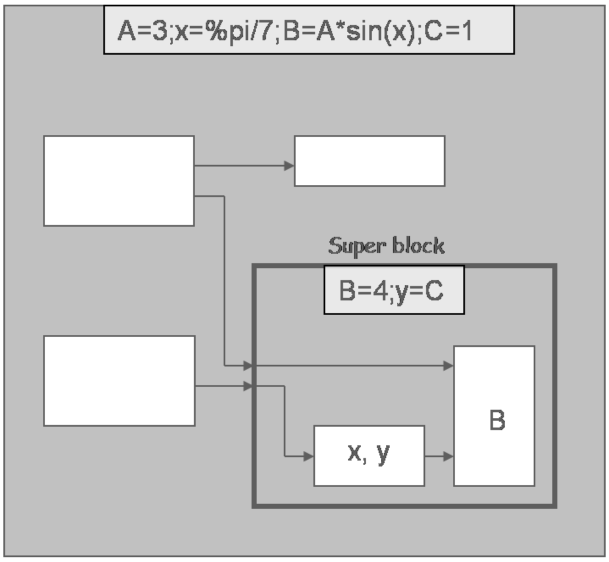


Figure 7.21. The parameters of the masked Super block will be `C` and `x`.

Masking of the Super block in this case automatically finds the parameters of the masked block, which are `C` and `x`. Why? Because `C` and `x` are not defined but are needed inside the Super block.

The use of Scilab functions such as `load` and `exec` that define Scilab variables implicitly should be avoided in Super block contexts for masking to function properly.

7.4 Save and Load Operations

7.4.1 Scicos File Formats

Scicos diagrams can be saved in two different formats. The most common format is a binary file for which by convention the extension `.cos` is used. This file is simply a binary Scilab data file and can be loaded in Scilab (assuming its name is `myfile.cos`) using the command

```
⟼load myfile.cos
```

There are three Scilab variables inside `.cos` files:

- `scs_m`: this Scilab structure contains all the information concerning the diagram
- `%cpr`: this Scilab structure contains the result of the compilation. It is an empty structure if the diagram had not been compiled before being saved (diagram compilation will be discussed later)
- `scicos_ver`: a string indicating the version of Scicos that has produced the diagram.

Normally, the user never needs to load this file in Scilab directly except for batch processing, which is discussed in Chapter 11. The loading is done in the Scicos environment using the `Open` button in the `File` menu. The file can also be loaded when launching Scicos as follows:

```
⟼scicos myfile.cos;
```

Scicos diagrams can also be saved in text format. The possible extensions of the text files are `.cosf` and .xml. The tt .cosf files are Scilab scripts that can be executed using the `exec` command, provided Scicos libraries are loaded in the environment. Note that Scicos libraries are not loaded by default into Scilab. They are loaded by the function `scicos`. So, just as in the binary case, the loading should be done by Scicos.

The expert user may be able to modify a diagram by editing the corresponding text file. But in most cases, it is much easier to use the Scicos graphical editor to do the editing.

Note that the name of the saved file, except the extension, corresponds to the name of the diagram. New diagrams are named by default `Untitled`. The name of the diagram can be changed using the `Rename` button in the `File` menu. The name is also changed if the file is saved, using the `Save As` button, under a different name. In that case, the name of the diagram becomes the name of the file (without the extension).

Finally note that when a diagram is saved window parameters, such as the position, size, zoom factor, etc., are saved and restored when it is opened.

7.4.2 Super Block and Palette

From within a Super block window, there are two saving options: one for saving the full diagram and another for saving the Super block alone. The contents of all the Super blocks present in a model are saved when the diagram is saved. So there is no need to save the content of a Super block unless we want to use it in the construction of another diagram. Super blocks can be saved just like any diagram in different formats. The saved file is identical to that of a main diagram. It can be loaded into a main Scicos diagram or inside a Super block.

A palette is also a standard Scicos diagram. Any Scicos diagram can be loaded as a palette. This means that blocks can be copied from it but the diagram itself cannot be edited. The button `Load as Palette` can be used to load any existing Scicos diagram as a palette. To avoid searching for palette files often used, Scicos provides the possibility of defining a list of palettes with their names and locations (see `Pal editor` in the menu `File`). These palettes can then be loaded simply by clicking on their names when the button `Palettes` is clicked. The default Scicos palettes are originally placed in this list.

7.5 Synchronism and Special Blocks

When two blocks are activated with the same activation source (for example the same `Event Clock`), we say that they are synchronized. In that case, they have the same activation times, and if the output of one is connected to the input of the other, the compiler makes sure the blocks are executed in the correct order. Synchronism is an important property. Two blocks activated by two independent clocks exactly at the same time are not synchronized. Even though their activations have identical timing, they can be activated in any order by the simulator.

On the other hand, two activations can be synchronous but not have exactly the same timings. For example, an activation can be a subset of another activation. Consider two activation sources, one generating a sequence of events with frequency 2 and another with frequency 1. In this case, half the events generated by the fast clock are simultaneous with the events of the slow clock. If these two activations are generated by two independent clocks, then they are not synchronized. So even when the two events are simultaneous, the blocks they activate can be activated in any order by the simulator. To enforce synchronism in this case, we have to make sure the two activations have the same source (same `Event Clock` for example). In this example this means that the slow activation is obtained from the fast activation.

There are two very special blocks in Scicos: the `If-then-else` and the event select blocks. They are in the `Branching` Palette. Even though they look like standard Scicos blocks and can be manipulated by the editor as such, strictly speaking they are not blocks. We shall give a detailed presentation of these blocks and the notion of synchronism in Chapter 8. Here, it suffices to say that these blocks are the only blocks that generate events synchronous with the incoming event that had activated them. These blocks, which can be considered as counterparts of conditional statements `If-then-else` and `Switch` in the C language, are used for conditional subsampling of activation signals.

Consider the diagram in Figure 7.22. The `If-then-else` block *redirects* the event it receives toward its `then` output port, activating the delay block `1/z` if its regular input is positive. If not, the activation goes out through the `else` port, which is not connected to anything. The `square wave generator` generates a series of alternating ones and minus ones. Every time the `1/z` block is activated, it adds the output of the generator to its content (initially set to zero). Since the activation takes place only when the output of the generator is positive, the content of `1/z` goes up by one at every other activation of the clock. This is confirmed by the result of the simulation illustrated in Figure 7.23. The period of the clock in this simulation is set to 1.

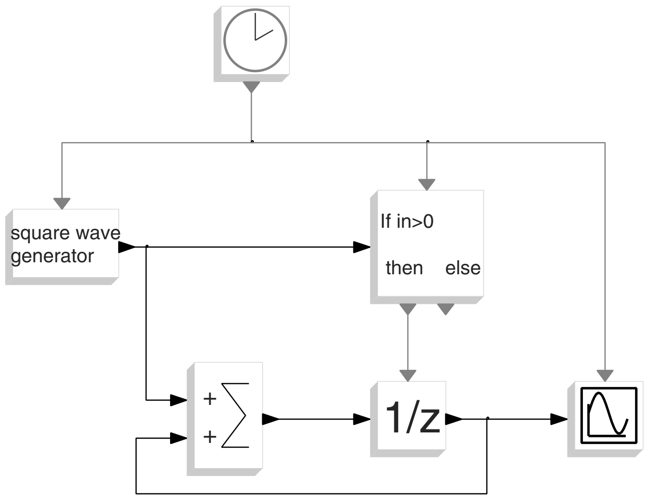


Figure 7.22. A synchronous Scicos diagram.

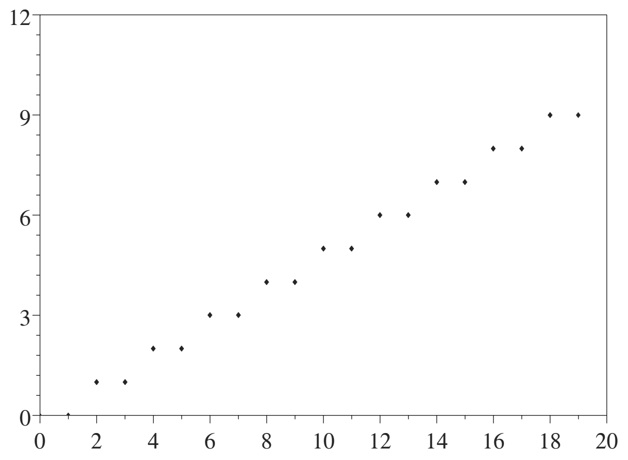


Figure 7.23. Simulation of Figure 7.22 showing that the counter goes up by one at every other activation of the clock.

There are other blocks in Scicos palettes that have special functionalities. For example, the `GOTO` and `FROM` blocks are used to establish connections without actually drawing links. These blocks are useful for connecting blocks that are situated in different Super blocks and at different levels of hierarchy.

The `END` block can be used to set the final time of simulation using a symbolic parameter defined in contexts. Note that the Scilab variables defined in the contexts can only be used for defining block symbolic parameters; they cannot be used for setting the simulation parameters, and in particular the final simulation time, in the `Setup` dialog of the `Simulation` menu.