Contents

Preface

About Simulink

Simulink is a *block diagram* based program for simulation of dynamic systems. Simulink must be used together with Matlab.[1] The mathematical model of the system to be simulated must be represented by blocks, which can be standard blocks available from the Simulink block-libraries or blocks created by the user. Some features of Simulink are:

- There is a large number of blocks for models of various types: Continuous-time and discrete-time transfer functions and state-space models, and models including nonlinearities. Furthermore, there are blocks for signal generation, for data storage, and for plotting.
- You can simulate hybrid systems, that is, systems consisting of both continuous-time and discrete-time (sub)systems, represented by blocks. One example is simulation of a control system in which a continuous-time process is controlled by a discrete-time controller.
- Compared to equation-based simulation programs, the graphical representation (as block-diagrams) makes it is easy to change the structure of the system since we can easily change the connections between the blocks, and add new blocks to the diagram and connect these to the existing diagram.
- The block diagram itself gives clear illustration of the structure of the system (the model).

[1] Matlab and Simulink are produced by **The MathWorks** (http://mathworks.com).

For whom this text is written

The text is written for students, teachers, researchers, and engineers who want to master Simulink in a few hours. Basic skills in using Matlab are assumed.

To be able to use Simulink you must have basic knowledge about linear dynamic systems, see e.g. [1].

Aims

The aims of this text is that you, after having carefully worked through all of the text, will be able to:

- Create a Simulink block-diagram ready for simulation.
- Simulate a block-diagram from the Simulink menu.
- Simulate a block-diagram by executing proper commands from the command-line or written in a script, that is, without using the menu. (I suggest you offer the necessary time to master using scripts. It will save you much time and effort may in the length (compared to just simulating from the menues). Personally, I use scripts for almost any Matlab and Simulink task.)

About this book

This book consists of descriptive text with figures, and frames of tasks that you should accomplish.

All files described in this book are available at **http://techteach.no**. It is of course up to you wether you create the files from scratch or if you download the files.

This book gives an introduction to Simulink which is sufficient for most simulation tasks. Several topics or featured are not covered, as:

- *S-functions* (system functions). An S-function is a computer language description of a Simulink block. S-functions can be written in Matlab, C, C++, Ada, or Fortran. S-functions allow you to create your own Simulink blocks. Examples of applications are estimators, advanced controllers, and signal processing functions.

- *Linearization tools* to automatically get a local linear model of an original nonlinear Simulink model at some operating point. With a linear model any linear analysis and design can be made, e.g. transfer function calculation, frequency response analysis, stability analysis, optimal controller design, pole placement controller design, etc.

- *Stateflow* which is an additional tool which can be purchased. Stateflow is built upon Simulink. Stateflow contains tools to define state diagram (state machine) control of models simulated by Simulink.

- *Real-Time Workshop* is an additional tool with which you can generate and execute stand-alone C code for developing and testing algorithms modeled in Simulink. The C code can be used in real-time (and non-real-time) applications, as simulation acceleration, rapid prototyping, and hardware-in-the-loop testing.

Many other add-ons are listed on
http://www.mathworks.com/products/simulink/.

Finn Haugen

Skien, Norway, March 2007

Chapter 1

Starting Simulink. The Simulink environment

You can start Simulink (from within Matlab) in several ways:

- Execute **simulink** at the Matlab command line.
- Menu selection **Start (in the Matlab window) / Simulink / Library Browser**.
- Click the **Simulink** button in the Matlab toolbar.

Start Simulink using the method you prefer.

The **Simulink Library Browser** window is thereby opened, see Figure 1.1.

Comments to the **Simulink Library Browser** window:

- *Block libraries*: The block libraries are listed to the left. Clicking on a library opens the blocks in that library. For example, in Figure 1.1 the **Continuous** library is opened.

  Note: If you install addition toolkits to Matlab and/or Simulink, new libraries are intalled, and they appear at the lower part of the library list. In Figure 1.1 the Control Design Toolkit and Stateflow libraries are examples of such additional libraries.
- *Search for blocks*: You can search for a specific block using the search area at the top of the **Simulink Library Browser** window.

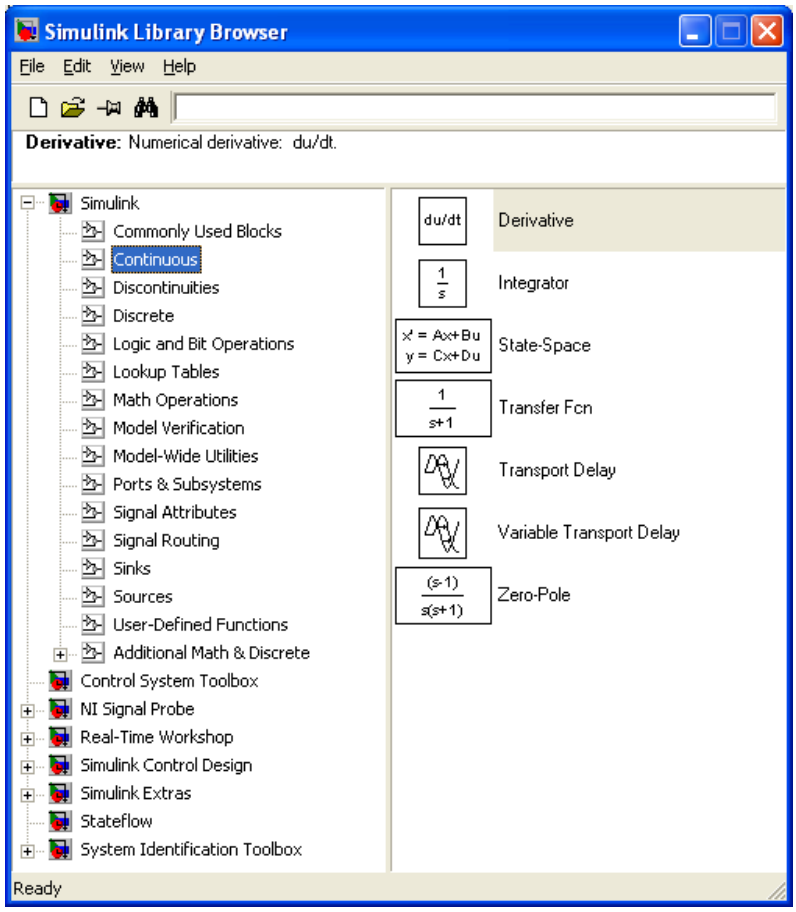

Figure 1.1: Simulink Library Browser

- *Opening a block diagram*: Using the **Open** button in the toolbar you can open an existing Simulink model file. Using the **New** button opens a new, blank block diagram.

Open each of the libraries in turn, and take a brief look at the contents.

Search for a block, e.g. an integrator.

Open a new block diagram, and then close it.

Chapter 2

How to construct block diagrams

2.1 Introduction

As an example, we consider the following first order differential equation:

$$T\frac{dx(t)}{dt} + x(t) = Ku(t-\tau) \tag{2.1}$$

where $T = 0.5$s (time constant), $K = 2$ (gain) og $\tau = 0.5$s (time delay). The input variable (the independent variable) u is a step from 0 to 2 at time 5s. The output variable (the dependent variable) is x, which is also the state variable of the system. The initial state is $x(0) = 4$. We will simulate the response in $x(t)$ from 0s to 10s. $u(t)$ and $x(t)$ will be plotted in the same figure.

The differential equation (2.1) can be represented in many ways in Simulink:

- Detailed block diagram representing the individual terms of the differential equation
- Transfer function (since the model is linear)
- State space (since the model is linear)
- LTI-model (Linear Time Invariant), which is a special model type supported by the Control System Toolbox

To get most out of this introductory example, we will use the first representation here, i.e. a detailed block diagram. (Transfer functions, state-space models, and LTI models are described in Ch. 4.)

2.2 Presenting the model on a proper form

The model (2.1) is differential equation including the time derivative $\dot{x}(t)$. The simulator will be used to calculate $x(t)$ which is the time *integral* of $\dot{x}(t)$. Let us start by writing the differential equation on *state-space form*[1]:

$$\dot{x}(t) = \frac{1}{T}\left[-x(t) + Ku(t-\tau)\right] \tag{2.2}$$

$x(t)$ is found by integrating $\dot{x}$ from time 0 to t:

$$x(t) = x(0) + \int_0^t \underbrace{\left\{\frac{1}{T}\left[-x(\theta) + Ku(\theta-\tau)\right]\right\}}_{\text{Right-hand side of diff. eq., i.e. } \dot{x}(t)} d\theta \tag{2.3}$$

This integral equation can be represented with the block diagram shown in Figure 2.1. (In this block diagram I have intentionally used block symbols which are a little different from those used in Simulink, just to demonstrate that a block diagram may be drawn in many ways.)

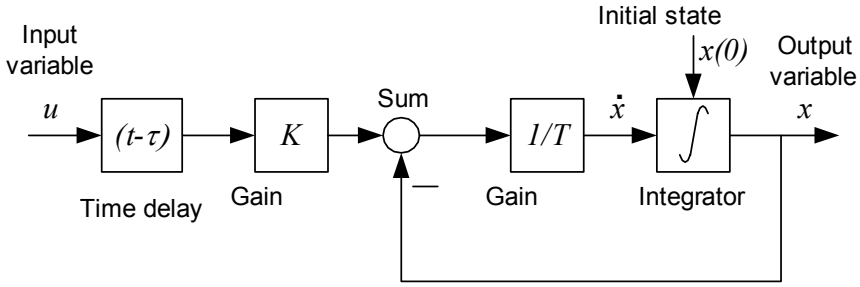

Figure 2.1: Block diagram of (2.3)

Actually, it is not necessary to explicitly develop the integral equation before drawing the block diagram, and experienced users don't. We can simply add one integrator for each of the time derivatives of the model. The output of the integrator is assumed to be the state variable, say x, and hence the input to the integrator is $\dot{x}$, and this input is constructed according to the right-side of the differential equation.

[1] A state-space model is a differential equation having only first order time-derivatives on its left-hand side.

2.3 Constructing the block diagram

Figure 2.2 shows the block diagram which we will now construct in Simulink.

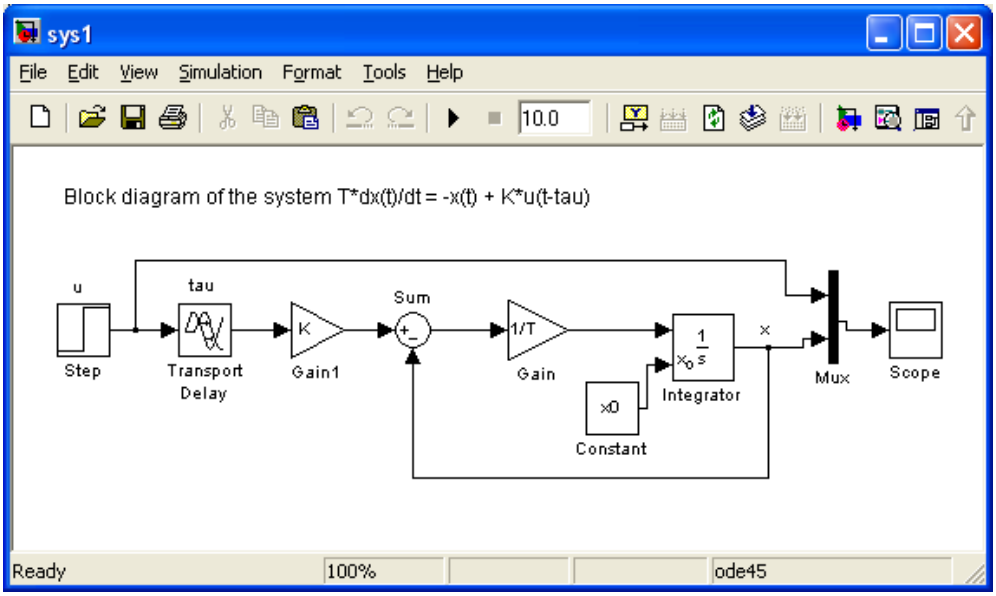

Figure 2.2: Block diagram model **sys1.mdl**

Select **File / New / Model** (or click the **Create a new model** button in the toolbar) in the Simulink window.

Typically, block diagram construction consists of the following basic steps:

1. Proper blocks are copied from the block libraries into the block diagram.
2. Then, the blocks are connected according to the hand-drawn block diagram.
3. Finally, the blocks are configured or parameterized, that is, values for the parameters of the blocks are entered into the blocks via the dialog box of each block.

In the following these three steps are explained in detail.

1. Copy the blocks into the block diagram

Do as follows:

1. Open the **Continuous** library. Click on the **Integrator** block with the right mouse button. While this button is held down, drag a copy of the block over to the block diagram.
2. Add two **Gain** blocks and one **Sum** block from the **Math Operations** library.
3. Add a **Step** block (it will be used to generate the input step) from the **Source** library.
4. Add a **Transport Delay** block from the **Continuous** library.
5. Add a **Mux** (multiplexer) block from the **Signal Routing** library.
6. Add a **Scope** block from the **Sinks** library. (This block will be used to plot responses.)

As you construct the block diagram you can manipulate the blocks as follows:

- *Move a block:* Select the the block, and move it.
- *Resize a block:* Select the block, and drag one of the handles of the block (a handle appears as a small square at the corner of the block).
- *Rotate a block:* Right-click on the block, and select **Format / Rotate** in the menu that appears.
- *Reverse a block:* Right-click on the block, and select **Format / Flip**.
- *Change the block name:* Click the block name under the block, and type the new name.
- *Hide the block name:* Right-click on the block, and select **Format / Hide**.
- *Copy, cut, paste a block:* This can be done in the usual Windows way, e.g. **Ctrl + c** (copy), **Ctrl + x** (cut), **Ctrl + v** (paste).
- *Manipulate several block simultaneously:* Start by selecting the blocks. Then perform the manipulation by right-clicking one of the selected block and select the proper action.

Manipulate the blocks as you wish.

You can add annotations (free text) anywhere in the block diagram by double-clicking and typing.

Add an annotation to the block diagram, cf. Figure 2.2.

It is wise to save the block diagram frequently!

Save the block diagram with name **sys1** (in any folder you prefer) using the **File / Save As** menu. (The file name becomes **sys1.mdl**.)

2. Connect the blocks

You can connect two blocks by drawing a line from the output terminal of the first block to the input terminal of the second block. (Alternatively, you can do in the reversed order, i.e. draw a line from the input to the output.)

Connect the blocks as shown in Figure 2.2.

A connection line can be manipulated in several ways:

- *Move the line:* Click the line, and move it.
- *Delete a line:* Select the line, and press the Delete button on the keyboard.
- *Give a name to the line:* Right-click the line, and select **Signal properties** in the menu that is opened, etc.

Manipulate the connection lines in the block diagram as you wish.

3. Configuring the blocks

We will now assign values to the parameters of the blocks. This can be done either by typing numerical values directly or by typing parameter names into the parameter fields of each block. One benefit of using parameter names is that it is time-saving in the long run since we avoid opening a block each time we need to change the value of a parameter, and it is less prone to typing errors. How does Simulink know the value of a parameter? It looks for the value in the Matlab workspace. The parameter

values may be defined at the Matlab command line, or by *defining the values in a Matlab script and running the script*. The latter is (in my opinion) the best way, so we will do that in the following.

Configure the blocks of **sys1.mdl** as described below.

- Open the parameter settings window of the **Step** block by double-clicking the block, cf. Figure 2.3. Enter the parameters as

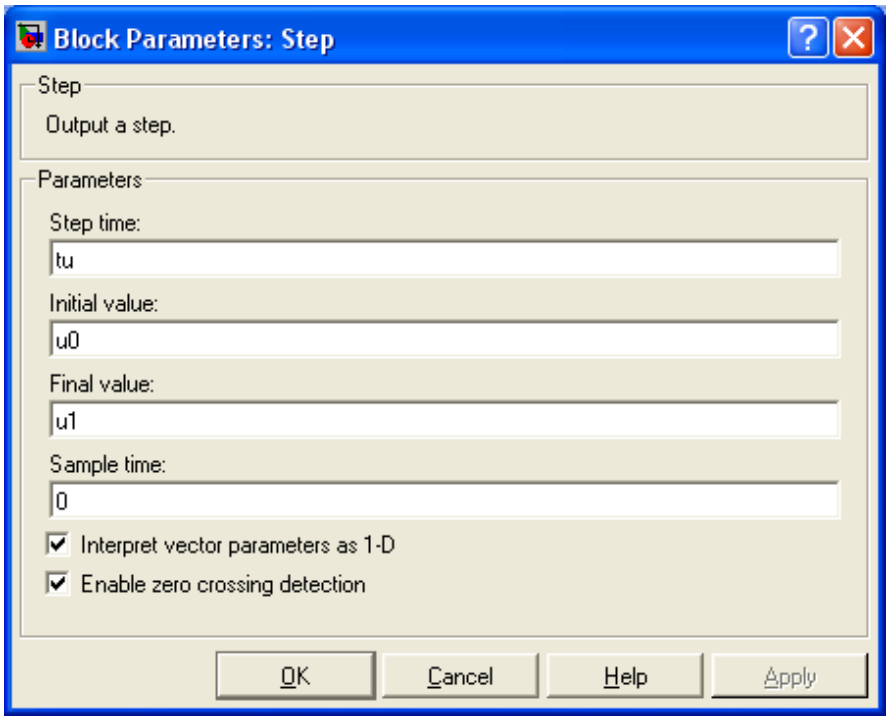

Figure 2.3: Parameter settings window of the **Step** block

shown in the figure. The parameter fields are fairly self-explanatory. If you need help, click the **Help** button in the window. Finally, close the window.

- **Transport Delay** block: Set parameter **Time delay** to **tau**. Keep the default settings in the other fields.
- **Gain1** block: Parameter **Gain** is set to **K**.
- **Sum** block: Select **Round** in the **Icon shape** pulldown list. The sign of each of the input terminals on the **Sum** block is defined with a plus or minus sign in the **List of signs** field. Empty terminals

(non-nonnected) terminals are defined using the | symbol (vertical bar). For the present block in our block diagram, set the list of signs to |+-.

- **Gain1** block: Parameter **Gain** is set to **1/T**.
- **Integrator** block: Set **Initial condition source** to **External** (which means that the initial value of the integrator output must be defined externally, at an input to the block – see below).
- **Constant** block: Set **Constant value** to **x0**.
- **Mux** block (multiplexer): This block is used to collect or bundle a number of signals into one vectorized (or multivariable) signal. Set **Number of inputs** to 2. Set **Display option** to **Bar**.

2.4 Subsystems

You can group a number of selected blocks into a *subsystem* which is represented by one single block with inputs and outputs. Here are some benefits of using subsystems:

- Complicated block diagrams can be given a more compact and informative representation, since the block diagrams on a higher level will look simpler (i.e. they have fewer blocks).
- A subsystem may represent a specific system which may be re-used, e.g. included in other block diagrams. Hence, with subsystems you can create modular models.

Figure 2.4 shows one example. There is a main system, **sys1sub**, and a subsystem created from the system **sys1** shown in Figure 2.2.

It is simple to create a subsystem:

1. First, select the part of the block diagram which will make up the subsystem.
2. Second, select the menu **Edit / Create subsystem**.

To *open* an existing subsystem, double-click the block of the subsystem.

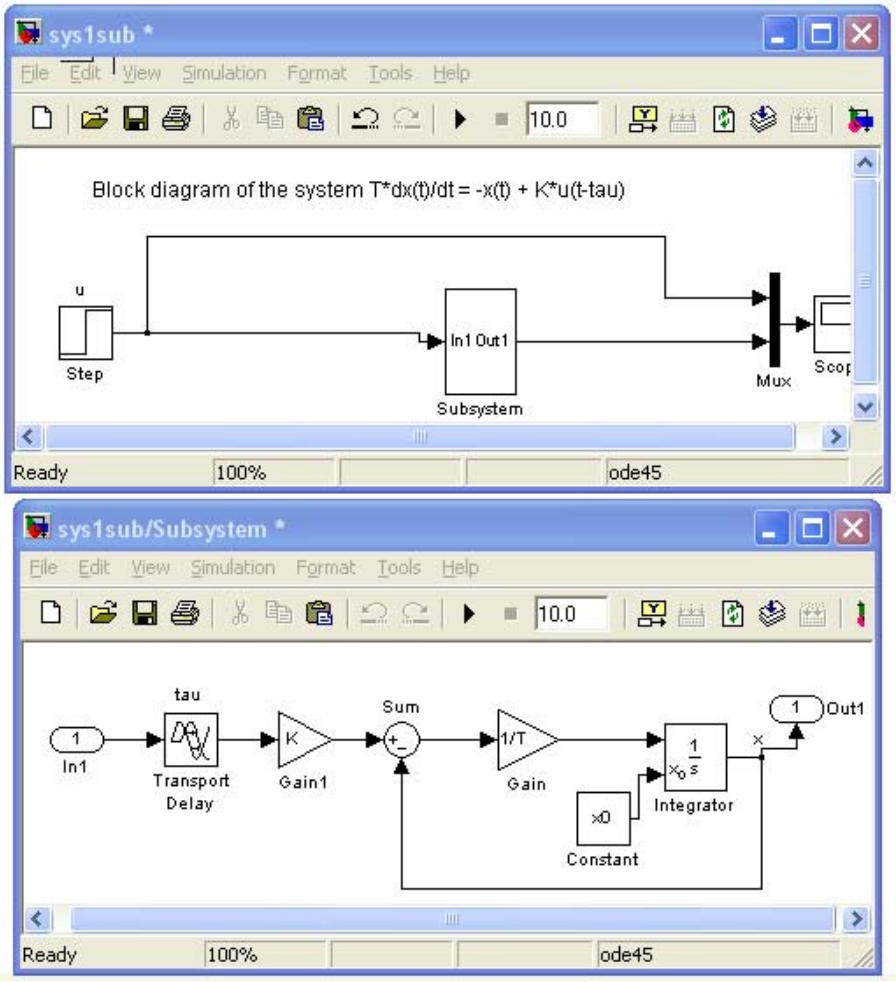

Figure 2.4: A main system and a subsystem. **sys1syb.mdl**.

Note: You can *redo* the creation of a subsystem using the ordinary **Undo** button in the toolbar. However, there is no function for unpacking a subsystem.

Save a copy of **sys1** with name e.g. **sys1sub**. Then, create a subsystem as shown in Figure 2.4. (We will not use this subsystem later in this book.)

Figure 2.5: Coffee time!

Chapter 3

Running a simulation

3.1 How to start a simulation

You can start a simulation in several ways:

- Using the **sim** command, e.g. **sim('sys1')** (assuming that the system or model has name **sys1.mdl**).
- Using the **Start simulation** button in the Simulink window.
- Using the **Simulation / Start** menu.

Probably, you will run a simulation over and over again, with some modifications between the different runs. It may be cumbersome to start repeated simulations from the **Simulation** menu. I suggest that you in stead run the simulations from a (user-written) MATLAB-script! In this case, to re-run a simulation, you simply re-run the (modified) script. And you may even use a for-loop in the script to run a sequence of simulations (more about this a litle later).[1]

Note that it is the block diagram shown on the screen that is simulated. (Strictly, it is not necessary to save the modified block diagram before you simulate it.)

You must set several *simulation parameters* before you run the simulation. Section 3.2 describes the simulation parameter setting in detail. Now, we

[1] Personally, I use scripts for almost every simulation. This is the first option in the list above.

will just run a simulation of **sys1.mdl** with default simulation settings. Figure 2.2 shows the block diagram to be simulated.

The Matlab script **scriptsim.m** shown below defines the model parameters, and runs the simulation with the **sim** command.

Create **scriptsim.m** given below (in the Matlab editor).

scriptsim.m:

```
%Defining model parameters:
tu=5; %Time of step in u
u0=0; %Value of u before the step
u1=2; %Value of u after the step
T=0.5; %Time constant
K=2; %Gain
tau=0.5; %Time delay
x0=4; %Initial value of x

sim('sys1') %Simulates model sys1.mdl
```

Simulate **sys1** by running the script **scriptsim.m**. Then double-click the **Scope** block in the block diagram to see the responses (click the binocular button in the **Scope** window to scale the axes automatically).

The simulated response should be as shown in Figure 3.1.

3.2 How to configure the simulation

3.2.1 Introduction

You must set several *simulation parameters* before you run the simulation. The most important parameters are the following:

- *Start time and stop time*
- *Time step (or step size).* The time step is the resolution along the simulation time axis. In other words, the time step is the time interval between the discrete points of time for which Simulink calculates the response. Figure 3.2 illustrates the time step, which is h in the figure. The time step, and start time and stop time, are just numbers. The time unit is the same as the one you have used in the

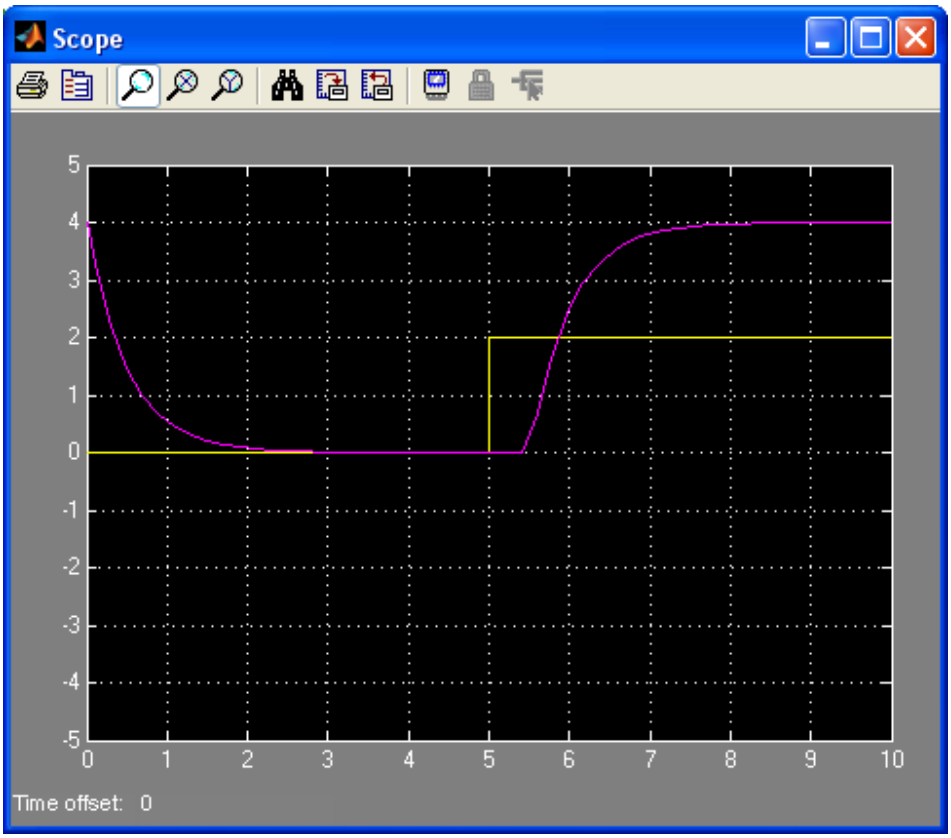

Figure 3.1: Simulated response shown in the Scope in **sys1.mdl**

blocks in the block diagram containing time as unit, e.g. seconds, minutes or hours.[2]

Note that Simulink may use a fraction of a second to run through a simulation covering hours. Thus, Simulink does not run simulations in real time.

- *Simulation algorithm (or method)*

The configuration can be made in two ways, and both ways are described in detail in the following sections:

- In the **Simulation parameters** dialog window, which is opened with the menu selection **Simulation / Configuration Parameters**.

[2] I almost always use second as time unit.

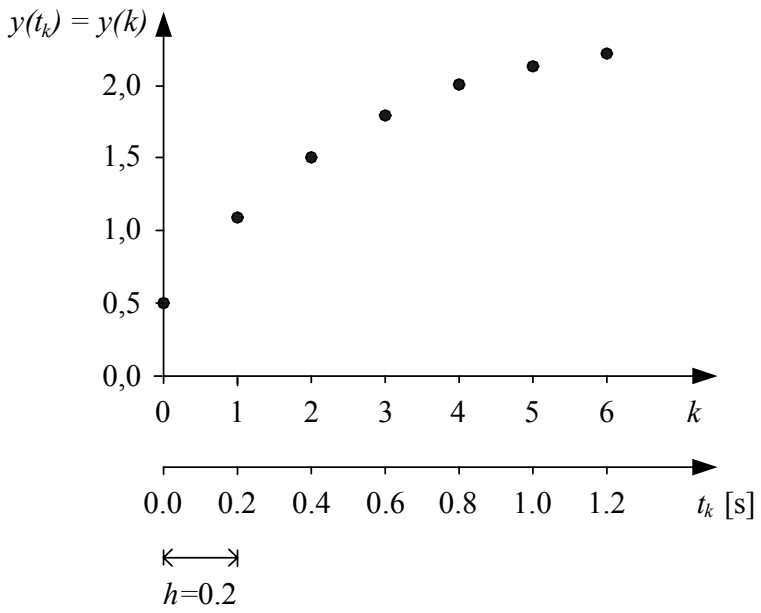

Figure 3.2: Time step is h.

- Using arguments in the **sim** function which can be used in a Matlab script.

Both ways are described in detail in the following sections.

A simulation parameter setting using the **sim** function overrides the setting in the dialog window.

3.2.2 Using the Configuration parameters dialog window

Note: The parameter settings are automatically saved together with the model.

Open the **Configuration Parameters** dialog window with menu **Simulation / Configuration Parameters** in the Simulink window.

Figure 3.3 shows the **Configuration parameters** dialog window with default settings.

Here is a description of the most relevant parameters:

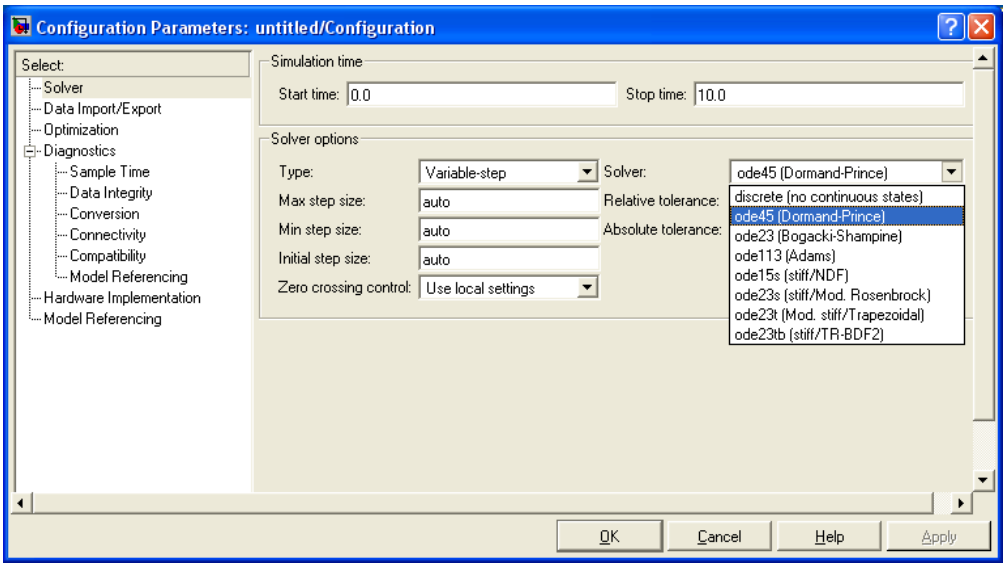

Figure 3.3: The **Configuration parameters** dialog window. **Type** is **Variable step**.

- *Simulation time*: **Start time** is usually set to 0. **Stop time** is the final time of the simulation.
- *Solver options*: **Type** can be either **Variable step** (default) or **Fixed step**. For each of these selection, you can select among different simulation algorithms or methods in the field to the right of the **Type** field, see Figure 3.3:
  - *If you have selected* **Variable step**: Simulink calculates a proper step size as the simulation proceeds. The step size will be time-varying, dependent of the variation of the simulated response. The larger variation, the less step size. Simulink tries different step sizes until a certain tolerance of accuracy is obtained. You can set the tolerance together with the maximum and minimum step size, in the dialog window.

    Solver (i.e. method used to solve the underlying differential and/or difference equations making up the model) can in most cases be set to **ode45 (Dormand-Prince)**.[3]

    If the system is *stiff*, that is, parts of the system has much faster dynamics (e.g. shorter time constant) than other parts, you should select a method which is developed particularly for stiff

[3] ode means ordinary differential equation.

systems. Available methods are indicated by **stiff** in the **Solver** list.

– *If you have selected* **Fixed step**: Figure 3.4 shows the dialog window according to this selection. In this case Simulink uses a

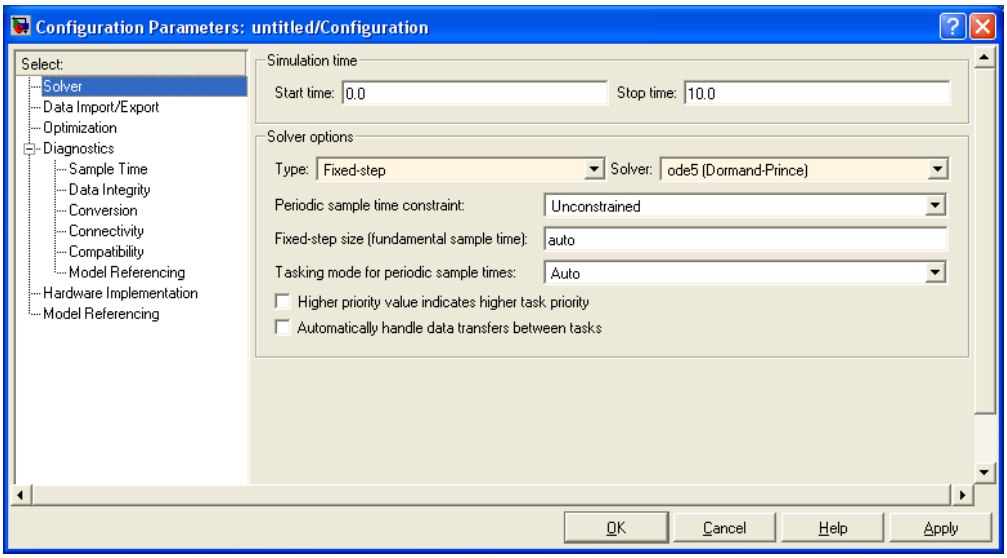

Figure 3.4: The **Configuration parameters** dialog window. **Type** is **Fixed step**.

fixed step size during the entire simulation.

If the step size is too large the simulation will become inaccurate, i.e. there will be a substantial difference between the simulated response and the true response. You can even get an unstable simulation! This will soon be demonstrated.

3.2.3 Guidelines for selecting solver method and time step

My general advice is that you use *fixed step size* in stead of variable step size. Among the various fixed step methods, the **ode 5 (Dormand-Prince)** method can be used as the default method. Using a fixed step size simplifies postprocessing the data, e.g. signal filtering, signal analysis, and data presentation.

How do you select a proper fixed time step size? Either you can let Simulink select a proper step size, or you can set it manually, cf. Figure 3.4. If you do it manually you can proceed as follows: *Set the step size equal to 1/1000 of the simulation time range*, which is the time from the

start time to the stop time. Then simulate. If you have any reason to increase the time step size, increase it by a factor of 5. (One reason may be that the amount of data becomes too large.) If the response depends on the step size, reduce it. If you do not have any reason to increase the step size, decrease it by a factor of 5, and simulate. If the response does not depend on the step size, keep the previous step size.

As an alternative to just trying a time step equal to 1/1000 of the simulation time range, you can try setting it equal to a fraction of the smallest of the time constants and the time delays of the system. This fraction may be 1/10. (If your model is a state space model, you can calculate the smallest time constant as the absolute value of the inverse of the largest eigenvalue of the system.)

Now, let us play a little with the solver method and the time step!

Let us apply a *fixed step size*:

Set **Type** to **Fixed step**. Set **Fixed-step size** to 0.01. Set Solver to **ode 5**. Then simulate **sys1** (by running **scriptsim.m**).

You should observe a simulated response as shown in Figure 3.5. By comparing with the simulation shown in Figure 3.1 where a variable step method was used, you can see that the simulation with the fixed step size of 0.01s is somewhat more accurate. *A conclusion is that a variable step size method with default settings may not simulate as accurately as a fixed-step method having a sufficiently small step size.*

Let us *increase the step size*:

Set **Type** to **Fixed step**. Set **Fixed-step size** to 1. Set Solver to **ode 5**. Then simulate **sys1** (by running **scriptsim.m**).

You should observe a simulated response as shown in Figure 3.6. The simulated response now differs clearly from the more accurate simulation shown in Figure 3.5. The difference is due to the relatively large step size. *A conclusion is that it is important to select a small step size, otherwise the simulatioin may be inaccurate.*

Finally, let us *change to a (simpler) solver method*:

Set **Type** to **Fixed step**. Set **Fixed-step size** to 1. Set Solver to **Euler** (which is the simplest and most innaccurate simulation method). Then simulate **sys1** (by running **scriptsim.m**).

You should observe a simulated response as shown in Figure 3.7. The

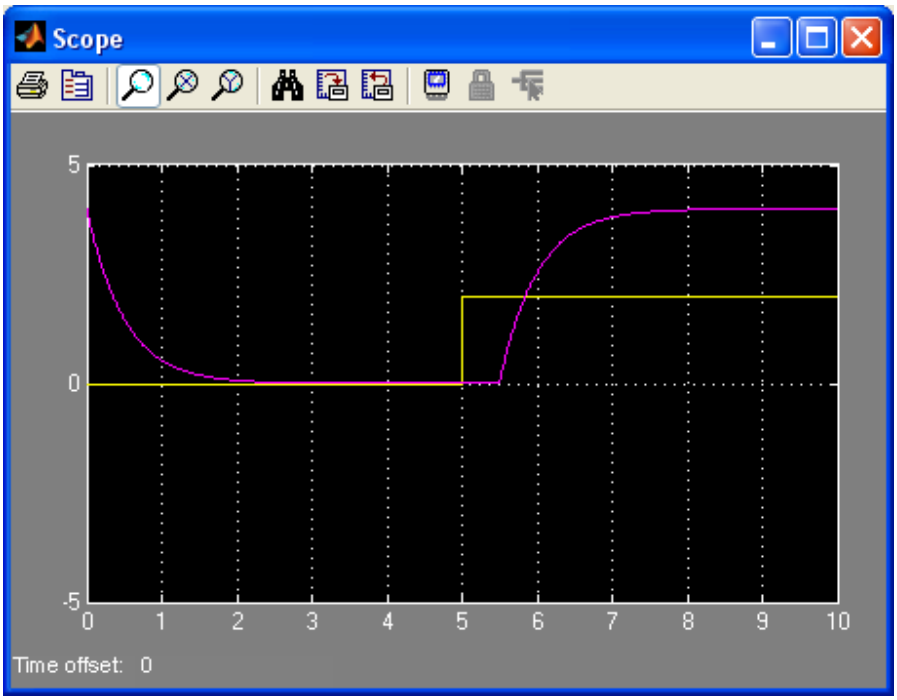

Figure 3.5: Simulation of **sys1.mdl** with fixed step size of **0.01s** and solver **ode5**.

simulation is now unstable! *A conclusion is that a simulation can become unstable if the step size is too large or if an improper solver method is used.*

3.2.4 Setting simulation parameters with the sim function

As an alternative to setting simulation parameters in the **Configuration parameters** dialog window, cf. Figure 3.3, you can set the parameters by defining them as arguments to the **sim** function which you use in a Matlab script. Such parameter settings overrides settings in the dialog window.

There are many parameters that can be used in the **sim** function. These parameters can be collected as one argument using the simset function, and then you use this one argument as an argument to the **sim** function.[4]

We will now simulate **sys1.mdl** using the **sim** function.

Create **scriptsimparam.m** given below (in the Matlab editor).

[4] **help sim** and **help simset** provides help.

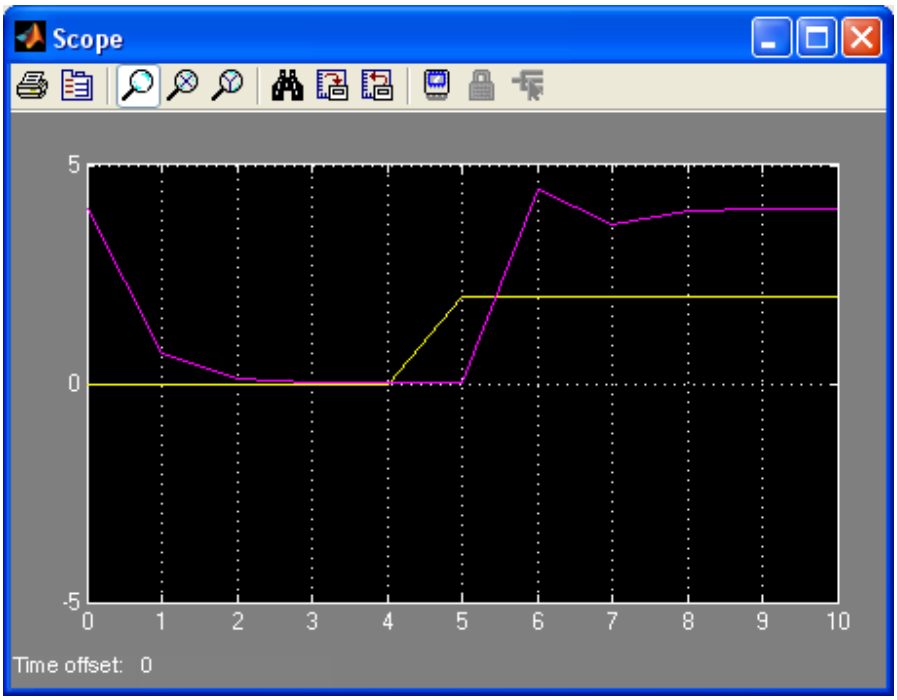

Figure 3.6: Simulation of **sys1.mdl** with fixed step size of **1s** and solver **ode5**.

scriptsimparam.m

```
%Defining model parameters:
tu=5; %Time of step in u
u0=0; %Value of u before the step
u1=2; %Value of u after the step
T=0.5; %Time constant
K=2; %Gain
tau=0.5; %Time delay
x0=4; %Initial value of x


%Setting simulation parameters:
timestep=0.01;
options=simset('solver','ode5','fixedstep',timestep);
tstop=10;

%Simulates sys1.mdl:
sim('sys1',tstop,options)
```

A few comments to the script: In the **simset** function the string **'solver'**

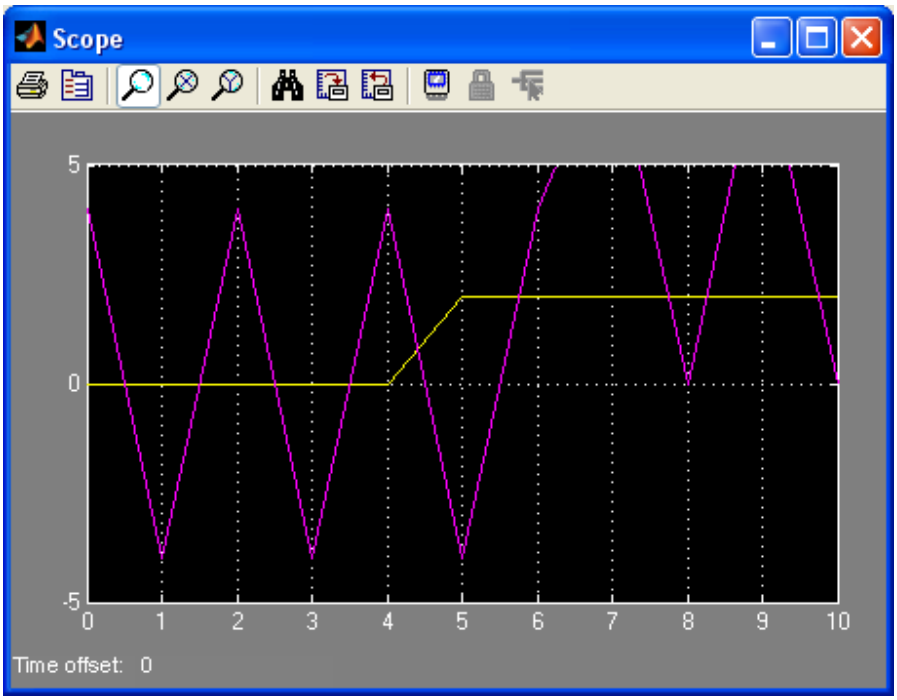

Figure 3.7: Simulation of **sys1.mdl** with fixed step size of **1s** and solver **Euler**.

represents a *property*, and **'ode5'** is the corresponding *property value*. Furthermore, **'fixedstep'** is a property having value **timestep** which is given a numerical value in the script.

Run **scriptsimparam.m**.

The response is identical to the response shown in Figure 3.5.

3.3 How to save simulated data as Matlab variables

Sometimes it is useful to save simulated data (responses) as variables in the Matlab workspace. The purpose may be plotting using the **plot** function, some kind of signal analysis, or saving data to a file. Saving data as Matlab variables can be done in several ways:

- Using **To Workspace** blocks in the block diagram

- Via the **Data import/export** menu option in the **Configuration parameters** dialog window, cf. Figure 3.3
- Via the **Parameters** button in a **Scope** block in the block diagram

I prefer using **To Workspace** blocks because then the block diagram shows (explicitly) which signals are saved as Matlab variables.

Let us try:

Modify **sys1.mdl** as described below, and save it as **sys1workspace.mdl**.

- Include a **Clock** block (from the **Source** library) to generate the simulation time, t.
- Add three **To Workspace** blocks, and connect them to the t, u, and x signals.
- In the parameter dialog window of each of the **To Workspace** blocks: Type t, u, and x, respectively, as **Variable name**.
- In the same parameter dialog window, set **Save format** to **Array**.

The resulting block diagram should be similar to **sys1workspace.mdl** shown in Figure 3.8.

Create the script **scriptsimworkspace.m** shown below. (It can be generated by modifying **scriptsimparam.m**.)

The script simulates the system **sys1workspace.mdl**. The script demonstrates using simulated responses saved as Matlab variables after the simulation has finished. In this script responses are plotted using the **plot** function.

scriptsimworkspace.m

```
%Defining model parameters:
tu=5; %Time of step in u
u0=0; %Value of u before the step
u1=2; %Value of u after the step
T=0.5; %Time constant
K=2; %Gain
tau=0.5; %Time delay
x0=4; %Initial value of x
```

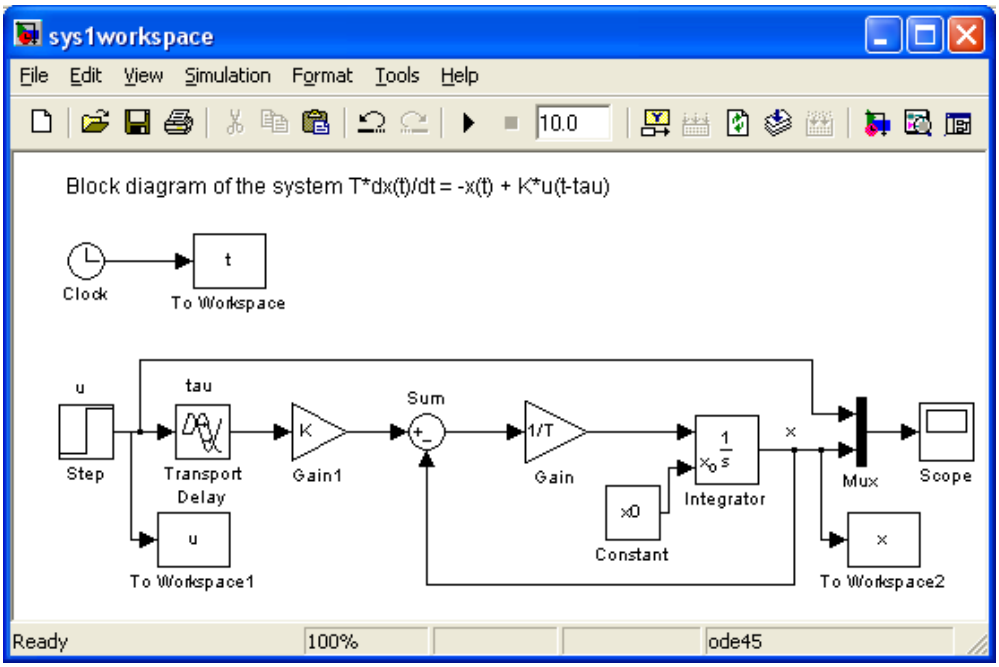

Figure 3.8: **sys1workspace.mdl** containing **To Workspace** blocks

```
%Setting simulation parameters:
timestep=0.01;
options=simset('solver','ode5','fixedstep',timestep);
tstop=10;

%Simulates sys1workspace.mdl:
sim('sys1workspace',tstop,options)

%Postprocessing simulated responses
%(here only plotting):
figure(1)
plot(t,u,'--',t,x,'--')
ylabel('u (dashed) and x')
xlabel('t [sec]')
```

Run the simulation by running **scriptsimworkspace.m**.

After the script has been run (i.e. the simulation has been finished) **u** and **x** are plotted as functions of time **t** as shown in Figure 3.9.

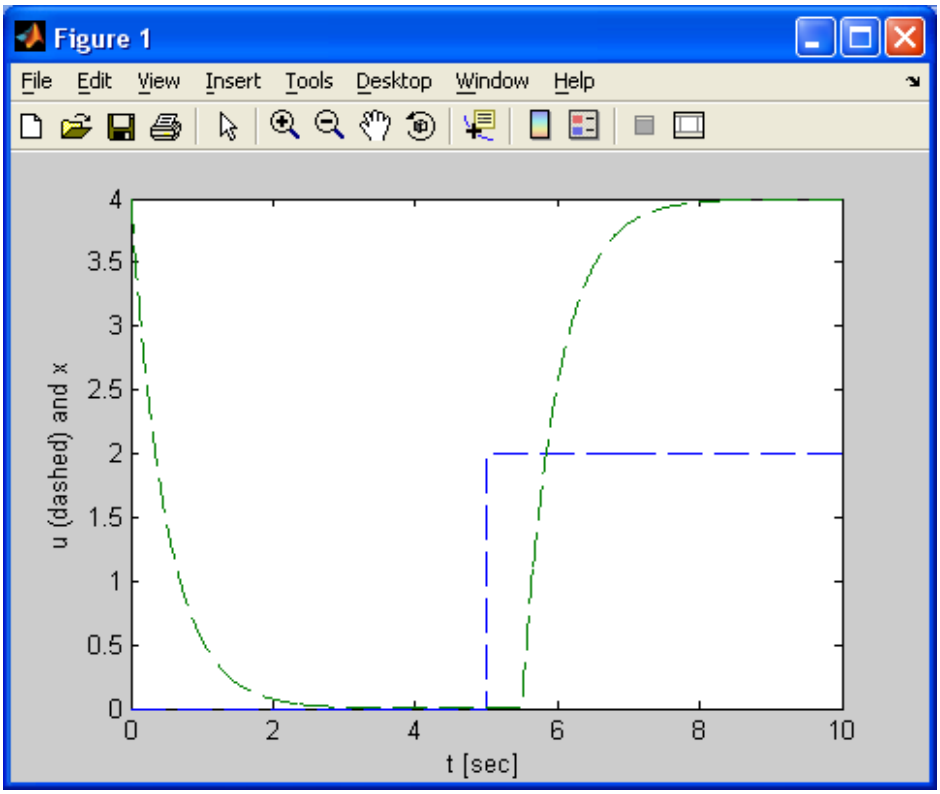

Figure 3.9: Plot generated by the **plot** function in **simscriptworkspace.m**

3.4 Running a sequence of simulations

May be you want to run a sequence of simulations with some parameter adjustment before each simulation? Such a sequence can be realized using a **for loop** in a Matlab script. As an example we will simulate our system for four different values of the time constant **T**: 0.25, 0.5, 1, and 2.

Create the script shown below, and save it as **scriptsimsequence.m**.

scriptsimsequence.m

```
%Defining model parameters:
tu=5; %Time of step in u
u0=0; %Value of u before the step
u1=2; %Value of u after the step
T=0.5; %Time constant
K=2; %Gain
```

```
tau=0.5; %Time delay
x0=4; %Initial value of x


T_array=[0.25 0.5 1 2]; %Four T values.

%Setting simulation parameters:
timestep=0.01;
options=simset('solver','ode5','fixedstep',timestep);
tstop=10;

%Running a sequence of simulations:
for k=1:length(T_array)
T=T_array(k); %A new T value is fetched from the array.
sim('sys1workspace',tstop,options)
%Saves responses in x as columns in x_matrix:
x_matrix(:,k)=x;
end %for

%Plots x_matrix and u as functions of t:
figure(1);
plot(t,x_matrix,t,u)
title('u (dashed) and x for T=0.25, 0.5, 1 og 2.')
xlabel('t [sec]')
```

Run **scriptsimsequence.m**.

The set of simulated responses are plotted in one plot, see Figure 3.10.

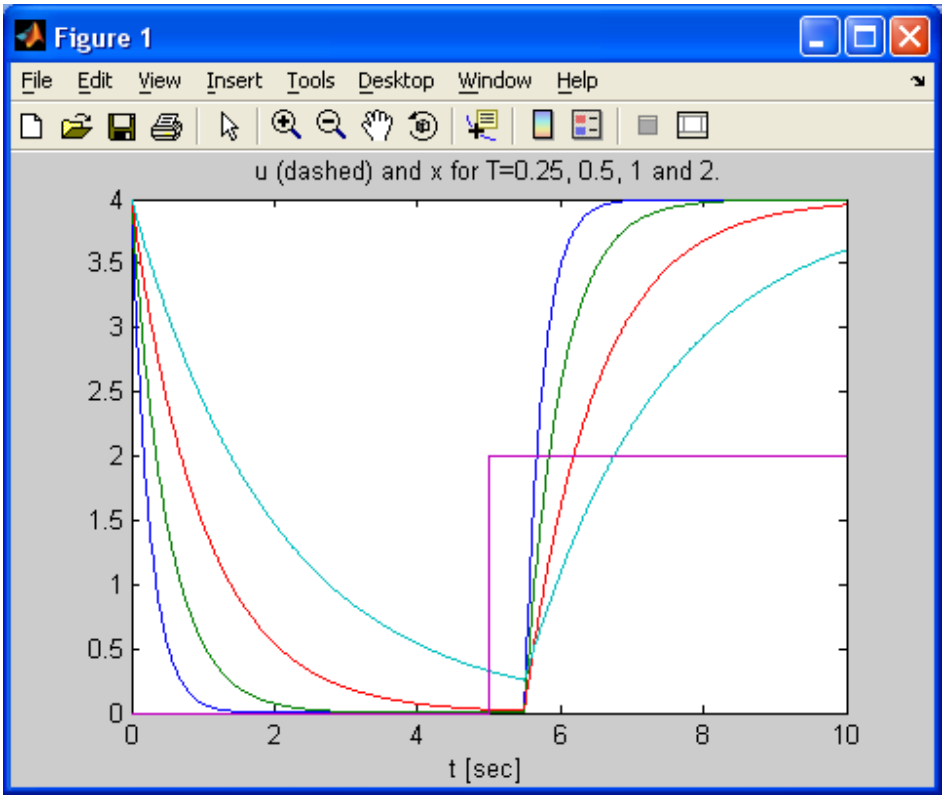

Figure 3.10: A set of responses generated by **scriptsimsequence.m**

Figure 3.11: A break?

Chapter 4

Various model types in block diagrams

4.1 Linear models

You can use various types of linear mathematical models in Simulink: *Transfer functions* — both s-transfer functions (continuous-time) and z-transfer functions (diskcrete-time), and *state-space models* — which are differential equations for continuous-time systems and difference equations for discrete-time systems. If you have installed the Control System Toolbox, you can use *LTI models* (Linear Time Invariant) defined using functions in the Control System Toolbox.

A block diagram can contain models of different types, that is, Simulink supports *hybrid* models.

Let us look at one compound example which demonstrates how to use the following model types:

- s-transfer function
- z-transfer function
- Continuous-time state-space model
- Discrete-time state-space model
- LTI-model, here an s-transfer function

The models are as follows:

- **s-transfer function:**

$$H_c(s) = \frac{0.5s}{1s^2 + 2s + 4} = \frac{b_1 s + b_0 s^0}{a_2 s^2 + a_1 s^1 + a_0 s^0} \tag{4.1}$$

Thus,

$$b_1 = 0.5;\ b_0 = 0;\ a_2 = 1;\ a_1 = 2;\ a_0 = 4; \tag{4.2}$$

- **z-transfer function:**

$$H_d(z) = \frac{1z}{5z^2 + 3z + 1} = \frac{d_1 z + d_0 z^0}{c_2 z^2 + c_1 z^1 + c_0 z^0} \tag{4.3}$$

Thus,

$$d_1 = 1;\ d_0 = 0;\ c_2 = 5;\ c_1 = 3;\ c_0 = 1; \tag{4.4}$$

The sampling time of this model is assumed to be

$$h_1 = 0.1\text{s} \tag{4.5}$$

- **Continuous-time state-space model:**

$$\dot{x}_c = A_c x_c + B_c u_c \tag{4.6}$$

$$y_c = C_c x_c + D_c u_c \tag{4.7}$$

where

$$A_c = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix} \tag{4.8}$$

$$B_c = \begin{bmatrix} 0 \\ 4 \end{bmatrix} \tag{4.9}$$

$$C_c = \begin{bmatrix} 1 & 0 \end{bmatrix} \tag{4.10}$$

$$D_c = [0] \tag{4.11}$$

- **Discrete-time state-space model:**

$$x_d(n+1) = A_d x_d(n) + B_d u_d(t) \tag{4.12}$$

$$y_d(n) = C_d x_d(n) + D_d u_d(n) \tag{4.13}$$

where

$$A_d = \begin{bmatrix} 1 & 0.2 \\ -0.3 & -0.8 \end{bmatrix} \tag{4.14}$$

$$B_d = \begin{bmatrix} 0 \\ 0.5 \end{bmatrix} \tag{4.15}$$

$$C_d = \begin{bmatrix} 1 & 0 \end{bmatrix} \tag{4.16}$$

$$D_d = [0] \tag{4.17}$$

The sampling time of this model is assumed to be

$$h_2 = 0.2\text{s} \tag{4.18}$$

- **LTI-model:** It is the same as H_c given by (4.1).

The input to each of these models will be a unit step (from 0 til 1) at time $t = 0$. The responses are plotted using **Scope** blocks.

Create the Matlab-script **scriptsimsys2.m** shown below which defines the models to be simulated, and also runs the simulation.

scriptsimsys2.m

```
%Defining model parameters:
tu=0; %Time of step in u
u0=0; %Value of u before the step
u1=1; %Value of u after the step

%s-transfer function, Hc:
a2=1; a1=2; a0=4; b1=0.5; b0=0;
num_Hc=[b1,b0];
den_Hc=[a2,a1,a0];

%z-transfer function, Hd:
c2=5; c1=3; c0=1; d1=1; d0=0;
num_Hd=[d1,d0];
den_Hd=[c2,c1,c0];
h1=0.1; %Sampling time

%Continuous-time state-space model
Ac=[0,1;-2,-3]; Bc=[0;4]; Cc=[1,0]; Dc=[0];
xc0=[1,2]; %Initial state

%Discrete-time state-space model
Ad=[1,0.2;-0.3,-0.8]; Bd=[0;0.5]; Cd=[1,0]; Dd=[0];
xd0=[3,4]; %Initial state
h2=0.2; %Sampling time

%LTI model (same as Hc defined above):
Hc=tf(num_Hc,den_Hc);

%Setting simulation parameters:
timestep=0.01;
options=simset('solver','ode5','fixedstep',timestep);
tstop=10;

%Simulates sys2.mdl:
sim('sys2',tstop,options)
```

Figure 4.1 shows the Simulink block diagram containing the models, and the subsequent figures shows the parameter dialog windows for each of the model blocks. Continuous-time model blocks (for transfer functions and state-space models) are available in the **Continuous** library of Simulink, and discrete-time model blocks are in the **Discrete** library.

Create the block diagram **sys2.m** shown in Figure 4.1. The parameter windows of each of the model blocks are shown in the subsequent figures. The **Step** block should be configured with tu as **Step time**, u0 as **Initial value**, and u1 as **Final value**.

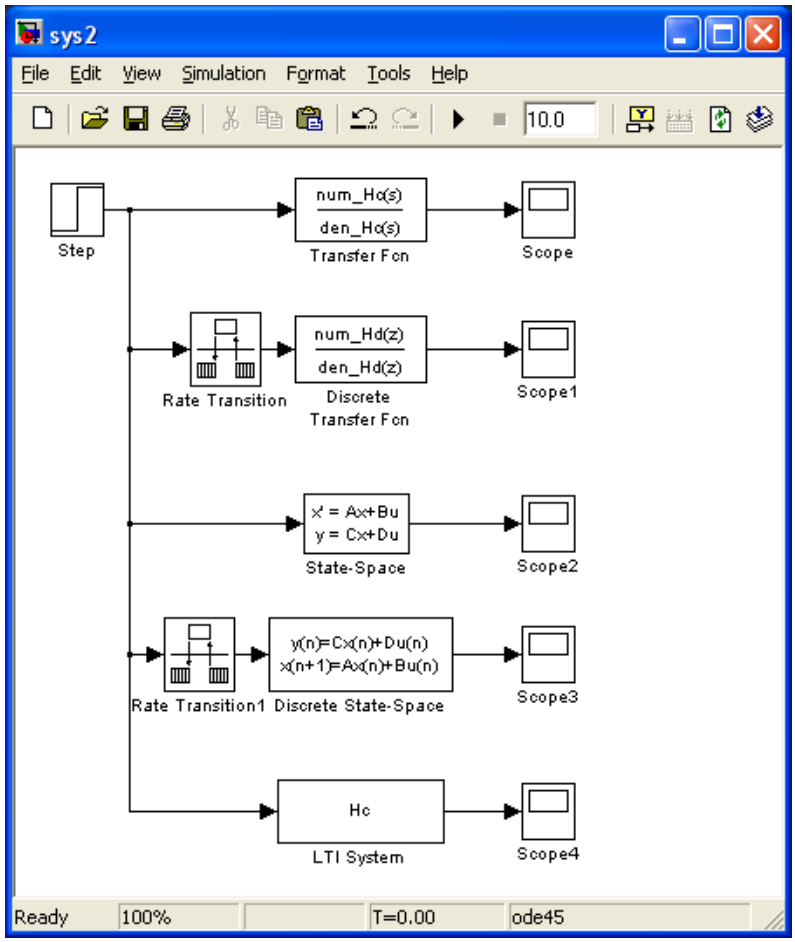

Figure 4.1: The block diagram **sys2.mdl** containing different types of models

Note: The LTI block will cause the error message shown in Figure 4.2 until the LTI model has been defined (e.g. by running the script).

Here are comments to the script and the block diagram:

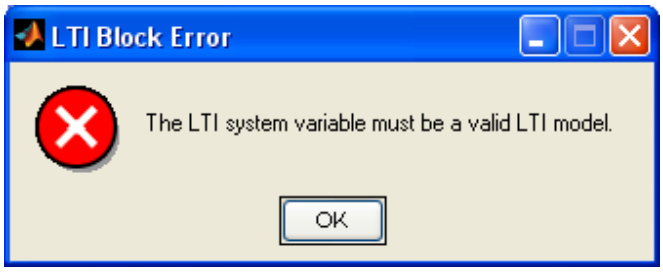

Figure 4.2: The LTI block will cause this error message until the LTI model has been defined.

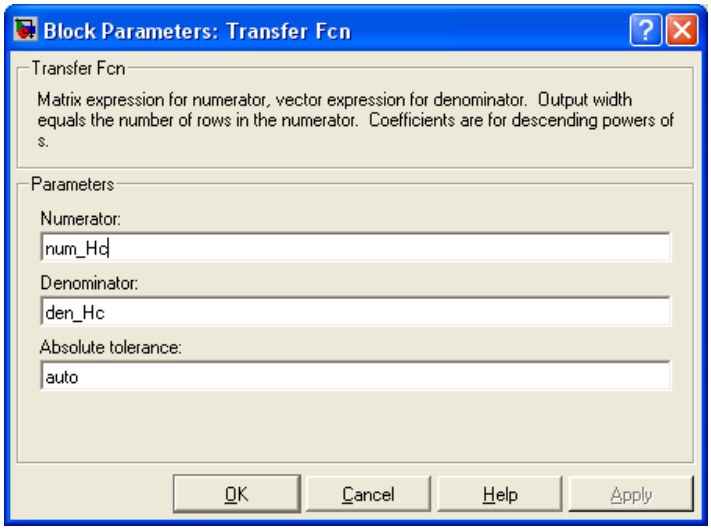

Figure 4.3: Parameter dialog window of the s-transfer function block (Transfer Fcn)

- *Transfer functions* are represented using two Matlab arrays representing the coefficients of the numerator and the denominator polynomial of s for continuous-time transfer functions (or z for discrete-time transfer functions). The order of the coeffients (from left to right in the array) corresponds to a *descending order* of the exponentials of s or z.

  In the script the s-transfer function is defined with the two arrays **num_Hc** and **den_Hc**, and these arrays are used in the corresponding **Transfer fcn** block, see Figure 4.3.

  The z-transfer function is defined with the two arrays **num_Hd** and **den_Hd**, and these arrays are used in the corresponding **Discrete Transfer fcn** block, see Figure 4.4. Note that the sampling time is

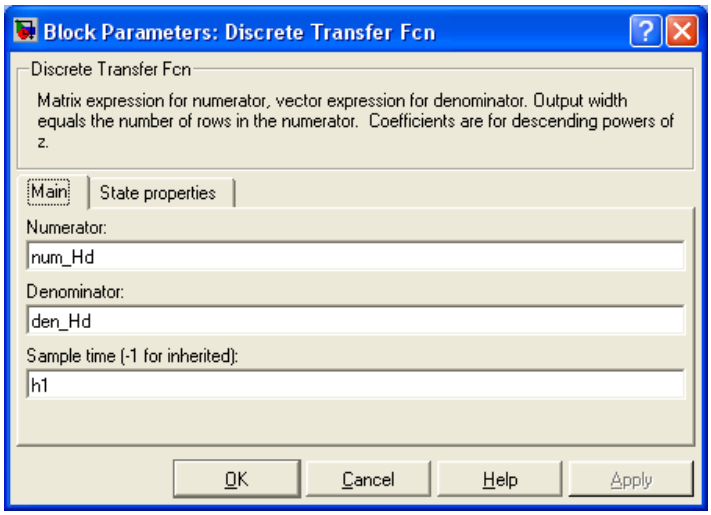

Figure 4.4: Parameter dialog window of the z-transfer function block (Discrete Transfer Fcn)

defined with **h1**. The sampling time need is generally not the same as the time step of the simulator!

A **Rate Transition** block is used to define a transition of the sampling interval between the **Step** block, which runs with a sampling interval equal to the simulation time step which is **timestep** = 0.01s in this example, and the sampling interval of the **Discrete Transfer fcn** block which runs with sampling interval **h1**.

- *State-space models* are represented with their system matrices.

  In the script the continuous-time state space model is defined with the matrices **Ac**, **Bc**, **Cc**, and **Dc**, which are used in the **State Space** block, see Figure 4.5.

  The discrete-time state space model is defined with the matrices **Ad**, **Bd**, **Cd**, and **Dd**, which are used in the **Discrete State-space** block, see Figure 4.6. The sampling time is defined with **h1**.

  A **Rate Transition** block is used to define a transition of the sampling interval between the **Step** block and the **Discrete State-space** block.

- *The LTI model*, **Hc**, is created using the **tf** function in Control System Toolbox. **Hc** is used in the **LTI** block, see Figure 4.7.

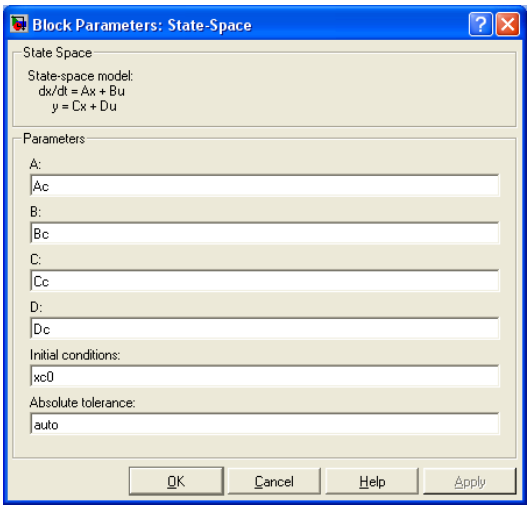

Figure 4.5: Parameter dialog window of the continuous-time state-space model block (State-Space)

4.2 PID-controllers

The **SIMULINK Extras / Additional Linear** library contains two PID-controllers:

- **PID Controller**, which is an ideal PID-controller which should not be used alone, but in series with a low-pass filter (a first order transfer function). The reason for this is explained in most control engineering text-books.
- **PID Controller with Approximative Derivative**, which may be used in most cases because it contains a necessary lowpass filter in the D-term. Figure 4.9 shows this PID block and its parameter dialog window.

Be aware of the parameterization of the PID-controllers! Assume the PID controller transfer function is written as

$$H(s) = \frac{u(s)}{e(s)} = K + \frac{K_p}{T_i s} + \frac{K_p T_d s}{T_f s + 1} \tag{4.19}$$

where K_p is the gain, T_i is the integral time, T_d is the derivative time, and T_f is the filter time constant. Then the parameters used in the dialog window shown in Figure 4.9 are as follows:

$$\mathrm{P} = K_p \tag{4.20}$$

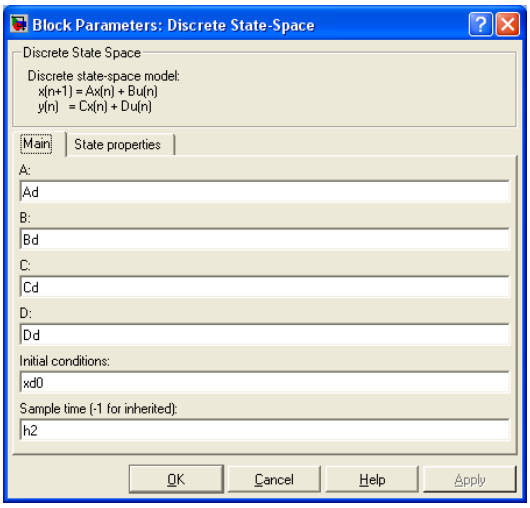

Figure 4.6: Parameter dialog window of the discrete-time state-space model block (Discrete State-Space)

$$\mathrm{I} = \frac{K_p}{T_i} \tag{4.21}$$

$$\mathrm{D} = K_p T_d \tag{4.22}$$

$$\mathrm{N} = \frac{1}{T_f} \tag{4.23}$$

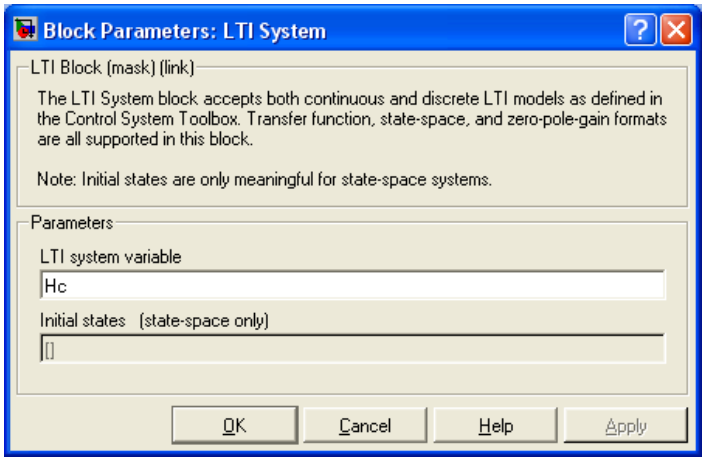

Figure 4.7: Parameter dialog window of the **LTI** block

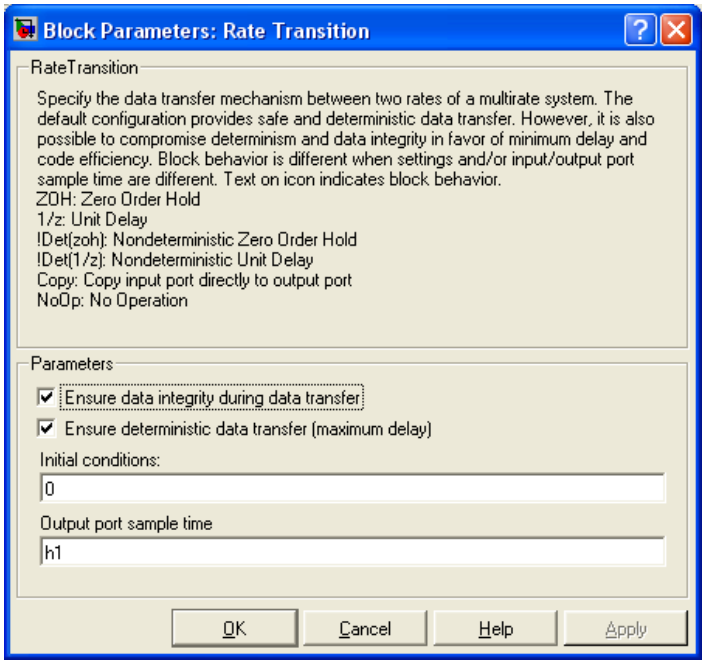

Figure 4.8: Parameter dialog window of the **Rate Transition** block

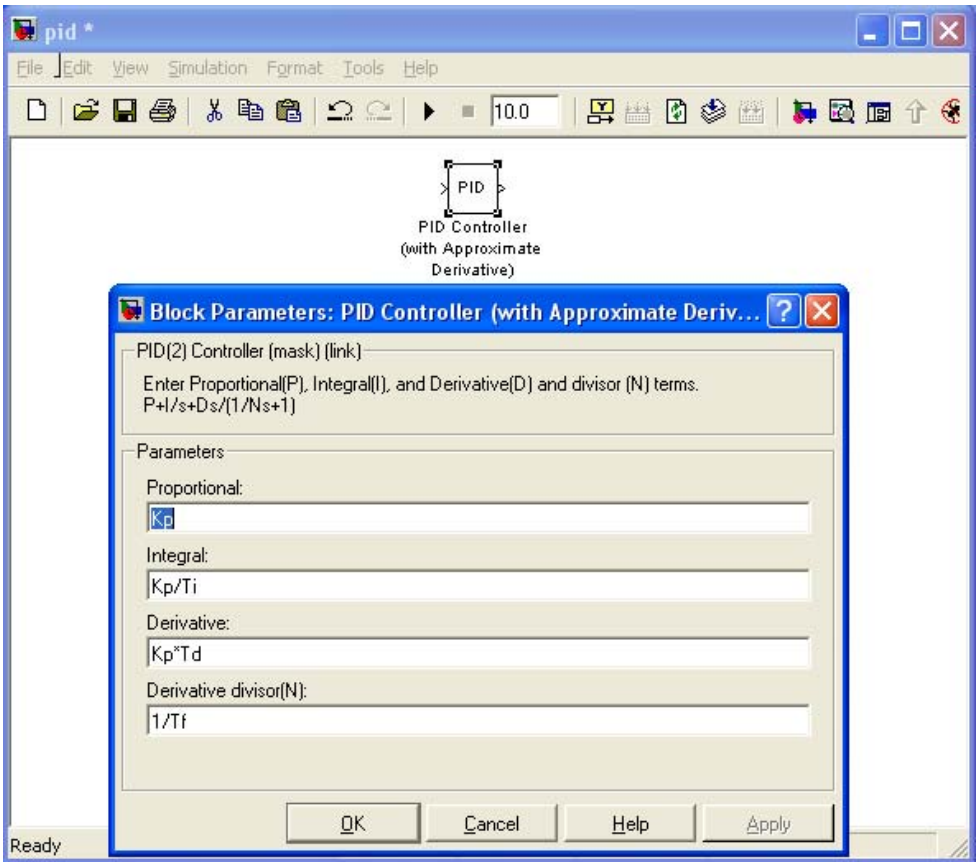

Figure 4.9: The **PID Controller with Approximative Derivative** block and its parameter dialog window

Figure 4.10: You deserve this!

Bibliography

[1] F. Haugen: **Dynamic Systems – modeling, analysis, and simulation**. ISBN 82-519-1926-6. Tapir Academic Publisher, Norway. 2004.