TSFS06 LAB EXERCISE 3
Diagnosis of nonlinear systems

April 9, 2018

1 OBJECTIVES

The objective with this lab exercise is to illustrate diagnosis methods for a non-linear system. A small, but still complicated enough to include many topics covered in the course, is a system of two coupled water tanks. A basic physical model is easily derived, and then a diagnosis system based on this model is to be designed and evaluated. Faults considered in the process are leakages, actuator, and sensor faults.

2 REQUIREMENTS

A written report with a clear presentation of how a diagnosis system is designed, how it is evaluated, and how it fulfills specified performance requirements should be submitted through email. This memo includes a few short leading questions that should be used as help in the design process, the report should not be organized as a series of replies to these questions.

There are two options; a first where a reduced set of faults are considered and a second where a larger set of faults should be considered. The second is intended for those who want a more challenging problem. The reduced version is sufficient to pass the examination. Precise requirements are described in Section 7.

The report is expected to be orderly, coherent, and well-written describing your work. The text should be independent, meaning that it should be readable without any other documents, including this memo. It means that used notation should be described/defined even if they coincide with the course literature. Figures shall always be commented, and clearly stated what a reader should see in the particular figure. Do not include figure with only a comment "and this is the result". Well thought-out headings are important. Common, but not necessary, components of a well written report are an introduction, a number of section describing different parts of the work, a result section, and a conclusion and summary.

3 IMPLEMENTATION

The files needed for the implementation can be downloaded from the course web page and includes the following files:

1. `TSFS06Lab3.mdl`, Simulink model of the water tank process.

2. `lab3init.mat`, Matlab files with the parameter values needed to run the Simulink model.
3. Skeleton file, `labbskal.m`, for initiation of variables needed in simulation. In beginning of this file there is a detailed description of all variables used in the simulation, see Appendix A.

   Important: It is strongly recommended that all Matlab commands used in your solution is written into this file. then you can easily reproduce, repeat, and maintain your solution.
4. Function `decisioncalc` is used to perform single fault isolation according to Chapter **??** in the course literature.
5. The function `tanklinj.m` linearizes the upper tank in an operating point given by the user, `watertank.m` is an S-function for simulating the water tank system with given fault modes, and function `obsgain.m` computes the observer gain given chosen pole positions.
6. Templates for an observer and a dynamic residual generator based on consistency relations is shown in `obs1.m` and `consrel1.m`. See also Appendix B for more information about these files.

3.1 Preparations

To be able to use the scheduled lab-session efficiently, it is recommended that a few of the exercises be done in advance. An exercise marked with **P** is a preparation exercise. It is also recommended to read Section 4.7 in the course book about CUSUM tests.

4 THE WATER TANK PROCESS

The water tank system is illustrated in Figure 1, with an input signal u_{ref} and four outputs $y_1, \ldots, y_4$. Water is pumped into the upper tank (tank 1) and the control signal u is controlled with respect to the water level in the upper tank. Both tanks has level measurements y_1 and y_2, and flow measurements y_3 and y_4. Table 1 summarizes the faults that can be introduced in the system.

Table 1: Possible faults in the water tank system.

F_a	Actuator fault n the pump.
F_{h1}	Fault in sensor 1 measuring the water level h_1 in the upper tank, tank 1.
F_{h2}	Fault in sensor 2 measuring the water level h_2 in the lower tank, tank 2.
F_{f1}	Fault in sensor 3 measuring the flow f_1 between tank 1 and tank 2.
F_{f2}	Fault in sensor 4 measuring the flow f_2 out of tank 2.
F_{l1}	Leakage between tank 1 and sensor 3.
F_{l2}	Leakage between sensor 3 and tank 2.
F_{l3}	Leakage between tank 2 and sensor 4.
F_{c1}	Partial obstruction (clogging) in the pipe between tank 1 and tank 2.
F_{c2}	Partial obstruction (clogging) after tank 2.

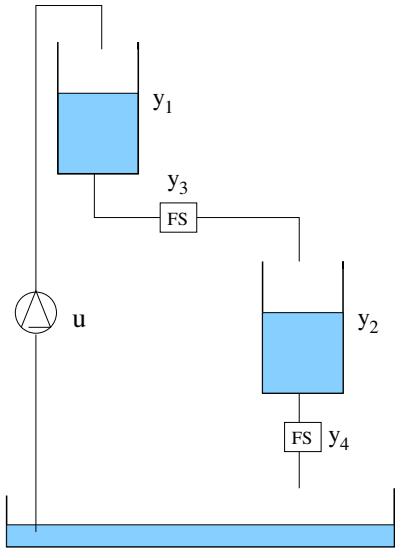

Figure 1: A schematic setup of two coupled water tanks. Each tank also has a level sensor (y_1 and y_2). The flow out of each tank is measured with a flow sensor (FS) (y_3 and y_4).

5 A MODEL OF THE WATER TANK SYSTEM

Let us start with a model of the fault-free case and then introduce models for the different fault cases.

5.1 Fault-free model

Mass balance for a tank means that a change in volume equals in flow - out flow, i.e.,

$$\frac{dV}{dt} = A\frac{dh}{dt} = Q_{in} - Q_{ut} \tag{1}$$

where A is the bottom area of the cylindrical tank, h is the water level, and Q_{in}, Q_{ut} flow in and out of the tank respectively. The flow Q_{ut} out of a tank can be described by Bernoulli's law as

$$Q_{ut} = a\sqrt{2gh} \tag{2}$$

where a is the *efficient* flow area and g is the gravity constant. Equations (1) and (2) then gives that the water level h_1 in the upper tank (tank 1) is described by:

$$\frac{dh_1}{dt} = \frac{1}{A_1}u - \frac{a_1}{A_1}\sqrt{2gh_1} \tag{3}$$

and water level h_2 in the lower tank (tank 2) by:

$$\frac{dh_2}{dt} = \frac{a_1}{A_2}\sqrt{2gh_1} - \frac{a_2}{A_2}\sqrt{2gh_2} \tag{4}$$

Introduce the constants $c_i := \frac{a_i\sqrt{2g}}{A_i}$, and $b_i := \frac{1}{A_i}$, where index i indicates tank i. A model for the tank system in the fault free case is then

$$\frac{dh_1}{dt} = b_1u - c_1\sqrt{h_1} \tag{5a}$$

$$\frac{dh_2}{dt} = \frac{b_2}{b_1}c_1\sqrt{h_1} - c_2\sqrt{h_2} \tag{5b}$$

$$y = \begin{pmatrix} h_1 \\ h_2 \\ \frac{c_1}{b_1}\sqrt{h_1} \\ \frac{c_2}{b_2}\sqrt{h_2} \end{pmatrix} \tag{5c}$$

5.2 Modelling the fault cases

The different fault modes that can be introduced in the system (see Table 1) can be modeled in several different ways, e.g., as signals or deviations in constant parameters. Here, F_a denotes fault mode and f_a denotes the modeled fault, e.g., the fault signal. Here, actuator and sensor faults are modeled using additive signals, and remaining fault modes as constant parameters. With different assumptions on how, e.g., a leak affects the out flow of a tank; there are different fault models. It is here assumed that a leak do not increase outflow of a tank. The resulting model of the water tank system is then (compare with the model in the fault-free case (5))

$$\dot{h}_1 = b_1 u - c_1(1 - f_{c1})\sqrt{h_1} + f_a \tag{6a}$$

$$\dot{h}_2 = \frac{b_2}{b_1} c_1(1 - f_{c1})(1 - f_{l1})(1 - f_{l2})\sqrt{h_1} - c_2(1 - f_{c2})\sqrt{h_2} \tag{6b}$$

$$y = \begin{pmatrix} h_1 + f_{h1} \\ h_2 + f_{h2} \\ \frac{c_1(1-f_{c1})(1-f_{l1})}{b_1}\sqrt{h_1} + f_{f1} \\ \frac{c_2(1-f_{c2})(1-f_{l3})}{b_2}\sqrt{h_2} + f_{f2} \end{pmatrix} \tag{6c}$$

To simplify the presentation, we introduce the new parameters $d_1, \ldots, d_6$ in the model

$$\dot{h}_1 = d_1 u - d_2(1 - f_{c1})\sqrt{h_1} + f_a \tag{7a}$$

$$\dot{h}_2 = d_3(1 - f_{c1})(1 - f_{l1})(1 - f_{l2})\sqrt{h_1} - d_4(1 - f_{c2})\sqrt{h_2} \tag{7b}$$

$$y = \begin{pmatrix} h_1 + f_{h1} \\ h_2 + f_{h2} \\ d_5(1 - f_{c1})(1 - f_{l1})\sqrt{h_1} + f_{f1} \\ d_6(1 - f_{c2})(1 - f_{l3})\sqrt{h_2} + f_{f2} \end{pmatrix} \tag{7c}$$

The model (7) then describes the system, including the different fault modes.

6 SIMULATION MODEL

A simulation model is implemented in Simulink in the file `TSFS06lab3.mdl`. The Simulation model in Figure 2 reads the signal *Uref* from Matlab workspace, a reference signal to the water level controller for the upper tank, and a time vector t. The simulation model produces the following output signals:

tut Time vector for the simulation model.

styr Control signal for the pump.

y Outputs from the water tanks. Includes measurement noise.

res Residual values.

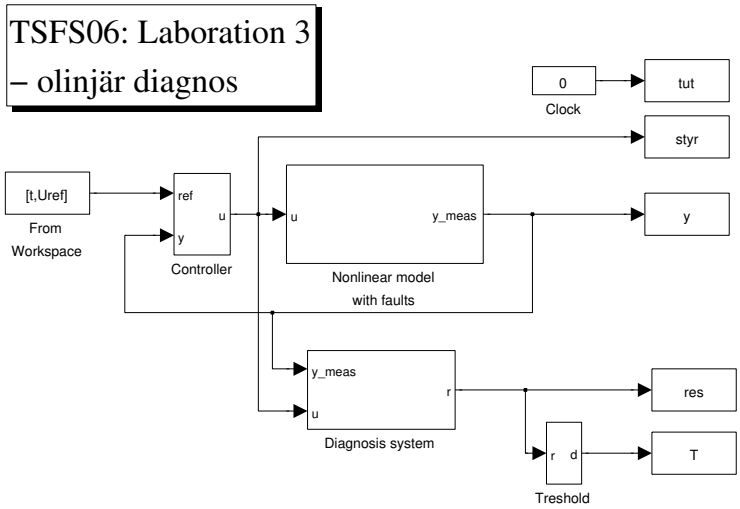

Figure 2: Simulink model TSFS06Lab3.mdl.

T Thresholded residuals, i.e., 0 or 1 dependent on if the residual is above the threshold or not.

f Vector with values of the introduced faults.

The block `'Nonlinear model with faults'` simulates the water tank system with introduced faults and measurement noise. Fault sizes can be modified by double clicking on the block and specify fault sizes in the dialog window shown in Figure 3. One introduced fault is shown in Figure 4. A value 0 corresponds to no fault. The fault is introduced at `'Fault initialization time'` and the parameter `'Fault ramp duration'` determines how fast the fault is introduces, i.e., how long time it takes from fault initiation to maximum value. See Figure 4 for the case where this time is 2 s. Measurement noise can be activated/deactivated with the `'Measurement noise'`.

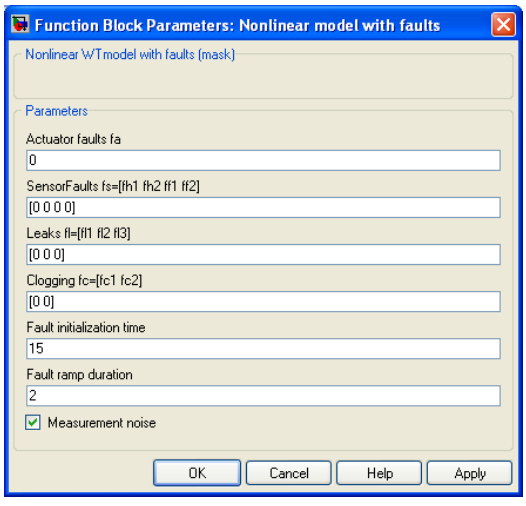

Figure 3: Dialog window for the simulation model where, e.g., the different fault modes can be specified and introduced.

The block `'Controller'` is a pre-calibrated PI-controller. The Simulink block `'Diagnosis system'` contains the residual generators you are designing during the lab exercise.

7 DIAGNOSIS SYSTEM

When designing the diagnosis system, it is required that test quantities/residuals are design corresponding to a specified decision structure. In addition, thresholds has to be selected for each residual. The block `'Diagnosis system'` is implemented in Simulink as seen in Figure 5. To design residuals, there are many approaches to apply; mentioned in the course are, e.g., *non-linear consistency relations* and *non-linear observers*, see Chapter **??** in the course literature. Apply both these methods in your solution, at least one test quantity (in addition to the ones already implemented) of each method shall be part of your solution.

7.1 Definition of requirements

In Section 2 two options were described; one considering a reduced set of faults and one considering all faults. In the full version, design a diagnosis system for all fault modes defined in Table 1 while in the reduced version, only fault modes $F_a, F_{h2}, F_{f1}, F_{l2}, F_{l3}, F_{c1}$ need to be considered. Requirements on detectable fault sizes are given in Table 2. Note that the detection requirements is specified with sensor noise variances $10^{-4}(5, 0.25, 0.5, 0.5)$.

Table 2: Performance requirements on fault sizes.

Type of fault	Requirement	Corresponding fault size
Actuator fault	5% of max value of u	0.5
Sensor fault	Deviation of measured value of 0.4	0.4
Leakage	40% of the flow in fault free mode	0.4
Clogging	40% of flow in fault free mode	0.4

For the set of faults you have chosen it is required that

- all faults shall be detectable by your diagnosis system
- all single fault should be isolated as far as possible. For cases where unique isolation is not possible, convincing argument shall be given in the report.

For every designed residual, the report shall include

- which equations have been used in the derivation
- which fault sensitivity is expected
- the expressions for the residual generators
- a verification of the fault sensitivity with suitable plots

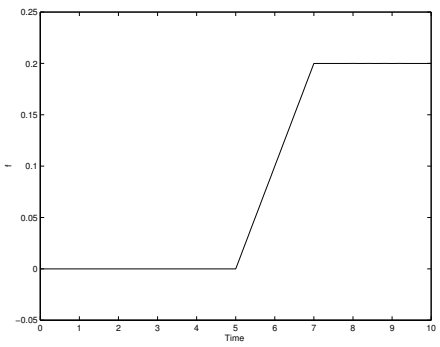

Figure 4: Example of an introduced fault.

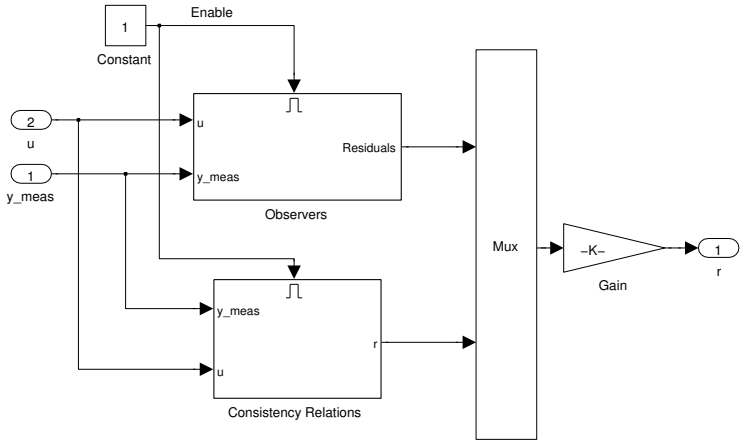

Figure 5: Simulink block Diagnosis System.

In addition, the report shall include an isolability matrix for the model and an isolability matrix for your developed diagnosis system.

7.2 Non-linear observers

Non-linear observers are useful for generating residuals and the basic principle is to estimate a specific output signal and compare with the measurement.

Below is illustrated how an observer that is, e.g., sensitive to a fault in level sensor 1 (f_{h1}) can be designed

$$\dot{\hat{x}}_1 = d_1 u - d_2\sqrt{\hat{x}_1} + K_1(y_1 - \hat{x}_1) \tag{8a}$$
$$r_1 = y_1 - \hat{x}_1 \tag{8b}$$

The observer estimating $\hat{x}_1$ is used in the residual r_1. Te observer feedback gain K_1 should be chosen such that the observer error dynamics is stable. This can, for example, be achieved by linearizing the system and then choosing a conservative K_1 such that disturbances are not amplified. Two Matlab functions are provided to help; the function `'tanklinj'` linearizes the system in a selected operating point and the function `'obsgain'` determines an observer gain based on given pole placement. Section A.1 shows the file `labbskal.m` that shows how a stabilizing K_1 can be determined for the observer (8).

By considering (7) the hypotheses for the observer (8)

$$H_1^0 : \mathbf{F_p} \in \{\mathbf{NF}, \mathbf{F_{h2}}, \mathbf{F_{f1}}, \mathbf{F_{f2}}, \mathbf{F_{l1}}, \mathbf{F_{l2}}, \mathbf{F_{l3}}, \mathbf{F_{c2}}\}$$
$$H_1^1 : \mathbf{F_p} \in \{\mathbf{F_a}, \mathbf{F_{h1}}, \mathbf{F_{c1}}\}$$

See Appendix B for information on how such a residual is implemented in Simulink.

Exercise 1 (P)**.** Derive more observers for the water tank system.

Answer:

Exercise 2. Implement your observers in Simulink in the block `'Observers'`, see Figure 6. The observer based residual (8) is already implemented. Use that implementation as a template for your own observers. See further in Appendix B how to implement them. Only implement one observer in each m-file.

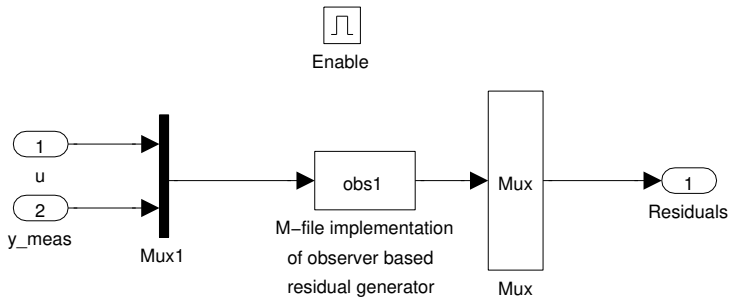

Figure 6: Simulink block Observers.

7.3 Non-linear consistency relations

A non-linear consistency relation g is a relation between an input u and output y (and possibly derivatives in case of a dynamic system) that is fulfilled in the fault free case as

$$g(y, \dot{y}, \ddot{y}, \ldots, u, \dot{u}, \ddot{u}, \ldots) = 0 \tag{9}$$

Exercise 3 (P)**.** Is g a consistency relation if it is fulfilled also when there is a fault?

Answer: .....

One example of a consistency relation for the water tank system is

$$\dot{y}_1 - d_1 u + d_2 \sqrt{y_1} = 0 \tag{10}$$

It is derived from (7a) and (7c) and is based on that the level change in tank 1 is related to the inflow minus the outflow. Determine a residual r_1 from this consistency relation and by considering (7) we can determined the following corresponding hypotheses for r_1

$$H_1^0 : \mathbf{F_p} \in \{\mathbf{NF}, \mathbf{F_{h2}}, \mathbf{F_{f1}}, \mathbf{F_{f2}}, \mathbf{F_{l1}}, \mathbf{F_{l2}}, \mathbf{F_{l3}}, \mathbf{F_{c2}}\}$$
$$H_1^1 : \mathbf{F_p} \in \{\mathbf{F_a}, \mathbf{F_{h1}}, \mathbf{F_{c1}}\}$$

A residual that is computed directly according to the consistency relation, i.e.,

$$r = \dot{y}_1 - d_1 u + d_2 \sqrt{y_1}$$

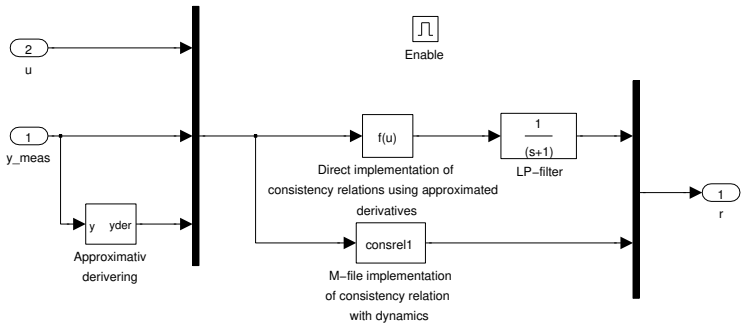

Figure 7: Simulink block Consistency Relations.

is implemented in the block `'Consistency relations'` in Figure 5 and is shown at the top of Figure 7. A problem when using (10) is that it contains the derivative of the measurement signal y_1. A simple way to still use the expression is to approximate the derivative using a filter $\frac{s}{1-s/s_0}$.

In this case it is easy to circumvent the derivative estimation since $\dot{y}_1$ appears linearly in the expression. This means that the linear methodology from Chapter **??** in the course literature to avoid estimating derivatives can be used. Thus, introducing linear residual generator dynamics makes it possible to state the residual in state-space form. Therefore, compute the residual as

$$\dot{r} + \alpha r = \dot{y}_1 - d_1 u + d_2 \sqrt{y_1}$$

which, with state variable $w = r - y_1$, results in

$$\dot{w} = -\alpha(w + y_1) - d_1 u + d_2 \sqrt{y_1} \tag{11a}$$

$$r = w + y_1 \tag{11b}$$

This residual generator is implemented in the lower residual generator block in Figure 7. See further in Appendix B how this implementation is done. for simplicity in debugging, implement only 1 consistency relation in each m-file.

Exercise 4 (P)**.** Derive more consistency relations for the water tank system.

Answer:

Exercise 5. Implement your consistency relations in Simulink. do the design both using derivative estimates and, where so is possible, use the linear methodology as described in (11) to avoid estimating derivatives.

Hint: If derivative estimates are used, it is suitable to also introduce a LP-filtering operation, see Figure 7.

Answer:

7.4 Normalization of test quantities

Exercise 6. Determine thresholds J_i such that, in the fault free case, the following holds:

$$P(|r_i| > J_i) \approx \alpha$$

where $\alpha = 0.1\%$, i.e., α is the probability for false alarm.

Compute thresholds and normalize the residuals such that all thresholds can be set to 1, i.e., $J_i = 1$. The normalization ensures that the predefined thresholding block in the Simulink model can be used as is. The normalization is performed using the variable `Jnorm`, see Figure 5. The variable `Jnorm` is defined in the file `labbskal.m` and observe that the number of residuals determines the number of elements.

7.5 Decision structure

Exercise 7 (P)**.** Which decision structure corresponds to your test quantities? Motivate your answer. You need only consider single faults.

Answer:

Enter your decision matrices in `'labbskal.m'`, see Section A.

Exercise 8 (P)**.** What can be stated about the thresholds if the final diagnosis decision includes most of the fault modes even in faulty cases?

Answer: .....

What can be stated about the thresholds if the diagnosis decision often is the empty set, i.e., no behavioral mode (including **NF**) is deemed consistent with the observed behavior.

Answer: .....

Exercise 9. Investigate by simulation that the diagnosis system behaves as expected and achieves desired detection, isolation, and false alarm performance.

Answer: .....

Exercise 10. Implement a CUSUM test in your diagnosis system in Simulink for, at least, one of your residuals. Compare its detection performance with thresholding the residual output. Are you able to detect and isolate even smaller faults? What about false alarm performance?

Answer: .....

A APPENDIX

A.1 labbskal.m

The file `labbskal.m` is the main skeleton file where you enter suitable code for your design and evaluation of the diagnosis system.

```
%% Labbskal för laboration 3 i TSFS06: Diagnos & Övervakning
clear all

%% ================================================
%  Sätt modellparametrar för simulering
%  ================================================

% Parametrar som används för Simulering av vattentanken
% d1--d6 Parametrar som beskriver dynamiken för vattentankssystemet.
%        Se labbkompendiet för detaljer.
% Uref   Referenssignal till regulator.
% t      Tidsvektor för referenssignal.
% h1Init Initialvärde för vattennivå i tank 1. Default är Uref(1).
% h2Init Initialvärde för vattennivå i tank 2. Beräknas utifrån
%        stationär punkt med avseende på h1Init.
%
load lab3init

h1Init=Uref(1,1);        % Initialnivå i tank 1
h2Init=(d3/d4)^2*h1Init; % Initialnivå i tank 2
d = [d1 d2 d3 d4 d5 d6]; % Spara alla modellparametrar i en vektor

watertankparams.x0 = [h1Init; h2Init]; % Initialnivån i resp. tank
watertankparams.d  = d; % Modellparametrar

% Sätt slumptalsfrö och brusintensiteter för simulering av mätbrus
NoiseSeed = floor(abs(randn(1,4)*100));
NoiseCov  = 1e-4*[5 0.25 0.5 0.5];

%% ================================================
%  Design av residualgenerator 'obs1'
%  via observatörsdesign
%  ================================================

% Linjärisering av vattentankssystemet för h1=4:
Gsys=tanklinj(4, d);
% Anpassa A och C-matrisen så att de passar
A  = Gsys.a(1,1);
C  = Gsys.c(1,1);

P  = [-1]; % Placering av polerna
K1 = obsgain(A,C,P);

% Spara parametrarna som skickas in till observatören i
obs1params.x0 = h1Init; % Initialvärde på observatörens tillstånd
obs1params.K1 = K1;     % Observatörsförstärkningen
obs1params.d  = d;      % Modellparametrar
```

```
%% ================================================
%  Design av residualgenerator consrel1
%  via konsistensrelation med dynamik
%  ================================================

% Sätt parametrar för konsistensrelation
c1params.x0   = -h1Init; % Initialvärde för residualgeneratorns tillstånd
c1params.alfa = 2;       % Placering av polen i -alfa
c1params.d    = d;       % Modellparametrar


%% ================================================
%  Tröskelsättning
%  ================================================

Jnorm=ones(1,3); % Default är alla trösklar satta till 1

%% ================================================
%  Simulera systemet
%  simuleringen kan antingen göras genom att välja menyn
%  Simulation->Start i Simulink fönstret
%  eller exekvera nedanstående rad
%  ================================================

sim('TSFS06Lab3');

%% ================================================
%  Definiera beslutsstrukturen via s0 och s1
%  Felfria fallet NF ska stå först
%  ================================================

% Beslut för residualer under tröskeln
%s0 = ones(3,11);
s0 = ones(3,7); %förenklad variant

% Beslut för residualer över tröskeln
%s1 = zeros(3,11);
s1 = zeros(3,7); %förenklad variant

%% ================================================
%  Beräkna diagnoser under ett enkelfelsantagande
%  ================================================
[S,alarm] = decisioncalc(T,s0,s1);


%% ================================================
%  Plotta resultatet
%  ================================================

% Förslag på plottar
figure(1)
plot( tut, y )
figure(2)
plot( tut, alarm )
figure(3)
plot( tut, res )
```

```
figure(4)
plot( tut,T )

figure(5)
% Kräver att felmoderna är definierade i samma ordning
% i 'S' som i 'name'.
name={'NF', 'Fa', 'Fh2', 'Ff1', 'Fl2', 'Fl3', 'Fc1'};
%name={'NF', 'Fa', 'Fh1', 'Fh2', 'Ff1', 'Ff2', 'Fl1', 'Fl2', 'Fl3',...
%'Fc1', 'Fc2'};

% Plottar diagnosbeslutet för de olika felmoderna enligt S
% och namnger dem efter name.
for n=1:length(name)
  subplot(3,3,n)
  plot(tut,S(:,n))
  title(name{n})
  axis([min(tut) max(tut) -0.1 1.1])
end
```

B IMPLEMENTATION OF RESIDUAL GENERATORS USING S-FUNCTIONS

This section will describe how residual generators with dynamics can be implemented in Simulink using S-functions. Observe that S-functions should *only* be used when the residual generator has dynamics, i.e., have states.

The observer solution (8), illustrated in Figure 6, and the dynamic residual generator (11), illustrated in Figure 7, are both implemented in a similar way using an S-function in Simulink. S-functions is an alternative to "drawing" the computation scheme for the residual generator and instead directly enter the equations. Which solution that is preferred, is up to you. Below are implementations for both residual generators (8) and (11) described in detail.

B.1 obs1.m

By double clicking on the block `obs1` in Figure 6 raises the dialog window in Figure 8. Here, two parameters can be entered; a file name (m-file) and parameters to

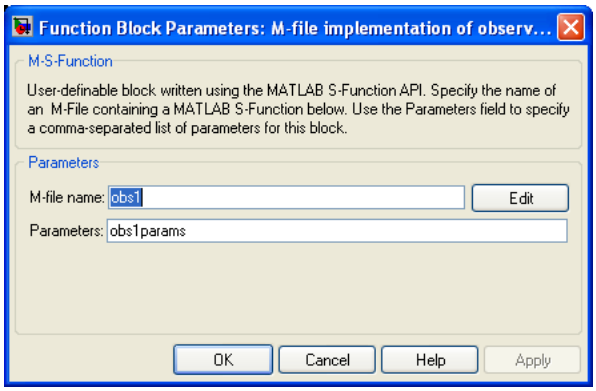

Figure 8: Dialog window for an S-function.

the S-function. In this case, the m-file is called `obs1.m` and is included below. The

parameter in this case is `obs1params` which are set in `labbskal.m`. In this case is `obs1params` a structure with three fields, one initial state x_0 for the observer, the observer gain $K1$, and a vector with the model parameters $d_1, \ldots, d_6$. It is completely free to define your own parameters, just add to the parameter structure as shown in `labbskal.m`.

Here, 4 parts of the m-file need to be written specifically for each residual generator:

1. `numstates`
   Number of states in the residual generator
2. Function `InitConditions`
   Sets the initial value for the states in the observer.
3. Function `Derivative`
   Define the observers dynamic equations, in this case the single equation (8a).
4. Function `Output`
   Define the output equations, in this case equation (8b).

In these functions the parameters will be directly accessible, in this case the variable `obs1params`, using `block.DialogPrm(1).Data`. Thus, the observer gain `K1` equals `block.DialogPrm(1).Data.K1` and corresponding for the initial state `x0` and the vector with model parameters `d`.

The input signals to the observer, i.e., the observations, is accessed by

```
u  = block.InputPort(1).Data(1);
y1 = block.InputPort(1).Data(2);
y2 = block.InputPort(1).Data(3);
y3 = block.InputPort(1).Data(4);
y4 = block.InputPort(1).Data(5);
```

The internal state of the observer, in this case $\hat{x}_1$, is accessed by

```
x1hat = block.ContStates.Data;
```

If you have more than one state, access them by `block.ContStates.Data(1)`, `block.ContStates.Data(2)` and so on.

Below is a full listing of the file `obs1.m`.

```
function obs1(block)

  setup(block);

function setup(block)
%% ================================================================
%  Definiera parametrar för residualgeneratorn
%  ================================================================
  numstates = 1; % Antal kontinuerliga tillstånd i funktionen
%  ================================================================

  numparams = 1;

  %% Register number of dialog parameters
  block.NumDialogPrms = numparams;

  %% Register number of input and output ports
  block.NumInputPorts  = 1;
  block.NumOutputPorts = 1;
```

```
  %% Setup functional port properties to dynamically
  %% inherited.
  block.SetPreCompInpPortInfoToDynamic;
  block.SetPreCompOutPortInfoToDynamic;

  block.InputPort(1).Dimensions        = 5;
  block.InputPort(1).DirectFeedthrough = true;
  block.OutputPort(1).Dimensions       = 1;

  %% Set block sample time to continuous
  block.SampleTimes = [0 0];

  %% Setup Dwork
  block.NumContStates = numstates;

  %% Register methods
  block.RegBlockMethod('InitializeConditions',    @InitConditions);
  block.RegBlockMethod('Outputs',                 @Output);
  block.RegBlockMethod('Derivatives',             @Derivative);

%endfunction

function InitConditions(block)
%% ================================================================
%  Definiera initialtillståndet
%  ================================================================
  x0 = block.DialogPrm(1).Data.x0;
%  ================================================================
  block.ContStates.Data = x0;

%endfunction

%% ================================================================
%  Definiera utsignalsekvationen
%  ================================================================
function Output(block)
  x1hat = block.ContStates.Data;
  y1    = block.InputPort(1).Data(2);

  r     = y1-x1hat;
  block.OutputPort(1).Data = r;

%endfunction

%% ================================================================
%  Definiera de dynamiska ekvationerna
%  ================================================================
function Derivative(block)
K1 = block.DialogPrm(1).Data.K1;
d  = block.DialogPrm(1).Data.d;

u     = block.InputPort(1).Data(1);
y1    = block.InputPort(1).Data(2);
x1hat = block.ContStates.Data;

dx1hat = d(1)*u-d(2)*sqrt(x1hat)+K1*(y1-x1hat);
```

```
block.Derivatives.Data = dx1hat;
```

B.2 consrel1.m

In Figure 7, equation (11) is implemented in the block `consrel1`. This block works in exactly the same was is described above for the observer case in the block `obs1`. Below is the m-file that implements the residual generator (11) and the only difference compared to `obs1.m` is the definitions of functions `InitConditions`, `Derivative` and `Output`.

```
function consrel1(block)

  setup(block);

function setup(block)
%% ================================================================
%  Definiera parametrar för residualgeneratorn
%  ================================================================
  numstates = 1; % Antal kontinuerliga tillstånd i funktionen
%  ================================================================

  numparams = 1;

  %% Register number of dialog parameters
  block.NumDialogPrms = numparams;

  %% Register number of input and output ports
  block.NumInputPorts  = 1;
  block.NumOutputPorts = 1;

  %% Setup functional port properties to dynamically
  %% inherited.
  block.SetPreCompInpPortInfoToDynamic;
  block.SetPreCompOutPortInfoToDynamic;

  block.InputPort(1).Dimensions        = 5;
  block.InputPort(1).DirectFeedthrough = true;
  block.OutputPort(1).Dimensions       = 1;

  %% Set block sample time to continuous
  block.SampleTimes = [0 0];

  %% Setup Dwork
  block.NumContStates = numstates;

  %% Register methods
  block.RegBlockMethod('InitializeConditions',    @InitConditions);
  block.RegBlockMethod('Outputs',                 @Output);
  block.RegBlockMethod('Derivatives',             @Derivative);
%endfunction


function InitConditions(block)
%% ================================================================
```

```
%  Definiera initialtillståndet
%  ====================================================================
  x0 = block.DialogPrm(1).Data.x0;
%  ====================================================================

  %% Initialize Dwork
  block.ContStates.Data = x0;
%endfunction

%% ====================================================================
%  Definiera utsignalsekvationen
%  ====================================================================
function Output(block)
  w  = block.ContStates.Data;
  y1 = block.InputPort(1).Data(2);
  r  = w+y1;

  block.OutputPort(1).Data = r;
%endfunction

%% ====================================================================
%  Definiera de dynamiska ekvationerna
%  ====================================================================
function Derivative(block)
alfa = block.DialogPrm(1).Data.alfa;
d    = block.DialogPrm(1).Data.d;

u  = block.InputPort(1).Data(1);
y1 = block.InputPort(1).Data(2);
w  = block.ContStates.Data;

dw = -alfa*(w+y1)-d(1)*u+d(2)*sqrt(max(0,y1));

block.Derivatives.Data = dw;
```