TSFS06 LAB EXERCISE 1
Linear Residual Generation

April 20, 2020

1 OBJECTIVE

The goal with this lab is to show how linear models can be used for fault detection and isolation. Important concepts that are illustrated are fault modeling, residual generation, and basic fault isolation.

2 PREPARATIONS BEFORE THE LAB EXERCISE

To participate in the lab exercise, a number of preparation tasks must be solved and then reported and approved before the lab session. Visit one of the staff at scheduled time slots for approving the preparation tasks. The preparation tasks are: 1–2, 4–5, 8 and 10

The available lab material includes this textlab manual and a number of data files that are explained in Section 4. The data files can be downloaded from the course webpage. Read the manual and the data files carefully before the session. Read also the introduction to polynomial toolbox/Matlab carefully.

3 REQUIREMENTS AND IMPLEMENTATION

To pass this lab exercise requires correct answers to all tasks in Section 6. The answers are to be presented orally during the lab session or as a written report.

The main focus in this lab is on the tasks in Sections 6.5 and 6.6 which are relying on analyses that require simulations. **These tasks are important and can take a lot of time if the preparations are not done properly**.

The introductory tasks in Section 6 are meant to guide through the lab exercise and the tasks in Section 6.5 are organized for a step-by-step design of a diagnosis system that will be analyzed in Section 6.6

4 SIMULATION MODEL AND MATLAB SCRIPTS

Begin with downloading the data files for this lab exercise from the course homepage to your own directory.

The data files are:

1. A Simulink model of the process, `TSFS06lab1.slx`.
   which can be opened by writing `TSFS06lab1` in the Matlab command window.

2. Function `controllerdesign`
   This function is used to compute the controller to be used by the simulation. This data file is only used for the simulation and is not directly used in the lab exercise.

3. Skeleton file, `labskel.m`, to initialize variables that is used for the simulation. Here the variables used for the Simulink model to simulate are initiated. There is a detailed description of each variable in the beginning of this file.

   Important: It is strongly recommended that all Matlab code that you use in this lab exercise are written in this file. Then you can copy/paste the code to run in the command window. This is to simplify when making modifications and rerunning the code.

4. Function `decisioncalc`
   This function is used to compute diagnoses. The theory that is used is presented in detail in Chapter 3 in the course literature.

   Write `help decisioncalc` in the Matlab command window for more detailed explanations.

5. Function `lab1_modelcheck.p` and data file `lab1_modelcheck.m`
   These are used as help to verify that the model is formulated correctly.

5 MODEL

The model used describes a DC servo connected to a flywheel via a torsion spring. The process is inspired by industrial robots where a motor controls an arm (where elasticities provides oscillations). The model for such a motor+robotic arm is similar to the process considered in this lab exercise. Another application that reminds of the model in this lab is a truck driveline. The driveline contains the system from the engine to the wheels. The winding of the spring represents winding of the driveshaft (which can easily be as largeas 30 degrees).

A schematic of the lab process is shown in Figure 1 where u is the actuator that

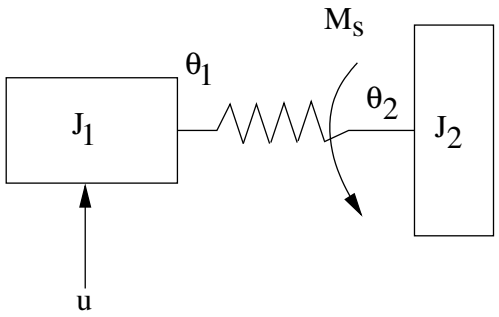

Figure 1: Schematic figure of the lab process.

result in engine torque. The variables θ_1 and θ_2 represent the angles of the motor and flywheel respectively, M_s is the troque transmitted by the spring, J_1, J_2 are inertias of the motor and flywheel respectively.

A simple linear model is

$$
\begin{aligned}
J_1\ddot{\theta}_1(t) &= ku(t) - \alpha_1\dot{\theta}_1(t) - M_s(t) \\
M_s(t) &= \alpha_2(\theta_1(t) - \theta_2(t)) + \alpha_3(\dot{\theta}_1(t) - \dot{\theta}_2(t)) \\
J_2\ddot{\theta}_2(t) &= -\alpha_4\dot{\theta}_2(t) + M_s(t)
\end{aligned}
$$

The first equation models the dynamic of the DC servo. The second equation models the torsion spring where the first term gives the torque caused by the torsion of the spring and the second term represent torque losses in the spring. The third equation models the dynamics of the flywheel. The parameters α_1, α_4 model viscous friction in the motor and flywheel suspension respectively, α_2 models the spring constant, and α_3 models energy losses in the spring.

The process has three sensors measuring the angle (y_1) and angular velocity (y_2) of the DC servo and the angle (y_3) of the flywheel.

6 TASKS

6.1 Modeling

Exercise 1. Formulate the fault-free model, including sensor equations, in the general form

$$H(p)x + L(p)z = 0$$

i.e., define the matrices, $H(p)$ and $L(p)$, and the vectors, x and z. Use the command `lab1_modelcheck` to verify that the model is well defined. Make sure that the order of the elements in vector z representing the observations is the same as

$$z = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ u \end{pmatrix}.$$

Otherwise, the simulation model and `lab1_modelcheck` will not work as expected.

Answer:

Exercise 2. Use additive fault models to model faults in the three sensors (f_1, f_2, f_3) and in the motor actuator (f_4). Define $F(p)$ such that the complete model is written in the form

$$H(p)x + L(p)z + F(p)f = 0$$

Use the command `lab1_modelcheck` to verify that the model is well defined.

Answer:

Exercise 3. The fault signals are not fully implemented in the Simulink model. Open the Simulink model `TSFS06lab1.slx` and connect the fault signals in the model.

6.2 Consistency relations

The purpose with this section is to clarify how to find consistency relations for a system described by a linear model. First, consider the *fault-free* system:

Exercise 4. Find a simple consistency relation by hand given the model equations.
Answer:

Here follows a number of tasks to illustrate how computer based tools can be used to find a base for all consistency relations.

Exercise 5. Which matrix spans all consistency relations? How many linearly independent base relations will be found? Motivate.

Answer: .....

Exercise 6. Use polynomial toolbox to find the basis. Write the vectors the span the space of consistency relations.
Answer:

Exercise 7. Each base relation corresponds to a consistency relation in the time domain. Write the differential equations corrsponding to each base relation.
Answer:

Now consider also the fault models of the system:

Exercise 8. Which matrix spans all consistency relations when the fault f_1 is decoupled? How many linearly independent base relations will be found? Motivate.

Answer: .....

Exercise 9. Use polynomial toolbox to find the base. Write the vectors that span the space of consistency relations where fault f_1 is decoupled. How can you see in the base that the fault f_1 is successfully decoupled?
Answer:

6.3 Redundancy analysis

Exercise 10. How many faults can maximally be decoupled in each residual and why?
Answer: .....
.....
.....

Exercise 11. Determine for each fault if it is detectable, strongly detectable, or not detectable.
Svar:

Exercise 12. Compute single fault isolability given by the model. Present the results as an isolability matrix as described in Section 3.6 in the course literature.
Svar:

6.4 Decision structure

Exercise 13. Define a set of tests/residual generators that give the maximum detectability and isolability performance defined by the model as computed in exercises 11 and 12. Take these results into consideration when doing exercise 10 when formulating the tests. Present the results by filling in the decision structure below.

	f_1	f_2	f_3	f_4
r_1				
r_2				

Motivate the choice of decision structure.

Answer: .....

Exercise 14. Assume that it is possible to generate a diagnosis system that works exactly as your decision structure specified in the previous task. How will the diagnosis system behave when there are multiple faults?
Answer: .....

.....

.....

6.5 Design and evaluation of residual generators

Exercise 15. Determine each residual's fault sensitivity, i.e., determine for each residual which faults that are weakly detectable, strongly detectable, or not detectable, respectively. Does the fault sensitivity of each residual correspond to the decision structure outlined in task 13.
Answer: .....

.....

.....

Exercise 16. Design a set of linear residual generators that realizes the specified decision structure. Describe how you choose the design parameters $d(p)$, $\gamma(p)$. Print the bode-plots showing the transfer functions from faults to residuals. Comment on the relation between the fault sensitivity of each residual in the bode-plots and the answer in exercise 15. Note, do *not* redefine the variable s using the control toolbox, do the state-space realization using the polynomial toolbox command `ss`. Example

```
1 >> R = ss(gamma*Nh*L, d);
```

Answer: .....

.....

.....

Exercise 17. Did you receive the specified decision structure? If not, explain why.

Answer: .....

Write the acheived decision structure (if it differs from the previously specified):

	f_1	f_2	f_3	f_4
r_1				
r_2				

6.6 Simulation and analysis of the diagnosis system

Exercise 18. Design thresholds T_i for all tests. In this lab, it is not necessary to perform accurate selection of suitable thresholds but simpler strategies for selecting thresholds are accepted.

When you have decided the threshold values, normalize the residual generators such that the corresponding thresholds can be set to 1. The normalization allows that the threshold block in the Simulink model can be used as it is without modifications.

Present your methodology and how the residual generators are normalized.

Answer: .....

.....

.....

Exercise 19. Simulate the different faults and evaluate detectability and isolability of the diagnosis system. Does the diagnosis system work as expected?

Experiment for different fault magnitudes and comment your results. How small faults can you reliably detect? We expect the fault sizes to be no larger than 10. Print relevant plots. Are there any faults that are more difficult to isolate than others? If that is the case, explain why.

Answer: .....

.....

.....

Exercise 20. In exercise 16, the parameter $d(s)$ was selected when designing the residual generators. Vary the parameter $d(s)$ for the different residual generators, simulate and comment your results.

Answer: .....

.....

.....

A APPENDIX

This appendix contains the source code of two important Matlab files for this lab: 'labskel.m' and 'decisioncalc.m'.

labskel.m

```
%% Lab skeleton for lab 1 in TSFS06: Linear residual generation

%% short description of the simulation model
% Variables saved in the workspace after simulation:
% t   time vector
% ref reference signal to the controller (not used here)
% y   measurement signals
% f   fault signals
% res residuals
% T   Thresholded residuals (with threshold 1)
%
% Variables from the workspace that is used in the simulation model:
% Fc     State-space model of the controller, computed in controllerdesign.m
% Model parameters J1, J2, k, alpha1, alpha2, alpha3, alpha4
% Tfault Time of failure injection. Default: 25
% NP     Defines noise power. Default value: NP=0.00005
% R      State-space model for the residual generator.
%        Code for this is done by student in this file.
% Rseed  Random seed for noise generation

clear
addpath /courses/TSFS06/polynomial/ % Linux

pinit

% Set the random seed
Rseed = floor(sum(100*clock));

% Define model constants. Make sure these are not overwritten in the code
% below. These values are used in the simulation model.
J1     = 1;
J2     = 0.5;
k      = 99/90;
alpha1 = 1;
alpha3 = 0.1;
alpha2 = 0.05;
alpha4 = 0.1;

% Define model matrices H, L, and F.
H = [];
L = [];
F = [];

%% Verify that the model is correctly formulated
% For this to work, make sure that the order of the signals in the z and f
% vectors is correct.
% z = (y1, y2, y3, u ) where y1, y2, and y3 are the sensors measuring
% theta1, theta1', and theta2
% f = (f1,f2,f3,f4)
```

```
lab1_modelcheck(H,L) % Verify that the nominal model is correct
lab1_modelcheck(H,L,F) % Verify that the model with faults is correct

%% Controller design
controllerdesign;

%% Noise power for simulating measurement noise
NP = 0.00005; % Set to 0 to simulate without noise
Tfault = 25;

%% Residual generator design
% Put your code here, name the residual generators R1, R2, etc.
%
% R = ss(Ra,Rb,Rc,Rd);
R1 = ss([0 0 0 0]); % Dummy residual generator
R2 = ss([0 0 0 0]); % Dummy residual generator

% Collect all residual generators into one single state-space object.
% The object must be called R for the simulation model to work.

R = [R1;R2];

%% Simulate
% Set fi=0 i=1,..,4 for the fault free case. Set auitable values for
% when faults are simulated.
f1=0;
f2=0;
f3=0;
f4=0;

% Simulation is started either by choosing the menu
% Simulation->Start in the Simulink window of by executing the code below
sim('TSFS06lab1');

%% Calculate the single fault diagnoses
% Define decision matrices for when residuals are over and under the
% thresholds respectively.
s0 = ones(2,5); % dummy exempel
s1 = [0 0 1 1 1;0 1 0 1 1]; % dummy exempel

% Calculate the diagoses
[S,alarm] = decisioncalc(T,s0,s1);

%% Plota and evaluate
figure(100)
plot( t, y ) % Plot
ylabel('utsignaler')
xlabel('t')
legend('y_1','y_2','y_3','Location','SE')

figure(101)
subplot(211)
plot( t, res(:,1))
ylabel('residual 1')
xlabel('t')
```

```
subplot(212)
plot( t, res(:,2))
ylabel('residual 2')
xlabel('t')

figure(102)
plot( t, alarm)
title('alarm')
xlabel('t')

figure(103)
subplot(231)
plot(t,S(:,1));
ylabel('NF indicator');
xlabel('t')
axis([min(t) max(t) -0.3 1.3]);

subplot(232)
plot(t,S(:,2));
ylabel('F_1 indicator');
xlabel('t')
axis([min(t) max(t) -0.3 1.3]);

subplot(233)
plot(t,S(:,3));
ylabel('F_2 indicator');
xlabel('t')
axis([min(t) max(t) -0.3 1.3]);

subplot(234)
plot(t,S(:,4));
ylabel('F_3 indicator');
xlabel('t')
axis([min(t) max(t) -0.3 1.3]);

subplot(235)
plot(t,S(:,5));
ylabel('F_4 indicator');
xlabel('t')
axis([min(t) max(t) -0.3 1.3]);
```

decisioncalc.m

```
function [S,alarm] = decisioncalc(T,s0,s1)
% DECISIONCALC  Calculate the diagnosis S from thresholded residuals
%
%   Syntax: [S,alarm] = decplot(T,s0,s1)
%
%   In:      T  The thresholded residuals. Each thresholded
%               residual in separate columns.
%           s0  Binary table describing the decision when residuals is
%               below the threshold.
%           s1  Binary table describing the decision when residuals is
%               above the threshold
%
```

```
%   Out: alarm  Signal indicating when a fault is detected
%            S  Binary matrix describing the diagnosis. A 1 on row i
%               column j indicates fault mode j is detected at time i.
%
%   Example:
%          Consider the decisionstructure
%
%          NF f1 f2 f3
%       T1 0  0  X  X
%       T2 0  X  0  1
%       T3 0  X  X  0
%
%       s0 = [1 1 1 1;1 1 1 0;1 1 1 1]
%       s1 = [0 0 1 1;0 1 0 1;0 1 1 0]
%
%   Fault mode NF (No Fault) MUST be the first fault mode.

mT = size(T,2);
nT = size(T,1);

if (mT~=size(s0,1))|(mT~=size(s1,1))
  error('decstruc mismatch');
end
nF = size(s1,2);

si = zeros(nT,mT*nF);
for k=1:mT
  idx = find(T(:,k)==0);
  si(idx,(k-1)*nF+1:k*nF)=ones(length(idx),1)*s0(k,:);
  idx = find(T(:,k)==1);
  si(idx,(k-1)*nF+1:k*nF)=ones(length(idx),1)*s1(k,:);
end
S = ones(nT,nF);
for k=1:nF
  for l=1:mT
    S(:,k) = S(:,k).*si(:,(l-1)*nF+k);
  end;
end
alarm = 1-S(:,1);
```

lab1_modelcheck.m

```
function r=lab1_modelcheck(H,L,F)
% LAB1_MODELCHECK  Verify that the model equations are correct
%
%   Syntax: r = lab1_modelcheck(H,L,F)
%
%   In:      H, L, F are the polynomial matrices in the model equations
%            H(s)x + L(s)z + F(s)f=0
%
%            The argument for F is not required, i.e., if only matrices
%            H and L are used, only the fault-free model is tested.
%
%            Important note: FFor this command to work, the z and f vectors
%            must be ordered as in the lab text.
```

```
%
%           z = (y1, y2, y3, u) where y1=motor angle, y2,
%           motor angular velocity, y3=angle at wheel, u
%           torque fot the DC servo.
%
%           f = (f1,f2,f3,f4), where f1-f3 are sensor faults for sensors
%           y1 to y3 and f4 is a fault in the actuator.
%
%   Out:    r is 1 if the model is OK and 0 if it is not.
%
```