

*An easy guide to*  
***Control System Toolbox***  
**version 4 (for MATLAB 5)**

by

[\*Finn Haugen\*](#)

Telemark College  
Porsgrunn, Norway

*February 25, 1998*

This publication can be downloaded and copied freely, but reference to the source is appreciated.

---

***Contents:***

**[1](#) Introduction**

**[2](#) Creating and handling LTI-models (Linear Time-Invariant)**

[2.1](#) Creating models

[2.2](#) Extracting subsystems

[2.3](#) Setting and accessing LTI-properties

[2.4](#) Conversion between state-space models and transfer functions

[2.5](#) Conversion between continuous-time and discrete-time models

[2.6](#) Combining models (series, parallel, and feedback)

### **[3](#) Model analysis tools**

[3.1](#) Simulation (time response)

[3.2](#) Poles, eigenvalues, and zeros

[3.3](#) Frequency response

[3.4](#) Stability analysis based on frequency response

[3.5](#) Pade-approximations

[3.6](#) The LTI-viewer

### **[4](#) Controller design tools**

[4.1](#) Root locus

[4.2](#) Pole placement

[4.3](#) Optimal control

[4.4](#) Kalman estimator (or filter)

### **[5](#) Overview over functions in Control System Toolbox**

---

## 1 Introduction

This text gives an easy guide to Control System Toolbox version 4 which assumes the installation of MATLAB version 5.

The text is self-instructive: You are asked to perform a number of simple tasks through which you will learn to master this toolbox, and the expected responses are even shown in the text.

It is assumed that you have the basic knowledge about dynamic systems and control theory. It is also assumed that you have basic skills in MATLAB. May I suggest [Learn MATLAB 5 in 6 hours!?](http://www.techteach.no/textbook.htm) (See <http://www.techteach.no/textbook.htm>). And if you want to simulate your (control) systems, you should consider using SIMULINK which is based on MATLAB. [Learn SIMULINK 2 in 3 hours!](#) (see [www.techteach.no](http://www.techteach.no)) gives an introduction.

Compared to the older versions of this toolbox, this version has some new features:

- Systems or models are represented as *LTI-objects* (Linear Time-Invariant) (we may also say LTI-systems and LTI-models when it is proper). An LTI-object or -system can be represented by a single MATLAB-variable, such as `sys1`. The introduction of LTI-objects have made it possible to simplify the syntax of the functions. For example, you execute `"bode(sys1)"` whether `sys1` is a state-space model or a transfer function, and whether it is a continuous-time model or a discrete-time model. However, version 3 of the toolbox is still supported, which means that you can still execute `"bode(num,den)"` where `num` and `den` are ordinary MATLAB-vectors containing the coefficients of the numerator and denominator polynomials of the transfer function. Note that there is available a block in SIMULINK version 2 which can be used to represent LTI-objects.
- The *LTI-viewer* is a new graphical user interface which makes it quicker to apply various analytical tools such as simulation and frequency response on one model or on a number of models simultaneously.
- Time delays can be included in a continuous-time model.

In the following, CST will be used as an abbreviation for Control System Toolbox.

## 2 Creating and handling LTI-models (Linear Time-Invariant)

The models supported by CST are continuous-time models and discrete-time models of the following types:

- transfer functions

- state-space models

The CST also supports zero-pole-gain models, but since we will in practice hardly use that model format, it is not described in this text. If you need to use it, you can get information by executing "help zpk".

The various functions will be introduced via simple examples.

## 2.1 Creating models

We will start with continuous-time models, and then take discrete-time models.

Transfer functions

The function "tf" creates *transfer functions*. "tf" needs two MATLAB-vectors, one containing the coefficients of the *numerator* polynomial - taken in descending orders - and one for the *denominator* polynomial of the transfer function. As an example we will create the transfer function

$$H_0(s) = K_0*s/(T_0*s+1)$$

```
K0=2; T0=4;  
num0=[K0,0]; den0=[T0,1];  
H0=tf(num0,den0);
```

To display H0 we execute

```
H0
```

MATLAB's response is

```
Transfer function:
```

```
  2 s  
-----
```

$$4s + 1$$

An LTI-model can contain a *time-delay*, as in the following model:

$$H_d(s) = [K_0*s/(T_0*s+1)]e^{(-T_{delay}*s)}$$

This system is created by (except from the time-delay term,  $H_d$  is equal to  $H_0$  defined above):

```
Tdelay=1;
Hd=tf(num0,den0,'Td',Tdelay)
```

To display  $H_d$  we execute

```
Hd
```

MATLAB's response is

Transfer function:

$$\frac{2s}{4s + 1}$$

Input delay: 1

The expressions below create a two-input one-output *MIMO* (Multiple-Input Multiple-Output) system of the form

$$y(s) = H_1(s) u_1(s) + H_2(s) u_2(s)$$

The transfer function can be expressed as

$$H=[H_1,H_2]$$

```

K1=2; T1=4; num1=[K1,0]; den1=[T1,1];
K2=3; T2=5; num2=[K2]; den2=[T2,1];
Tdelay1=1;
Tdelay2=2;
H1=tf(num1,den1,'Td',Tdelay1);
H2=tf(num2,den2,'Td',Tdelay2);
H=[H1,H2];

```

To display H we execute

```
H
```

MATLAB's response is

Transfer function from input 1 to output:

$$\frac{2 s}{4 s + 1}$$

Transfer function from input 2 to output:

$$\frac{3}{5 s + 1}$$

Input delays (listed by channel): 1 2

Note 1: If the system has no time-delay, it is sufficient to write `tf(num1,den1)`.

Note 2: If the transfer function has the form

$$H(s) = \begin{bmatrix} H1(s) \\ H2(s) \end{bmatrix}$$

which may correspond to a one-input two-output system, the CST requires that the time-delays are equal.

Suppose we have created a transfer function, and we want to *retrieve detailed information* about the model. The function "tfdata" retrieves the information, which is stored in *cell arrays* (both in SISO-cases and in MIMO-cases). As an example, the following expressions retrieve the polynomials containing the coefficients of the numerator and denominator polynomials of the (SISO-) transfer function Hd(s) created earlier:

```
[numHd,denHd,Ts]=tfdata(Hd)
```

MATLAB's response is

```
numHd =  
[1x2 double]  
denHd =  
[1x2 double]  
Ts =  
0
```

Here, Ts is the sampling time which is zero for continuous-time models but non-zero for discrete-time models.

The response above is not very descriptive since we do not see any numerical values for the numerator and denominator coefficients. Suppose we want to retrieve these coefficients as *vectors*. This can be done as follows:

```
inputno=1;  
outputno=1;  
numpolHd=numHd{outputno,inputno}  
denpolHd=denHd{outputno,inputno}
```

MATLAB's response is

```
numpolHd =  
2 0  
denpolHd =  
4 1
```

Note: The use of `inputno` and `outputno` above can be applied also for MIMO-systems. For example, `inputno=3` and `outputno=2` corresponds to the transfer function from input no. 3 to output no. 2 in a MIMO-system.

Note: The time-delay of `Hd(s)` was ignored in the data retrieval above.

### State-space models

The function "ss" creates linear *state-space models* having the form

$$\dot{x}/dt = Ax + Bu$$

$$y = Cx + Du$$

as illustrated in the following example: Given the following state-space model:

$$\dot{x}_1/dt = x_2$$

$$\dot{x}_2/dt = -4*x_1 - 2*x_2 + 2*u$$

$$y = x_1$$

This model is created by

```
A=[0,1;-4,-2]; B=[0;2]; C=[1,0]; D=[0];  
ss1=ss(A,B,C,D);
```

To display `ss1` we execute

```
ss1
```

MATLAB's response is

a =

x1 x2

x1 0 1.00000

x2 -4.00000 -2.00000

b =

u1

x1 0

x2 2.00000

c =

x1 x2

y1 1.00000 0

d =

u1

y1 0

### **Continuous-time system.**

Above, a, b, c, and d, which are default names, are used only for displaying purposes, and they do in fact not exist in the workspace. To *retriev*e the system parameters and give them specific names, we can execute

```
[A1,B1,C1,D1,Ts]=ssdata(ss1)
```

MATLAB's response is

A1 =

0 1

-4 -2

B1 =

0

2

C1 =

1 0

D1 =

0

Ts =

0

### Discrete-time models

Discrete-time (z-)transfer functions and state-space models are created in the same way as for continuous-time models.

As an example, to create the transfer function

$$H_{dis}(z) = (z+0.5)/(z^2+1.5z+2)$$

with sampling time 0.4 (time-units), we execute

```
numdis=[1,0.5]; dendis=[1,1.5,2]; Tsdis=0.4  
Hdis=tf(numdis,dendis,Tsdis)
```

To display Hdis we execute

```
Hdis
```

MATLAB's response is

Transfer function:

$z + 0.5$

-----

$z^2 + 1.5 z + 2$

Sampling time: 0.4

In digital signal processing (DSP) it is common to use  $z^{-1}$  in the transfer functions (and not explicitly  $z$ ). The function "filt" supports this DSP-format, and "filt" requires two vectors (one for the numerator and one for the denominator) containing the coefficients for descending orders of  $z^{-1}$ . As an example, to create the transfer function

$$H_{dsp}(z) = (1+0.5z^{-1})/(1+1.5z^{-1}+2z^{-2})$$

with sampling time 0.4 (time-units), we execute

```
numdsp=[1,0.5];  
dendsp=[1,1.5,2];  
Tsdsp=0.4;
```

```
Hdsp=filt(numdsp,dendsp,Tsdsp)
```

To display Hdsp we execute

```
Hdsp
```

MATLAB's response is

Transfer function:

$1 + 0.5 z^{-1}$

-----

$1 + 1.5 z^{-1} + 2 z^{-2}$

Sampling time: 0.4

Note: Hdsp(s) is not equal to Hdis(s), although the coefficient vectors are equal!

Discrete-time *state-space models* are assumed to have the form

$$x(k+1) = A_d x(k) + B_d u(k)$$

$$y(k) = C_d x(k) + D_d u(k)$$

where  $A_d$ ,  $B_d$ ,  $C_d$ , and  $D_d$  are matrices.  $x(k)$  is the state-variable,  $u(k)$  is the input variable, and  $y$  is the output variable (these variables are generally vectors having one or more elements).  $k$  is the discrete-time or time-index. Discrete-time *state-space models* are created using the syntax

$$ssdis=ss(A_d, B_d, C_d, D_d, T_s)$$

where  $T_s$  is the sampling-time.

## 2.2 Extracting subsystems

To create a new system (say, sys1) which is a subsystem of an existing system (say, sys), we use the syntax

```
sys1=sys(outputno,inputno)
```

As an example, let us extract the transfer function Hsub from input 2 (u2) to the output (y) from the MIMO-transfer function H defined earlier in this chapter:

```
outputno=1; inputno=2;
Hsub=H(outputno,inputno)
```

MATLAB's response is

Transfer function from input 2 to output:

$$\frac{3}{5s + 1}$$

Input delay: 2

In fact, we can use ordinary matrix-like accessing, like for example "subsys=sys(1,2:4)" which selects inputs 2 to 4 and output 1, that is, three subsystems altogether.

### 2.3 Setting and accessing LTI-properties

Several *properties* can be attached to the LTI-objects (or LTI-models). The table below gives an overview over the properties. To get detailed information about legal values of the properties, execute "help ltiprops".

In the table below, NU, NY, and NX refer to the number of inputs, outputs, and states (when applicable). TF refers to transfer functions, SS to state-space models, and ZPK to zero-pole-gain models.

Name	Description	Applies to LTI-object of type

Ts	Sample Time	All
Td	Input delays	All (only continuous systems)
Inputname	Input names	All
Outputname	Output names	All
Notes	Notes on model history	All
Userdata	Additional data	All
Num	Numerator	TF
Den	Denominators	TF
Variable	Transfer function variable (e.g. 's' or 'z')	TF, ZPK
Z	Zeros	ZPK
P	Poles	ZPK
K	Gains	ZPK
A	Matrix A	SS
B	Matrix B	SS
C	Matrix C	SS
D	Matrix D	SS

To see the values of the properties of a model, we use "get". Let us as an example look at the properties of the MIMO-transfer function H defined earlier:

```
get(H)
```

MATLAB's response is

```
num = {1x2 cell}
```

```
den = {1x2 cell}
```

```
Variable = 's'
```

```
Ts = 0
```

```
Td = [1 2]
```

```
InputName = {2x1 cell}
```

```
OutputName = {''}
```

```
Notes = {}
```

```
UserData = []
```

We can access specific LTI-properties. Let us as an example assign the value of the property Td of H to the variable Tdvec:

```
Tdvec=get(H,'Td')
```

MATLAB's response is

```
Tdvec =
```

```
1 2
```

To *set* certain LTI-properties, we use "set". Let us as an example set the property InputName of H to {'Voltage';'Flow'}, which is a cell array:

---

These input names are now used when displaying information about H:

H

MATLAB's response is

Transfer function from input "Voltage" to output:

$$\frac{2 s}{4 s + 1}$$

Transfer function from input "Flow" to output:

$$\frac{3}{5 s + 1}$$

Input delays (listed by channel): 1 2

## 2.4 Conversion between transfer functions and state-space models

From state-space model to transfer function

The function "tf" converts state-space models to a corresponding transfer function according to the formula(s)

$$H(s) = C(sI - A)^{-1}B + D \text{ (continuous-time case)}$$

$$H_d(z) = C_d(zI - A_d)^{-1}B_d + D_d \text{ (discrete-time case)}$$

In the following example a *continuous-time* state-space model is converted to a corresponding (continuous-time) transfer function:

```
A=1; B=2; C=3; D=4;
sscont1=ss(A,B,C,D);
Hcont1=tf(sscont1)
```

MATLAB's response is

Transfer function:

$$\frac{4s + 10}{s + 1}$$

And in the following example a *discrete-time* state-space model is converted to a corresponding (discrete-time) transfer function (the sampling-time is 0.1):

```
Ad=0.5; Bd=0.2; Cd=0.3; Dd=0.4; Ts=0.1;
ssdis1=ss(Ad,Bd,Cd,Dd,Ts);
Hdis1=tf(ssdis1)
```

MATLAB's response is

Transfer function:

$$\frac{0.4z - 0.14}{z - 0.5}$$

Sampling time: 0.1

From transfer function to state-space model

"ss" converts a transfer function to a state-space model. But this conversion is not unique, since there are an infinite number of state-space

models which have the same transfer functions. Hence, "ss" chooses one such realization, called the controller-canonical form.

## 2.5 Conversion between continuous-time and discrete-time models

### From discrete-time to continuous-time

A continuous-time LTI-model can be discretized using the function "c2d", using the syntax

$$\text{sysdis}=\text{c2d}(\text{syscont},\text{Ts},\text{method})$$

Ts is the sampling time. method is a string telling which discretizing method to be used, and the following are available:

'zoh' for zero order hold.

'foh' for first order hold.

'tustin' for Tustin's method of bilinear transformation.

'prewarp' for Tustin's method with frequency prewarping.

'matched' for the matched pole-zero method.

As an example, let us discretize a continuous-time model (transfer function) which contains a time-delay, assuming zero order hold:

```
K=2; T=4; num=[K,0]; den=[T,1]; Tdelay=1;
Hcont4=tf(num,den,'Td',Tdelay);
Ts=0.2;
Hdis4=c2d(Hcont4,Ts,'zoh')
```

MATLAB's response is

Transfer function:

$$0.5 z - 0.5$$

-----

$$z^6 - 0.9512 z^5$$

Sampling time: 0.2

(The high order of the denominator is due to the time-delay of the continuous-time system.)

From discrete-time models to continuous-time models

Use the function "d2c".

## 2.6 Combining models (in series, parallel, and feedback)

A model can be thought of as a block with inputs and outputs and containing a transfer function or a state-space model inside it. The functions "series", "parallel", and "feedback" can be used to perform basic block-diagram manipulation, as demonstrated in the following. These functions apply to continuous-time models as well as discrete-time systems.

As an example, assume given the following transfer functions:

$$H_a(s) = 1/(s+2)$$

$$H_b(s) = 3/(s+4)$$

First, let us create the above two systems:

```
k1=1; k2=2; k3=3; k4=4;  
Ha=tf(k1,[1,k2]);  
Hb=tf(k3,[1,k4]);
```

Let us connect  $H_a(s)$  and  $H_b(s)$  in *series*, which means that  $H_{ser}(s)$  becomes  $H_a(s)*H_b(s)$ :

```
Hser=series(Ha,Hb)
```

MATLAB's response is

```
Transfer function:          3
```

$$\frac{\quad}{s^2 + 6s + 8}$$

Below,  $H_a(s)$  and  $H_b(s)$  are connected in *parallel*, which means that  $H_{par}(s)=H_a(s)+H_b(s)$ .

```
Hpar=parallel(Ha,Hb)
```

MATLAB's response is

Transfer function:

$$\frac{4s + 10}{s^2 + 6s + 8}$$

Finally,  $H_a(s)$  and  $H_b(s)$  are connected in a negative *feedback* loop with  $H_a(s)$  being in the forward path and  $H_b(s)$  being in the negative feedback path:

```
feedbsign=-1;
Hfeedb=feedback(Ha,Hb,feedbsign)
```

Above,  $feedbsign$  is the sign of the feedback, and we set it to either -1 or 1. The transfer function  $H_{feedb}(s)$  is given by

$$H_{feedb}(s)=H_a(s)/[1+feedbsign*H_a(s)*H_b(s)]$$

MATLAB's response to the expressions above is

Transfer function:

$$\frac{s + 4}{s^2 + 6s + 11}$$

### 3 Model analysis tools

As a starting point for this chapter, let us define a continuous-time transfer function,  $H1c(s)$ , and a discrete-time transfer function,  $H1d(z)$  which will be the discretized version of  $H1c(s)$ . We take  $H1c(s)$  as

$$H1c(s) = (s+0.5)/(s^2+2s+4)$$

and we create  $H1c(s)$  with

```
n1=[1,0.5]; d1=[1,2,4];
H1c=tf(n1,d1)
```

MATLAB's response is

Transfer function:

$$\frac{s + 0.5}{s^2 + 2s + 4}$$

Then, let  $H1d(z)$  be the zero-order hold discretized version of  $H1c(s)$ :

```
Ts=0.2;
H1d=c2d(Hc1,Ts,'zoh')
```

MATLAB's response is

Transfer function:

$$0.1692 z - 0.1529$$

-----

$$z^2 - 1.54z + 0.6703$$

Sampling time: 0.2

### 3.1 Simulation (time response)

We can simulate the *step response* of one or several LTI-models with "step". The input step is then a unit step (the step height is equal to one). Let us simulate both H1c and H1d from t=0 to 10 time units:

```
Tf=10; step(H1c,H1d,Tf)
```

The resulting plot is not shown here in this text.

We can store the simulated time response, y, and the time vector, t, by using return (left) arguments:

```
Tf=10; [y,t] = step(H1c,H1d,Tf);
```

In this case, no plot is shown on the screen.

"impulse" simulates the *impulse response*:

```
Tf=10; impulse(H1c,Tf)
```

The resulting plot is not shown here in this text.

"initial" simulates the response from the initial state of an LTI-model.

"lsim" is a more general simulation function as it accepts any input signal, not just step or impulse. To generate the input signal, we may use

"gensig", which produces a sine wave, or a square wave, or periodic pulses. Let us as an example simulate H1c with a sinusoidal input, u, having period Tp=0.5, final time Tf=10, and time step Tstep=0.01:

```
 Tp=0.5; Tf=10; Tstep=0.01; [u,t]=gensig('sin',Tp,Tf,Tstep);  
 lsim(H1c,u,t)
```

The resulting plot is not shown here in this text.

Above, "gensig" was used to generate the input signal. Other signal generators are "square", "sawtooth", and "stepfun".

### 3.2 Poles, eigenvalues, and zeros

We can calculate the poles of an LTI-model with "pole", as in

```
p=pole(H1c)
```

MATLAB's response is

```
p =  
-1.0000 + 1.7321i  
-1.0000 - 1.7321i
```

For a state-space model we can analogously calculate the eigenvalues with "eig".

"tzero" calculates zeros (transmission zeros for MIMO-models). Here is an example based on H1c(s):

```
z=tzero(H1c)
```

MATLAB's response is

$z =$

-0.5000

With "[pol,zer]=pzmap" we can calculate and plot both the poles and the zeros. Omitting the return-parameters just plots the poles and the zeros in the complex plane:

```
pzmap(H1c)
```

The resulting plot is not shown here in this text.

Poles and eigenvalues will always be real numbers and/or complex conjugate numbers. "damp" calculates the relative damping,  $\zeta$ , and the natural (or undamped) resonance frequency,  $\omega_0$  for such complex poles/eigenvalues. It is assumed that the poles or eigenvalues are given as the roots of (the characteristic equation)

$$s^2 + 2\zeta\omega_0s + \omega_0^2$$

Here is one example:

```
damp(H1c)
```

MATLAB's response is

Eigenvalue Damping Freq. (rad/s)

-1.00e+000 + 1.73e+000i 5.00e-001 2.00e+000

-1.00e+000 - 1.73e+000i 5.00e-001 2.00e+000

### 3.3 Frequency response

The frequency response of an LTI-model can be calculated with "bode". Here, "bode" shows the Bode-plots of both H1c and H1d, in the same diagram:

```
bode(H1c,H1d)
```

The resulting plot is not shown here in this text. Using the syntax

$$[\text{mag,phase,w}] = \text{bode}(\text{sys})$$

the magnitude and the phase (in degrees), and the frequency vector (in rad/s) used by "bode", are returned. With "bode(sys,w)" the Bode-plot will be drawn using our own frequency vector, w.

The frequency response can be drawn in a Nichols-diagram with "nichols", and in a Nyquist-diagram with "nyquist" (examples follows in the next section).

### 3.4 Stability analysis based on frequency response

The function "margin" calculates the stability margins and the cross-over frequencies for a feedback system (continuous-time or discrete-time). In the following example "margin" draws a Bode-plot and writes the stability margins and the cross-over frequencies in the diagram. The loop transfer function of the feedback system is

$$H_{\text{loop}}(s) = (K/s) * e^{-T_{\text{del}} * s}$$

where  $K=0.19$ , and  $T_{\text{del}}=4.17$  (time-delay). Let us first create the LTI-object Hloop:

```
K=0.19; Tdel=4.17;  
Hloop=tf(K,[1,0],'Td',Tdel);
```

Then we execute

```
margin(Hloop)
```

The resulting Bode-plot is not shown in this text. We will see that the gain margin is 5.9 dB, and the phase margin is 44.3 degrees. The amplitude crossover frequency, which is the bandwidth of the loop, is 0.2 rad/sec (or 0.2 rad per time unit), and the phase crossover frequency is 0.4 rad/s. In general, these values are returned by "margin" if it is invoked with return arguments, using this syntax: "[Gm,Pm,Wcg,Wcp] = margin(Hloop)". However, on my installation, trying to execute this expression produced the error message "Cannot handle input delays.", which is strange since "margin(Hloop)" worked all right. But we will not give in, and will return to this problem after we have approximated the time-delay part of the (loop) transfer function with a Pade-approximation.

Stability analysis can also be performed using a Nyquist-plot or a Nichols-plot. Let us try Nichols:

```
nichols(Hc1),ngrid
```

Above, ngrid adds a Nichols grid to the plot. The resulting Nichols-plot is not shown in this text. However, you will see that the interesting details around the critical point (gain = 1 = 0dB, and phase = - 180 degrees) do not appear clearly. Try "zoom" to see the details!

### 3.5 Pade-approximation

Pade-approximations are rational transfer functions which approximates the transfer function of a time-delay. Pade-approximations can be calculated with "pade". Let us as an example develop a 5<sup>th</sup> order approximation to a time-delay of 2 seconds (or time-units), and represent it as an LTI-model (or -object):

```
Tdelay=2; N=5;  
[numpade,denpade]=pade(Tdelay,N);  
tfpade=tf(numpade,denpade)
```

MATLAB's response is

Transfer function:

$$-s^5 + 15 s^4 - 105 s^3 + 420 s^2 - 945 s + 945$$

$$s^5 + 15 s^4 + 105 s^3 + 420 s^2 + 945 s + 945$$

Now, let us (for fun) compare the step response of tfpade with the ideal step response of the time-delay (of 2 sec.). To generate a time-delayed step, we will use "stepfun".

```
tfinal=5; [ypade,time]=step(tfpade,tfinal);  
T0=2; yideal=stepfun(time,T0);  
plot(time,[ypade,yideal])
```

The plot will not be shown in this text, but it illustrates the approximation effectively.

In Sec. 3.4 we saw that "margin" did not accept a model containing a time-delay. So let us in stead use a Pade-approximation for the time-delay of the transfer function Hloop(s). "pade" is now used a little different than above:

```
K=.19; td=4.17; Hloop=tf(K,[1,0],'Td',td) %Just to repeat the creation of Hloop.  
Hlooppade=pade(Hloop,5);
```

So, Hlooppade is the Pade-approximation of the entire original transfer function Hloop, and not of just the time-delay. Finally, let us try "margin" one more:

```
[Gm,Pm,Wcg,Wcp] = margin(Hlooppade)
```

MATLAB's response is

Gm =

1.9826

Pm =

44.6046

Wcg =

0.3767

Wcp =

0.1900

which in fact is very close to the correct values shown in the Bode-plot produced by "margin" in Sec. 3.4. (Note that  $G_m = 1.9826$  is equal to 5.94 dB).

### 3.6 The LTI-viewer

LTI-viewer makes it easy to perform simulations, frequency response plots etc. for one or several of the LTI-models which exist in the working directory. The LTI-viewer is easy to use. Just execute

```
Ltview
```

and try it.

## 4 Controller design tools

### 4.1 Root locus

Given a closed-loop system with negative feedback and loop transfer function  $k*L0(s)$ , where  $k$  is a constant. [ $k*L0(s)$  is then the product of the individual transfer functions in the loop.] The function "rlocus" calculates the poles of the closed-loop system as a function of  $k$ . These poles are the root locus of the closed-loop system. The poles are the roots of the characteristic equation  $1+k*L0(s)$ . Here is one example: First, let us create the transfer function  $L0(s)$ :

```
L0=tf(1,[1 2 1 0])
```

MATLAB's response is

Transfer function:

$$\frac{1}{s^3 + 2s^2 + s}$$

Now, to plot the poles of the closed-loop system which have the loop transfer function of  $k*L0(s)$ , we execute:

```
L0=tf(1,[1 2 1 0])
rlocus(L0),sgrid
```

The resulting figure showing the root locus is not shown here in this text. Note that "sgrid" plots a grid of  $z$  and  $w_0$ -curves. ("zgrid" does the same in the discrete-time case.) MATLAB chooses automatically the proper range of  $k$  for which the root locus is drawn. However, we can use our own values of  $k$  by executing "rlocus(L0,k0)". Of course, we must have created  $k0$  (a vector) in advance.

Once the root locus is drawn (using "rlocus"), we can choose a pole location in the root locus plot, and ask MATLAB to calculate the corresponding value of  $k$ . This is done with "rlocfind". As an example, let us find (the approximate) value of  $k$  which causes one of the closed-loop poles to be placed on the imaginary axis (the system is then on the stability limit). (It can be shown that  $k0=2$  produces closed-loop poles at  $-2, \pm i$ .) We execute:

```
[k0,pole0]=rlocfind(L0)
```

"rlocfind" puts a crosshair cursor on the root locus plot. Let us, with the cursor, select the pole at  $+i$  (on the imaginary axis). As a result, MATLAB returns the both the selected point and the corresponding value of  $k$ :

```
selected_point =
-0.0023 + 0.9912i

k0 =
1.9606

pole0 =
```

-1.9921

-0.0040 + 0.9921i

-0.0040 - 0.9921i

In fact, our selected point (-0.0023 + 0.9912i) is not exactly on the root locus, but MATLAB finds the value of  $k_0$  corresponding to the nearest point, and calculates the corresponding poles (pole0).

## 4.2 Pole placement

Suppose the system to be controlled has the (continuous-time-) state-space model  $dx/dt=Ax+Bu$ . The system is to be controlled by state-feedback:  $u=-Gx$  where  $G$  is a matrix of gains. Then the closed-loop system has the state-space model  $dx/dt=Ax+B(-Gx)=(A-BG)x$ . The function "place" calculates  $G$  so that the eigenvalues of  $(A-BG)$  (the closed-loop system) are as specified. Here is one simple example: Assume the system to be controlled is given by  $dx/dt=u$  (which is an integrator, from  $u$  to  $x$ ). The closed-loop eigenvalue shall be  $E=-4$ :

```
E=-4;  
A=0;B=1;  
g1=place(A,B,E)
```

MATLAB's response is

```
place: ndigits= 15
```

```
Warning: Pole locations are more than 10% in error.
```

```
g1 =
```

```
4
```

(The answer  $g_1=4$  is correct despite the warning message.)

"place" works similarly for *discrete-time* systems: Suppose the system to be controlled has the model

$$x(k+1)=A_d x(k)+B_d u(k)$$

and the controller is

$$u(k)=-G_d x(k)$$

Then, the closed-loop system has the state-space model

$$x(k+1)=(A_d-B_d G_d)x(k)$$

"place" calculates  $G_d$  so that the eigenvalues of  $(A_d-B_d G_d)$  (the closed-loop system) are as specified in the vector  $E_d$ . Thus, we can execute "Gd=place(Ad,Bd,Ed)".

"place" can be used to calculate *state-estimator* gains, too: Suppose given a system with state-space model

$$dx/dt=Ax+Bu, y=Cx+Du, y=Cx+Du$$

and that the states  $x$  are to be estimated by the estimator

$$dx_e/dt=Ax_e+Bu+Ke, y_e=Cx_e+Du$$

where the residual  $e$  is given by  $e=y-y_e$ . It can be shown that the state estimation error  $z$  is given by  $dz/dt=(A-KC)z$ . The eigenvalues of this error model are given by  $E=eig(A-KC) \equiv eig(A'-C'K')$ . (Here,  $A'$  means the transpose of  $A$ .) Thus, we can calculate the estimator-gain by executing "K1=place(A',C',Ee); K=K1'".

### 4.3 Optimal control

The function "dlqr" performs linear-quadratic regulator design for discrete-time systems. This means that the controller gain  $G$  in the feedback

$$u(k)=-Gx(k)$$

for the controlled system

$$x(k+1) = A_d x(k) + B_d u(k)$$

is calculated so that the cost function

$$J = \text{Sum} \{x'Qx + u'Ru + 2*x'Nu\}$$

is minimized. The syntax is

$$[K,S,E] = \text{dlqr}(A_d, B_d, Q, R, N)$$

The matrix N is set to zero if omitted. Above, S is the solution of the Riccati equation, E contains the closed-loop eigenvalues which are the eigenvalues of  $(A_d - B_d G)$ . Here is one example:

```
Ad=0.85;Bd=0.15;
Q=1;R=0.2;
[G,S,E]=dlqry(Ad,Bd,Q,R)
```

MATLAB's response is

G =

1.1797

S =

2.3370

E =

0.6731

"lqr" works analogously for continuous-time systems. "dlqry" minimizes the cost function  $J = \text{Sum} \{y'Qy + u'Ru\}$  in which only the *output vector* y, and not the whole state-vector x, is weighted. "dlqy" works analogously for continuous-time systems.

#### 4.4 Kalman estimator (or filter)

"kalman" designs both continuous-time and discrete-time Kalman-filters. The *discrete-time* Kalman-filter is more commonly used. In this case, it is assumed that the system is given by a discrete-time state-space model in the form of an LTI-object. The model is

$$x(k+1) = Ax(k) + Bu(k) + Gw(k)$$

$$y(k) = Cx(k) + Du(k) + Hw(k) + v(k)$$

where  $w$  is the process noise, and  $v$  is the measurement noise. Note that the LTI-object must be created by the expression

$$\text{sys}=\text{ss}(A,[B,G],C,[D,H])$$

$Q$ ,  $R$ , and  $N$  are the white-noise covariances as follows:

$$E\{ww'\} = Q, E\{vv'\} = R, E\{wv'\} = N$$

(In the calling syntax,  $N$  may be omitted if it is zero.) The Kalman-estimator is

$$x(k+1|k) = Ax(k|k-1) + Bu(k) + L[y(k) - y(k|k)] \text{ (apriori estimate)}$$

$$x(k|k) = x(k|k-1) + M[y(k) - y(k|k)] \text{ (aposteriori estimate)}$$

where

$$y(k|k) = Cx(k|k) + Du(k)$$

The syntax of "kalman" is

$$[Kest,L,P,M,Z] = \text{kalman}(\text{sys},Q,R,N)$$

Here,  $Kest$  is the Kalman estimator in the form of an LTI-object having  $[u; yv]$  as inputs ( $yv$  is the measurements), and  $[yest, xest]$  as outputs.  $L$  is the steady-state filter gain. (In many cases, we do not need  $Kest$  and  $L$ .)  $M$  is the steady-state innovation gain, and  $P$ , and  $Z$  are the steady-state error covariances:

$$P = E\{[x - x(k|k-1)][x - x(k|k-1)]'\} \text{ (Covariance of apriori estimate error)}$$

$$Z = E\{[x - x(k|k)][x - x(k|k)]'\} \text{ (Covariance of aposteriori estimate error)}$$

Here is one example:

```
A=0.85;B=0.15;C=1;D=1;Ts=0.1;  
G=1;H=0;  
s1=ss(A,[B,G],C,[D,H],Ts);  
Q=1; R=0.2;  
[Kest,L,P,M,Z] = kalman(s1,Q,R)
```

MATLAB's response is

a =

x1\_e

x1\_e 0.12853

b =

u1 u2

x1\_e -0.57147 0.72147

c =

x1\_e

y1\_e 0.15121

x1\_e 0.15121

d =

u1 u2

y1\_e 0.15121 0.84879

x1\_e -0.84879 0.84879

Sampling time: 0.1

Discrete-time system.

L =

0.7215

P =

1.1226

M =

0.8488

Z =

0.1698

## 5 Overview over functions in Control System Toolbox

Version 4.0.1 07-Mar-1997

Creation of LTI models.

ss - Create a state-space model.

zpk - Create a zero/pole/gain model.

tf - Create a transfer function model.

dss - Specify a descriptor state-space model.

filt - Specify a digital filter.

set - Set/modify properties of LTI models.  
ltiprops - Detailed help for available LTI properties.

#### Data extraction.

ssdata - Extract state-space matrices.  
zpkdata - Extract zero/pole/gain data.  
tfdata - Extract numerator(s) and denominator(s).  
dssdata - Descriptor version of SSDATA.  
get - Access values of LTI model properties.

#### Model characteristics.

class - Model type ('ss', 'zpk', or 'tf').  
size - Input/output dimensions.  
isempty - True for empty LTI models.  
isct - True for continuous-time models.  
isdt - True for discrete-time models.  
isproper - True for proper LTI models.  
issiso - True for single-input/single-output systems.  
isa - Test if LTI model is of given type.

#### Conversions.

ss - Conversion to state space.  
zpk - Conversion to zero/pole/gain.  
tf - Conversion to transfer function.  
c2d - Continuous to discrete conversion.  
d2c - Discrete to continuous conversion.  
d2d - Resample discrete system or add input delay(s).

#### Overloaded arithmetic operations.

+ and - - Add and subtract LTI systems (parallel connection).  
\* - Multiplication of LTI systems (series connection).  
\ - Left divide --  $\text{sys1} \backslash \text{sys2}$  means  $\text{inv}(\text{sys1}) * \text{sys2}$ .  
/ - Right divide --  $\text{sys1} / \text{sys2}$  means  $\text{sys1} * \text{inv}(\text{sys2})$ .  
' - Pertransposition.  
' - Transposition of input/output map.  
[.] - Horizontal/vertical concatenation of LTI systems.  
inv - Inverse of an LTI system.

Model dynamics.

pole, eig - System poles.

tzero - System transmission zeros.

pzmap - Pole-zero map.

dcgain - D.C. (low frequency) gain.

norm - Norms of LTI systems.

covar - Covariance of response to white noise.

damp - Natural frequency and damping of system poles.

esort - Sort continuous poles by real part.

dsort - Sort discrete poles by magnitude.

pade - Pade approximation of time delays.

State-space models.

rss,drss - Random stable state-space models.

ss2ss - State coordinate transformation.

canon - State-space canonical forms.

ctrb, obsv - Controllability and observability matrices.

gram - Controllability and observability gramians.

ssbal - Diagonal balancing of state-space realizations.

balreal - Gramian-based input/output balancing.

modred - Model state reduction.

minreal - Minimal realization and pole/zero cancellation.

augstate - Augment output by appending states.

Time response.

step - Step response.

impulse - Impulse response.

initial - Response of state-space system with given initial state.

lsim - Response to arbitrary inputs.

ltiview - Response analysis GUI.

gensig - Generate input signal for LSIM.

stepfun - Generate unit-step input.

Frequency response.

bode - Bode plot of the frequency response.

sigma - Singular value frequency plot.

nyquist - Nyquist plot.  
nichols - Nichols chart.  
ltiview - Response analysis GUI.  
evalfr - Evaluate frequency response at given frequency.  
freqresp - Frequency response over a frequency grid.  
margin - Gain and phase margins.

#### System interconnections.

append - Group LTI systems by appending inputs and outputs.  
parallel - Generalized parallel connection (see also overloaded +).  
series - Generalized series connection (see also overloaded \*).  
feedback - Feedback connection of two systems.  
star - Redheffer star product (LFT interconnections).  
connect - Derive state-space model from block diagram description.

#### Classical design tools.

rlocus - Evans root locus.  
rlocfind - Interactive root locus gain determination.  
acker - SISO pole placement.  
place - MIMO pole placement.  
estim - Form estimator given estimator gain.  
reg - Form regulator given state-feedback and estimator gains.

#### LQG design tools.

lqr,dlqr - Linear-quadratic (LQ) state-feedback regulator.  
lqry - LQ regulator with output weighting.  
lqrd - Discrete LQ regulator for continuous plant.  
kalman - Kalman estimator.  
kalmd - Discrete Kalman estimator for continuous plant.  
lqgreg - Form LQG regulator given LQ gain and Kalman estimator.

#### Matrix equation solvers.

lyap - Solve continuous Lyapunov equations.  
dlyap - Solve discrete Lyapunov equations.  
care - Solve continuous algebraic Riccati equations.  
dare - Solve discrete algebraic Riccati equations.

Demonstrations.

ctrldemo - Introduction to the Control System Toolbox.

jetdemo - Classical design of jet transport yaw damper.

diskdemo - Digital design of hard-disk-drive controller.

milldemo - SISO and MIMO LQG control of steel rolling mill.

kalmdemo - Kalman filter design and simulation.