CHAPTER 3

Control Systems

Introduction to Control Systems

MATLAB offers an integrated environment in which you can design control systems. The diagram in Figure 3-1 shows how an engineering problem leads to the development of models and the analysis of experimental data, which in turn lead to the design and simulation of control systems. The subsequent analysis of these systems leads to further modifications of the design, this development loop resulting in rapid prototyping and implementation of effective systems.

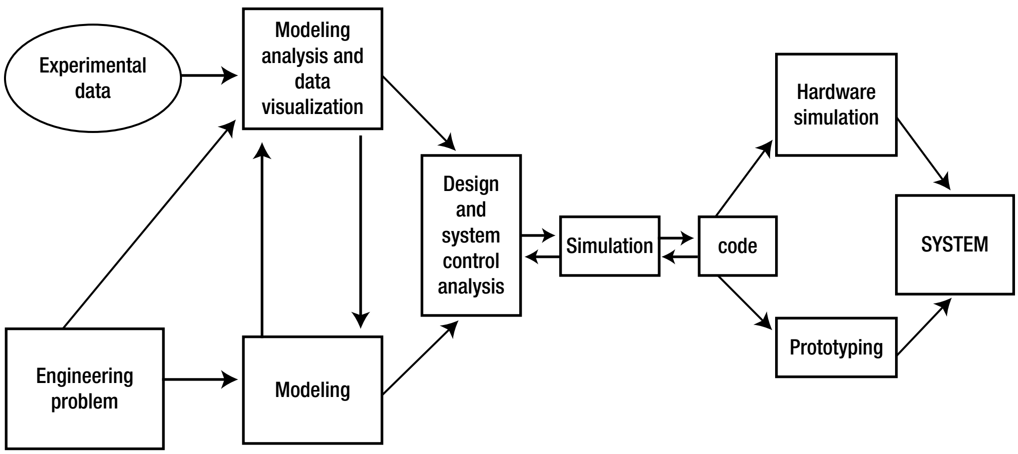
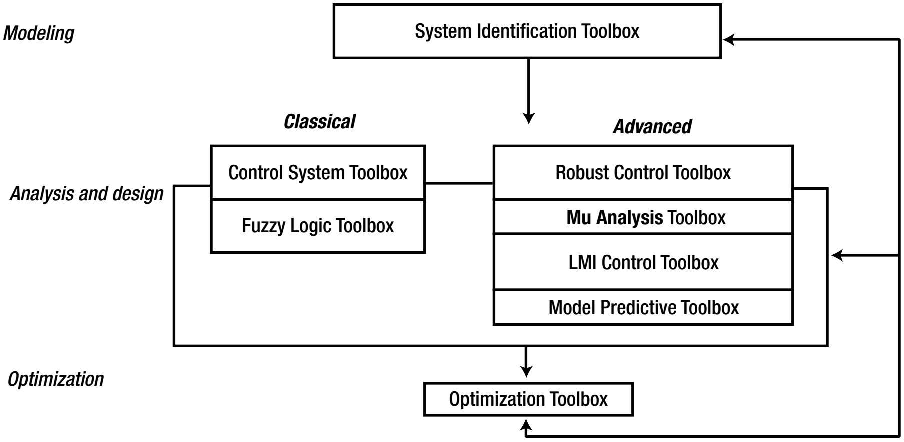


Figure 3-1.

MATLAB provides a high-level platform for technical model generation, data analysis and algorithm development. MATLAB combines comprehensive engineering and mathematics functionality with powerful visualization and animation features, all within a high-level interactive programming language. The MATLAB toolboxes extend the MATLAB environment to incorporate a wide range of classical and modern techniques for the design of control systems, providing cutting edge control algorithms developed by internationally recognized experts.

MATLAB contains more than 600 mathematical, statistical and engineering functions, providing the power of numerical calculation you need to analyze data, develop algorithms and optimize the performance of a system. With MATLAB, you can run fast iterations of designs and compare performances of alternative control strategies. In addition, MATLAB is a high-level programming language that allows you to develop algorithms in a fraction of the time spent in *C*, *C*++ or FORTRAN. MATLAB is open and extendible, you can see the source code, modify algorithms and incorporate existing *C*, *C*++ and FORTRAN programs.

The interactive *Control System Toolbox* tools facilitate the design and adjustment of control systems. For example, you might drag poles and zeros and see immediately how the system reacts (Figure 3-2). In addition, MATLAB provides powerful interactive 2-D and 3-D graphics features showing data, equations, and results (Figure 3-3). It is possible to use a wide range of visualization aids in MATLAB or you can take advantage of the specific control functions which are provided by the MATLAB toolboxes.



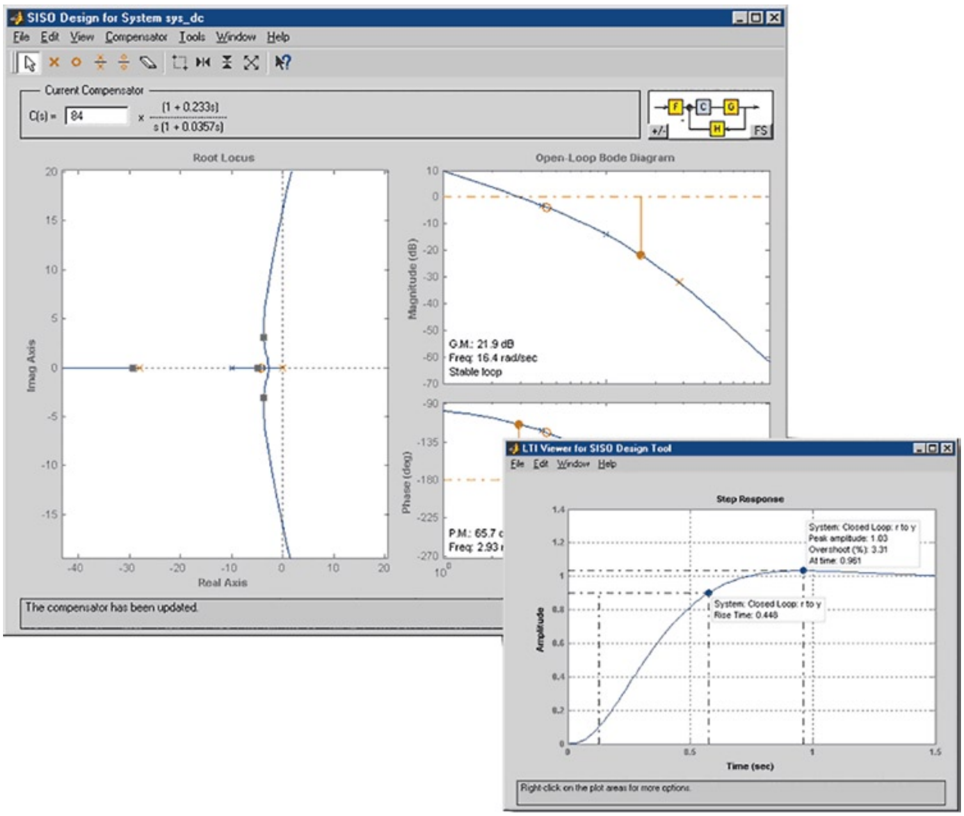


Figure 3-2.

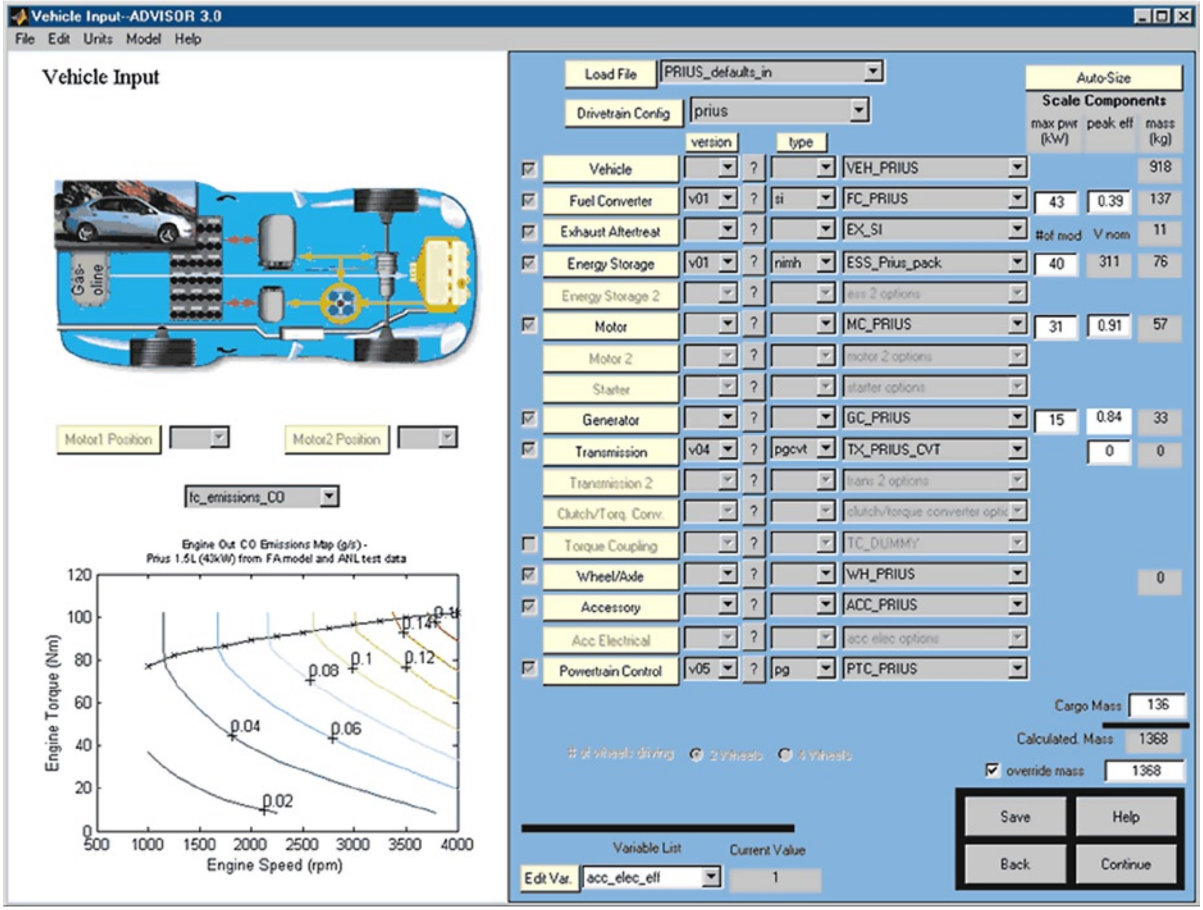


Figure 3-3.

The MATLAB toolboxes include applications written with MATLAB language-specific functionality. The MATLAB control-related toolboxes encompass virtually all of the fundamental techniques of control design, from LQG and root-locus to H and logical diffuse methods. For example, it might add a fuzzy logic control system design using the built-in algorithms of the *Fuzzy Logic Toolbox* (Figure 3-4).

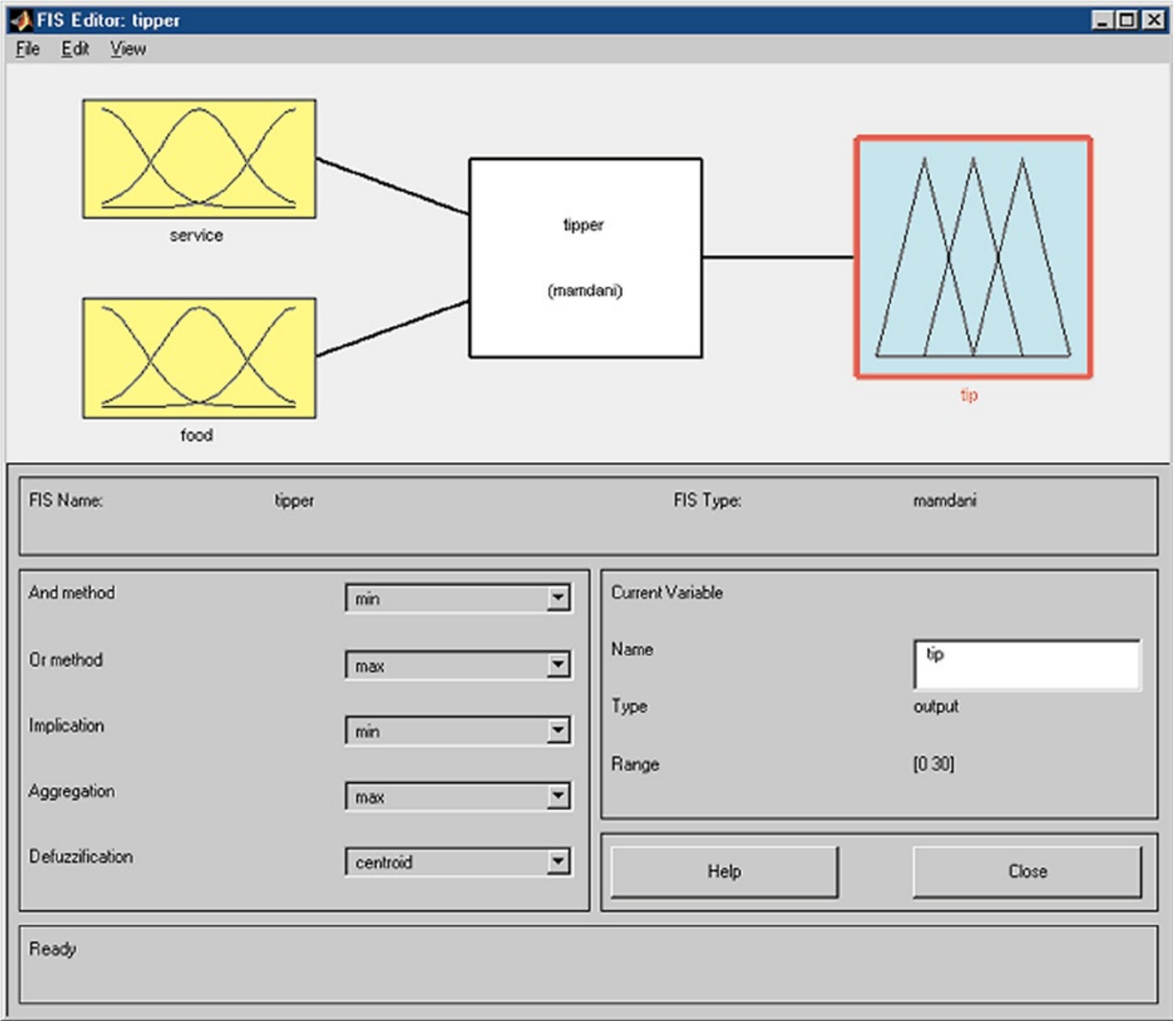


Figure 3-4.

The most important MATLAB toolboxes for control systems can be classified into three families: modeling (*System Identification Toolbox*), classical design and analysis products (*Control System Toolbox* and *Fuzzy Logic Toolbox*), design and advanced analysis products (*Robust Control Toolbox, Mu-Analysis Toolbox, LMI Control Toolbox* and *Model Predictive Toolbox*) and optimization products (*Optimization Toolbox*). The following diagram illustrates this classification.

Control System Design and Analysis: The Control System Toolbox

The *Control System Toolbox* is a collection of algorithms, mainly written as M-files, that implement common techniques of design, analysis, and modeling of control systems. Its wide range of services includes classical and modern methods of control design, including root locus, pole placement and LQG regulator design. Certain graphical user interfaces simplify the typical tasks of control engineering. This toolbox is built on the fundamentals of MATLAB to facilitate specialized control systems for engineering tools.

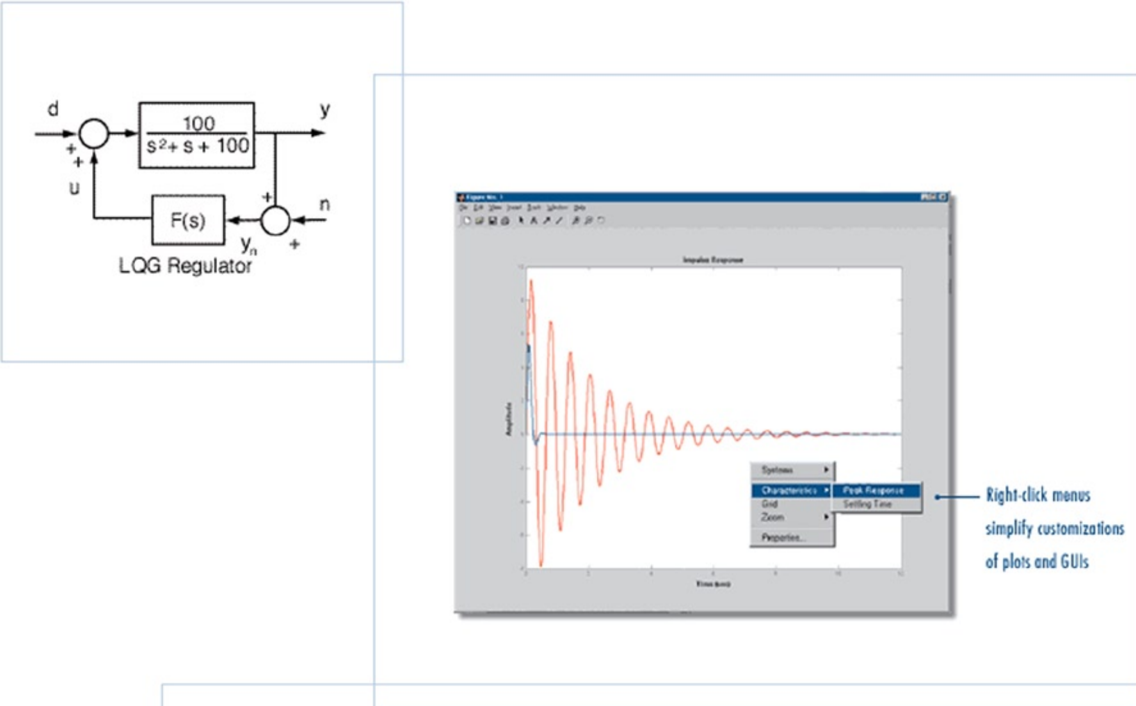
With the *Control System Toolbox* you can create models of linear time-invariant systems (LTI) in transfer function, zero-pole-gain or state-space formats. You can manipulate both discrete-time and continuous-time systems and convert between various representations. You can calculate and graph time response, frequency response and loci of roots. Other functions allow you to perform placement of poles, optimal control and estimates. The *Control System Toolbox* is open and extendible, allowing you to create customized M-files to suit your specific applications.

The following are the key features of the *Control System Toolbox*:

- *LTI Viewer*: An interactive GUI to analyze and compare LTI systems.
- *SISO Design Tool*: An interactive GUI to analyze and adjust single-input/single-output (SISO) feedback control systems.
- *GUI Suite*: Sets preferences and properties to give full control over the display of time and frequency plots.
- *LTI objects*: Structures specialized data to concisely represent model data in transfer function, state-space, zero-pole-gain and frequency response formats.
- MIMO: Support for multiple-input/multiple-output (MIMO) systems, sampled data, continuous-time systems and systems with time delay.
- *Functions and operators to connect LTI models*: Creates complex block diagrams (connections in series, parallel and feedback).
- Support for various methods of converting discrete systems to continuous systems, and vice versa.
- Functions to graphically represent solutions for time and frequency systems and compare various systems with a single command.
- Tools for classical and modern techniques of control design, including root locus analysis, loop shaping, pole placement and LQR/LQG control.

Construction of Models

The *Control System Toolbox* supports the representation of four linear models: state-space models (SS), transfer functions (TF), zero-pole-gain models (ZPK) and frequency data models (FRD). LTI objects are provided for each model type. In addition to model data, LTI objects can store the sample time of discrete-time systems, delays, names of inputs and outputs, notes on the model and many other details. Using LTI objects, you can manipulate models as unique entities and combine them using matrix-type operations. An illustrative example of the design of a simple LQG controller is shown in Figure 3-5. The code extract at the bottom shows how the controller is designed and how the closed-loop system has been created. The plot of the frequency response shows a comparison between the open-loop system (red) and closed loop system (blue).




```
G = ss(tf(100,[1 1 100]))              % state-space plant model
Klqr = lqry(G,10,1)                    % design feedback gain matrix
Kest = kalman(G(:,[1 1]),1, 0.01)      % Kalman estimator design
F = lqgreg(Kest, Klqr)                 % combine regulator and estimator
clsys = feedback(G,F,+1)               % form closed-loop system
impulse(G, 'r', clsys, 'b')            % generate and plot impulse response
```

Figure 3-5.

The *Control System Toolbox* contains commands which analyze and compute model features such as I/O dimensions, poles, zeros and DC gain. These commands apply both to continuous-time and discrete-time models.

Analysis and Design

Some tasks lend themselves to graphic manipulation, while others benefit from the flexibility of the command line. The *Control System Toolbox* is designed to accommodate both approaches, providing a complete set of functions for the design and analysis of models via the command line or GUI.

Graphical Analysis of Models Using the LTI Viewer

The *Control System Toolbox* LTI Viewer is a GUI that simplifies the analysis of linear time-invariant systems (it is loaded by typing `>>ltiview` in the command window). The LTI Viewer is used to simultaneously view and compare the response plots of several linear models. It is possible to generate time and frequency response plots and to inspect key response parameters such as time of ascent, maximum overshooting and stability margins. Using mouse-driven interactions, you can select input and output channels for MIMO systems. The LTI Viewer can simultaneously display up to six different types of plots including step, impulse, *Bode* (magnitude and phase or magnitude only), *Nyquist, Nichols*, sigma, and pole/zero. Right-clicking will reveal an options menu which gives you access to several controls and LTI Viewer Options, including:

- **Plot Type:** Change the type of plot.
- **Systems:** Selects or deselects any of the models loaded in the LTI Viewer.
- **Characteristics:** Displays parameters and key response characteristics.
- **Zoom:** Enlargement and reduction of parts of the plot.
- **Grid:** Add grids to the plots.
- **Properties:** Opens the *Property Editor*, where you can customize attributes of the plot.

In addition to the right-click menu, all the response plots include data markers. These allow you to scan the plot data, identify key data and determine the system font for a given plot. Using the LTI Viewer you can easily graphically represent solutions for one or several systems using step response plots, zero/pole plots and all frequency response plots (*Bode, Nyquist, Nichols* and singular values plots), all in a single window (see Figure 3-6). The LTI Viewer allows you to display important response characteristics in the plots, such as margins of stability, using data markers.

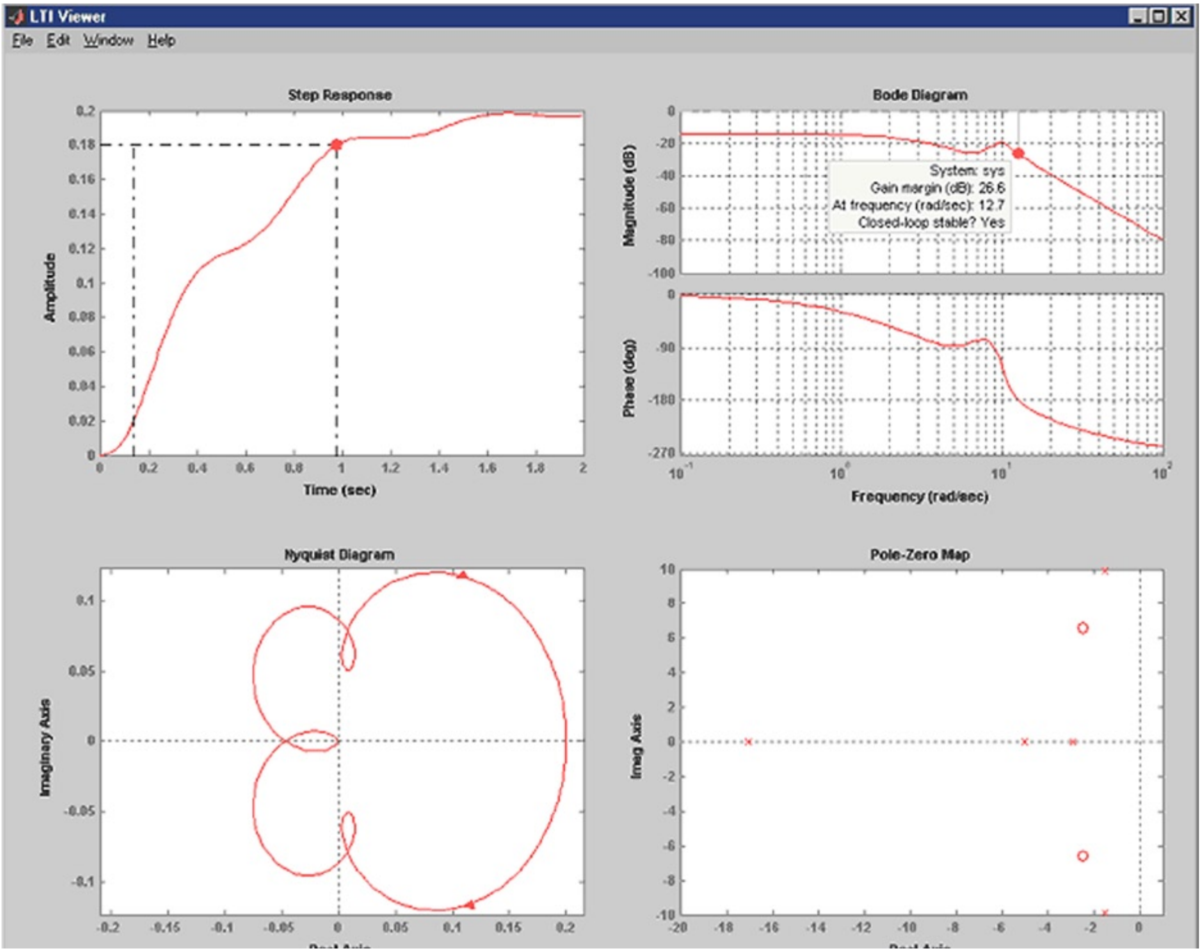


Figure 3-6.

Analysis of Models Using the Command Line

The LTI Viewer is suitable for a wide range of applications where you want a GUI-driven environment. For situations that require programming, custom plots or data unrelated to their LTI models, the *Control System Toolbox* provides command line functions that perform the basic frequency plots and time domain analysis used in control systems engineering. These functions apply to any type of linear model (continuous or discontinuous, SISO or MIMO) or arrays of models.

Compensator Design Using the SISO Design Tool

The *Control System Toolbox* SISO Design Tool is a GUI that allows you to analyze and adjust SISO control feedback systems (loaded by typing `>>sisotool` in the command window). Using the SISO Design Tool, you can graphically adjust the dynamics and the compensator gain using a mixture of root locus and loop shaping techniques. For example, you can use the view of the locus of the roots to stabilize a feedback loop and force a minimum buffer, and use Bode diagrams to adjust bandwidth, gain and phase margins or add a filter *notch* to reject disturbances. The SISO Design GUI can be used for continuous-time and discrete-time time plants. Figure 3-7 shows root locus and Bode diagrams for a discrete-time plant.

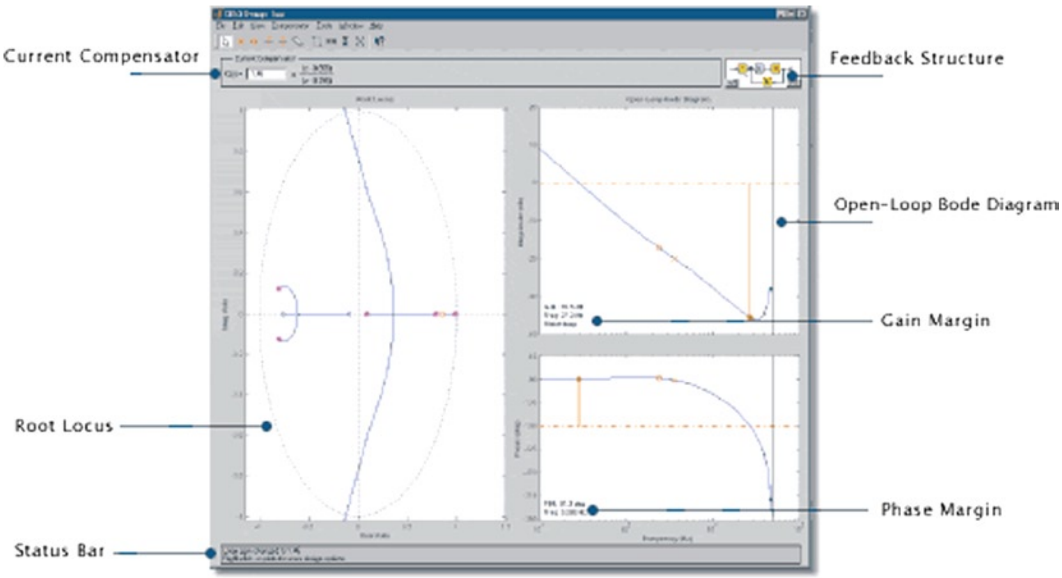


Figure 3-7.

The SISO Design Tool is designed to work closely with the LTI Viewer, allowing you to quickly reiterate a design and immediately see the results in the LTI Viewer. When making a change to the compensator, the LTI Viewer associated with the SISO Design Tool automatically updates the plots of the solution you have chosen. The SISO Design Tool integrates most of the functionality of the *Control System Toolbox* in a single GUI, dynamically linking time, frequency, and pole/zero plots, offering views of complementary themes and design goals, providing graphical changes in Design view and helping to manage the complexity and iterations of the design. The right-click and drop-down menus give you flexibility to design controls with a click of the mouse. In particular, it is possible to view Bode and root locus diagrams, place poles and zeros, add delay/advance networks and notch filters, adjust the compensator parameters graphically with the mouse, inspect closed loop responses (using the LTI Viewer), adjust gain and phase margins and convert models between discrete and continuous time.

Compensator Design Using the Command Line

In addition to the SISO Design Tool, the *Control System Toolbox* provides a number of commands that can be used for a wider range of control applications, including functions for classical SISO design (data buffer, locus of the roots and gain and phase margins) and functions for modern MIMO design (placement of poles, LQR/LQG methods and Kalman filtering). Linear-Quadratic-Gaussian (LQG) control is a modern state-space technique used for the design of optimal dynamic regulators, allowing the balance of benefits of regulation and control costs, taking into account perturbations of the process and measuring noise.

The Control System Toolbox Commands

The *Control System Toolbox* commands can be classified according to their purpose as follows:

General

Ctrlpref: Opens a GUI which allows you to change the *Control System Toolbox* preferences (see Figure 3-8).

Creation of linear models

tf: Creates a transfer function model
zpk: Creates a zero-pole-gain model
ss: Creates a state-space model

dss: Creates a descriptor state-space model
frd: Creates a frequency-response data model
set: Locates and modifies properties of LTI models

Data extraction

tfdata: Accesses transfer function data (in particular extracts the numerator and denominator of the transfer function)
zpkdata: Accesses zero-pole-gain data
ssdata: Accesses state-space model data
get:Accesses properties of LTI models

Conversions

s: Converts to a state-space model
zpk: Converts to a zero-pole-gain model
tf: Converts to a transfer function model
frd: Converts to a frequency-response data model
c2d: Converts a model from continuous to discrete time
d2c: Converts a model from discrete to continuous time
d2d: Resamples a discrete time model

System interconnection

append: Groups models by appending their inputs and outputs
parallel: Parallel connection of two models
series: Series connection of two models
feedback: Connection feedback of two systems
lft: Generalized feedback interconnection of two models
connect: Block diagram interconnection of dynamic systems

(*continued*)

Dynamic models

iopzmap: Plots a pole-zero map for input/output pairs of a model
bandwidth: Returns the frequency-response bandwidth of the system
pole: Computes the poles of a dynamic system
zero: Returns the zeros and gain of a SISO dynamic system
pzmap: Returns a pole-zero plot of a dynamic system
damp: Returns the natural frequency and damping ratio of the poles of a system
dcgain: Returns the low frequency (DC) gain of an LTI system
norm: Returns the norm of a linear model
covar: Returns the covariance of a system driven by white noise

Time-domain analysis

ltiview: An LTI viewer for LTI system response analysis
step: Produces a step response plot of a dynamic system
impulse: Produces an impulse response plot of a dynamic system
initial: Produces an initial condition response plot of a state-space model
lsim: Simulates the time response of a dynamic system to arbitrary inputs

Frequency-domain analysis

ltiview: An LTI viewer for LTI system response analysis
bode: Produces a Bode plot of frequency response, magnitude and phase of frequency response
sigma: Produces a singular values plot of a dynamic system
nyquist: Produces a Nyquist plot of frequency response
nichols: Produces a Nichols chart of frequency response
margin: Returns gain margin, phase margin, and crossover frequencies
allmargin: Returns gain margin, phase margin, delay margin and crossover frequencies
freqresp: Returns frequency response over a grid

Classic design

sisotool: Interactively design and tune SISO feedback loops (technical *root locus* and *loop shaping*)
rlocus: Root locus plot of a dynamic system

Pole placement

place: MIMO pole placement design
estim: Forms a state estimator given estimator gain
reg: Forms a regulator given state-feedback and estimator gains

LQR/LQG design

lqr: Linear quadratic regulator (LQR) design
dlqr: Linear-quadratic (LQ) state-feedback regulator for a discrete-time state-space system
lqry: Linear-quadratic (LQ) state-feedback regulator with output weighting
lqrd: Discrete linear-quadratic (LQ) regulator for a continuous plant
Kalman: Kalman estimator
kalmd: Discrete Kalman estimator for a continuous plant

(*continued*)

State-space models

rss: Generates a random continuous test model
drss: Generates a random discrete test model
ss2ss: State coordinate transformation for state-space models
ctrb: Controllability matrix
obsv: Observability matrix
gram: Control and observability gramians
minreal: Minimal realization or pole-zero cancelation
ssbal: Balance state-space models using a diagonal similarlity transformation
balreal: Gramian-based input/output balancing of state-space realizations
modred: Model order reduction

Models with time delays

totaldelay: Total combined input/output delay for an LTI model
delay2z: Replaces delays of discrete-time TF, SS, or ZPK models by poles at z=0, or replaces delays of FRD models [Note: in more recent versions of MATLAB, *delay2z* has been replaced with *absorbDelay*.]
pade: Padé approximation of a model with time delays

Matrix equation solvers

lyap: Solves continuous-time Lyapunov equations
dlyap: Solves discrete-time Lyapunov equations
care: Solves continuous-time algebraic Riccati equations
dare: Solves discrete-time algebraic Riccati equations

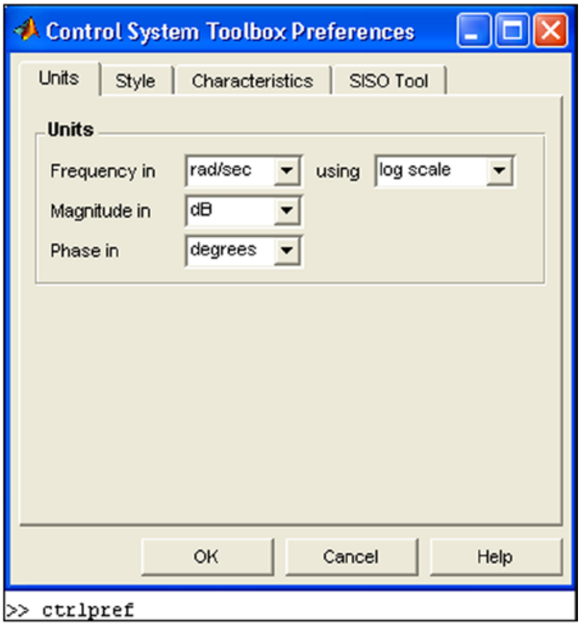


Figure 3-8.

The following sections present the syntax of the above commands, appropriately grouped into the previously mentioned categories.

LTI Model Commands

Command	Description
sys = drss(n, m, p)	*Generates a random discrete-time state-space model of order n with m inputs and p outputs.*
sys = drss(n, p)	*Equivalent to drss(n,m,p) with m = 1.*
sys = drss(n)	*Equivalent to drss(n,m,p) with n = m = 1.*
sys = drss(n,m,p,s1,...sn)	*Generates an array of state-space models.*
dss (A,B,C,D,E)	*Creates the continuous-time descriptor state-space model:* $E\frac{dx}{dt} = Ax + Bu$ $y = Cx + Du$
dss (A,B,C,D,E, Ts)	*Creates the discrete -time descriptor state-space model (with sample time Ts in seconds):* $Ex[n+1] = Ax[n]Bu[n]$ $y[n] = Cx[n] + Du[n]$
dss (A,B,C,D,E, ltisys)	*Creates the descriptor state-space model with generic LTI properties inherited from the model ltisys.*
dss (A,B,C,D,E, p1, p2, v1, v2,...)	*Creates the continuous-time descriptor state-space model with generic LTI properties given by the propery/value pairs (pi, vi).*
dss (A,B,C,D,E, Ts, p2, p1, v1, v2,...)	*Creates the discrete-time descriptor state-space model (with sample time Ts in seconds) with generic LTI properties given by the property/value pairs (pi, vi).*
sys = filt(num,den)	*Creates a discrete transfer function in the DSP format with numerator num and denominator den.*
sys = filt(num,den,Ts)	*Creates a discrete transfer function in the DSP format with numerator num, denominator den and sample time Ts in seconds.*
sys = filt (M)	*Specifies a static filter with gain matrix M.*
sys = filt(num,den, p1,v1,p2,v2,...)	*Creates a discrete transfer function in the DSP format with numerator num and denominator den and generic LTI properties given by the property/value pairs (pi, vi).*
sys = filt(num,den,Ts, p1,v1,p2,v2,...)	*Creates a discrete transfer function in the DSP format with numerator num and denominator den, sample time Ts in seconds, and generic LTI properties given by the property/value pairs (pi, vi).*

(*continued*)

Command	Description
sys = frd(r,f)	*Creates a frequency-response data (FRD) model from the frequency response data stored in r, where f represents the underlying frequencies for the frequency response data.f*
sys = frd(r,f,Ts)	*Creates a frequency-response data model with scalar sample time Ts in seconds.*
sys = frd	*Creates an empty frequency-response data model.*
sys = frd(r,f,ltisys)	*Creates a frequency-response data model object with generic LTI properties inherited from the model ltisys.*
sysfrd = frd(sys,f)	*Converts a TF, SS, or ZPK model to an FRD model with frequency samples given by f.*
sysfrd = frd(sys,f,u)	*Converts a TF, SS, or ZPK model to an FRD model with frequency samples given by f in units specified by the string u (for example 'rad/s' or 'Hz').*
[r,f] = frdata(sys)	*Returns the response data and frequency samples of the FRD model sys.*
[r,f,Ts] = frdata(sys)	*Returns the response data, frequency samples and sample time of the FRD model sys.*
[r,f] = frdata(sys,'v')	*Returns the response data and frequency samples of the FRD model sys directly as column vectors.*
get(sys)	*Displays all the properties and values of the FRD model sys.*
get(sys, 'P')	*Displays the current value of the property name P of the FRD model sys.*
sys = rss(n,m,p)	*Generates a random continuous test model of order n with m inputs and p outputs.*
sys = rss(n,p)	*Equivalent to rss(n,m,p) with m = 1.*
sys = rss(n)	*Equivalent to rss(n,m,p) with n = m = 1.*
sys = rss(n,m,p,s1,...sn)	*Generates an s1×...×sn array of nth order state-space models with m inputs and p outputs.*
set(sys,'P',V)	*Assigns the value V to the given property of the LTI model sys.*
set(sys,'P1',V1,'P2',V2,...)	*Allocates values V1,...,VN to the properties P1,...,PN of the LTI model sys.*
set(sys,'P')	*Returns the permissible values for the property P.*
set(sys)	*Displays all sys properties and their values.*
ss (A,B,C,D,E).	*Creates the continuous-time state-space model:* $$E\frac{dx}{dt} = Ax + Bu$$ $$y = Cx + Du$$
ss (A,B,C,D,E, Ts)	*Creates the discrete-time state-space model (with sample time Ts in seconds):* $$Ex[n+1] = Ax[n]Bu[n]$$ $$y[n] = Cx[n] + Du[n]$$
ss (D)	*Equivalent to* `ss([],[],[],D)`.

(*continued*)

Command	Description
ss (A,B,C,D,E, ltisys)	*Creates a state-space model with generic LTI properties inherited from the model ltisys.*
ss (A,B,C,D,E, p1, p2, v1, v2,...)	*Creates a state-space model with properties given by the property/value pairs (pi, vi).*
ss (a, b, c, d, e, Ts, p2, p1, v1, v2,...)	*Creates a discrete state-space model with properties given by the property/value pairs (pi, vi)) and sample time Ts in seconds.*
sys_ss = ss(sys)	*Converts the (TF or ZPK) model sys to a state-space model.*
sys_ss = ss(sys,'minimal')	*produces a state-space realization with no uncontrollable or unobservable states.*
[A,B,C,D] = ssdata(sys)	*Extracts the model data [A, B, C, D] from the state-space model sys.*
[A,B,C,D,Ts] = ssdata(sys)	*Extracts the model data [A, B, C, D] and the sample time Ts from the state-space model sys.*
[A,B,C,D] = dssdata(sys)	*Extracts the model data [A, B, C, D] from the descriptor state-space model sys.*
[A,B,C,D,Ts] = dssdata(sys)	*Extracts the model data [A, B, C, D] and the sample time Ts from the descriptor state-space model sys.*
sys = tf(num,den)	*Creates a continuous-time transfer function with specified numerator and denominator.*
sys = tf(num,den,Ts)	*Creates a discrete-time transfer function with specified numerator and denominator and sample of Ts time in seconds.*
sys = tf (M)	*Creates a static gain M (matrix or scalar).*
sys = tf(num,den,ltisys)	*Creates a transfer function with specified numerator and denominator and generic properties inherited from the LTI model ltisys.*
sys = tf(num,den, p1,v1,p2,v2,...)	*Creates a continuous-time transfer function with specified numerator and denominator and with properties given by the property/value pairs (pi, vi).*
sys = tf(num,den,Ts, p1,v1,p2,v2,...)	*Creates a discrete-time transfer function with specified numerator and denominator, sample time Ts in seconds, and properties given by the property/value pairs (pi, vi).*
s = tf('s')	*Specifies a TF model using a rational function in the Laplace variable s.*
z = tf('z',Ts)	*Specifies a TF model with sample time Ts using a rational function in the discrete-time variable z.*
tfsys = tf(sys)	*Converts a (TF or ZPK) model sys to a transfer function.*
tfsys = tf(sys,'inv')	*Converts a (TF or ZPK) model sys to a transfer function using investment formulas.*
[num,den] = tfdata(sys)	*Returns the numerator and denominator for type TF, SS, or ZPK sys transfer function models.*
[num,den] = tfdata(sys,'v')	*Returns the numerator and denominator as row vectors.*
[num,den,Ts] = tfdata(sys)	*In addition to the above, also returns sample time Ts.*
TD = totaldelay (sys)	*Gives the combined total input/output lag of the LTI model sys*

(*continued*)

Command	Description
sys = zpk (z, p, k)	*Creates a continuous-time zero-pole-gain model with zeros z, poles p and gains k.*
sys = zpk (z, p, k, Ts)	*Creates a discrete-time zero-pole-gain model with zeros z, poles p, gains k and sample time Ts in seconds.*
sys = zpk(M)	*Specifies a static gain M.*
sys = zpk(z,p,k,ltisys)	*Creates a continuous-time zero-pole-gain model with zeros z, poles p and gains k with generic properties inherited from the LTI model ltisys.*
sys=zpk(z,p,k,p1,v1,p2,v2,...)	*Creates a continuous-time zero-pole-gain model with zeros z, poles p and gains k and properties given by the property/value pairs (pi, vi).*
sys=zpk(z,p,k,Ts,p1,v1,p2,v2,..)	*Creates a discrete-time zero-pole-gain model with zeros z, poles p, gains k and sample time Ts, and properties given by the property/value pairs (pi, vi).*
sys = zpk('s')	*Specifies a continuous-time zero-pole-gain model using a rational function in the Laplace variable s.*
sys = zpk('z',Ts)	*Specifies a discrete-time zero-pole-gain model using a rational function in the discrete-time variable z.*
zsys = zpk(sys)	*Converts an LTI model sys into a zero-pole-gain model.*
zsys = zpk(sys,'inv')	*Converts an LTI model sys into a zero-pole-gain model using investment formulas.*
[z,p,k] = zpkdata(sys)	*Returns the zeros z, poles p and gains k of the model sys.*
[z,p,k] = zpkdata(sys,'v')	*Returns the zeros z, poles p and gains k of the model sys as column vectors.*
[z,p,k,Ts,Td] = zpkdata(sys)	*Returns in addition to the above the sample time Ts and the input lag Td.*

As a first example, we generate a random discrete LTI system with three states, two inputs and two outputs.

```
>> sys = drss(3,2,2)

a =
                 x1          x2          x3
     x1   -0.048856     0.40398     0.23064
     x2    0.068186     0.35404    -0.40811
     x3    -0.46016   -0.089457   -0.036824

b =
                 u1          u2
     x1    -0.43256     0.28768
     x2           0     -1.1465
     x3     0.12533      1.1909

c =
                 x1          x2          x3
     y1      1.1892     0.32729    -0.18671
     y2   -0.037633     0.17464     0.72579

d =
                 u1          u2
     y1           0     -0.1364
     y2      2.1832           0
```

```
Sampling time: unspecified
Discrete-time model.
>>
```

In the following example, we create the model

$$5\frac{dx}{dt} = x + 2u$$

$$y = 3x + 4u$$

with a gap of 0.1 seconds and tagged as '*voltage*' entry.

```
>> sys = dss(1,2,3,4,5,0.1,'inputname','voltage')

a =
                    x1
          x1         1
b =
               voltage
          x1         2

c =
                    x1
          y1         3


d =
               voltage
          y1         4

e =
                    x1
          x1         5

Sampling time: 0.1
Discrete-time model.
```

The example below creates the following two-input digital filter:

$$H\left(z^{-1}\right) = \left[\frac{1}{1+z^{-1}+2z^{-2}} \quad \frac{1+0.3z^{-1}}{5+2z^{-1}}\right]$$

specifying time displays and channel entries *'channel1'* and *'channel2'* :

```
>> num = {1 , [1 0.3]}
den = {[1 1 2] ,[5 2]}
H = filt(num,den,'inputname',{'channel1' 'channel2'})

NUM =

[1.00] [double 1 x 2]
```

```
den =

[double 1 x 3]    [double 1 x 2]

Transfer function from input "channel1" to output:

       1
-----------------
1 + z^-1 + 2 z^-2

Transfer function from input "channel2" to output:

1 + 0.3 z ^ - 1
--------------
 5 + 2 z ^ - 1

Sampling time: unspecified
```

Next we create a SISO FRD model.

```
>> freq = logspace(1,2);
resp = .05*(freq).*exp(i*2*freq);
sys = frd(resp,freq)

From input 1 to:

  Frequency(rad/s)        output 1
  ----------------   ------------------
     10.000000        0.204041+0.456473i
     10.481131       -0.270295+0.448972i
     10.985411       -0.549157+0.011164i
     11.513954       -0.293037-0.495537i
     12.067926        0.327595-0.506724i
     12.648552        0.623904+0.103480i
     13.257114        0.124737+0.651013i
     13.894955       -0.614812+0.323543i
     14.563485       -0.479139-0.548328i
     15.264180        0.481814-0.591898i
     15.998587        0.668563+0.439215i
     16.768329       -0.438184+0.714799i
     17.575106       -0.728874-0.490870i
     18.420700        0.602513-0.696623i
     19.306977        0.588781+0.765007i
         .
         .
         .
     86.851137       -2.649156-3.440897i
     91.029818        4.498503-0.692487i
     95.409548       -3.261293+3.481583i
     100.000000       2.435938-4.366486i

 Continuous-time frequency response data model.
```

Now we define an FRD model and its data is returned.

```
>> freq = logspace(1,2,2);
resp = .05*(freq).*exp(i*2*freq);
sys = frd(resp,freq);
[resp,freq] = frdata(sys,'v')

resp =
          0.20
          2.44
freq =
         10.00
        100.00
```

The following example creates a 2-output/1-input transfer function:

$$H(p)=\begin{bmatrix} \dfrac{p+1}{p^2+2p+2} \\ \dfrac{1}{p} \end{bmatrix}$$

```
>> num = {[1 1] ; 1}
den = {[1 2 2] ; [1 0]}
H = tf(num,den)

NUM =

[double 1 x 2]
[1.00]

den =

[double 1 x 3]
[1x2 double]
Transfer function from input to output...
         s + 1
#1:  -------------
    s ^ 2 + 2 s + 2

     1
#2:  -
     s
```

The following example computes the transfer function for the following state-space model:

$$A=\begin{bmatrix} -2 & -1 \\ 1 & -2 \end{bmatrix},\ B=\begin{bmatrix} 1 & 1 \\ 2 & -1 \end{bmatrix},\ C=\begin{bmatrix} 1 & 0 \end{bmatrix},\ D=\begin{bmatrix} 0 & 1 \end{bmatrix}$$

```
>> sys = ss([-2 -1;1 -2],[1 1;2 -1],[1 0],[0 1])
tf(sys)

a =
x1          x2
x1          -2          -1
x2           1          -2

b =
             u1          u2
x1           1           1
x2           2          -1

c =
            x1          x2
y1           1           0

d =
            u1          u2
y1           0           1

Continuous-time model.

Transfer function from input 1 to output:

s - 2.963e-016
--------------
s^2 + 4 s + 5

Transfer function from input 2 to output:

s ^ 2 + 5 s + 8
-------------
s ^ 2 + 4 s + 5
```

The following example specifies two discrete-time transfer functions:

$$g(z) = \frac{z+1}{z^2+2z+3} \qquad h(z^{-1}) = \frac{1+z^{-1}}{1+2z^{-1}+3z^{-2}} = zg(z)$$

```
>> g = tf([1 1],[1 2 3],0.1)

Transfer function:

     z + 1
-------------
z^2 + 2 z + 3

Sampling time: 0.1
```

```
>> h = tf([1 1],[1 2 3],0.1,'variable','z^-1')

Transfer function:

       1 + z^-1
-------------------
1 + 2 z^-1 + 3 z^-2

Sampling time: 0.1
```

We now specify the zero-pole-gain model associated with the transfer function:

$$H(z)=\begin{bmatrix} \dfrac{1}{z-0.3} \\ \dfrac{2(z+0.5)}{(z-0.1+j)(z-0.1-j)} \end{bmatrix}$$

```
>> z = {[] ; -0.5}
p = {0.3 ; [0.1+i 0.1-i]}
k = [1 ; 2]
H = zpk(z,p,k,-1)

z =

[]
[-0.5000]

p =

[    0.3000]
[1x2 double]

k =

1
2

Zero/pole/gain from input to output...

        1
#1:  -------
     (z-0.3)

           2 (z+0.5)
#2:  -------------------
     (z^2 - 0.2z + 1.01)

Sampling time: unspecified
```

In the following example the transfer function `tf([-10 20 0],[1 7 20 28 19 5])` is converted into zero-pole-gain format.

```
>> h = tf([-10 20 0],[1 7 20 28 19 5])

Transfer function:

             -10 s^2 + 20 s
----------------------------------------
s^5 + 7 s^4 + 20 s^3 + 28 s^2 + 19s + 5

>> zpk(h)

Zero/pole/gain:

      -10 s (s-2)
----------------------
(s) ^ 3 (s ^ 2 + 4s + 5)
```

Model Feature Commands

Command	Description
str = class(object)	*Displays a string describing which type of model* object *is ('tf,' 'zpk,' 'ss,' or 'frd').*
hasdelay(sys)	*Returns 1 if the LTI model sys has input, output, input/output or internal delays, and returns 0 otherwise.*
k= isa(obj,'class')	*Returns 1 if the object is of the given class.*
boo = isct(sys)	*Returns 1 if the LTI model sys is continuous.*
boo = isdt(sys)	*Returns 1 if the LTI model sys is discrete.*
boo = isempty(sys)	*Returns 1 if the LTI model sys has no input or output.*
boo = isproper(sys)	*Returns 1 if the LTI model sys is proper.*
boo = issiso(sys)	*Returns 1 if the LTI model sys is SISO.*
n = ndims(sys)	*Returns the number of dimensions in the LTI model or model array* sys.
size(sys)	*Displays the number of inputs/outputs of sys.*
d = size(sys)	*Assigns the number of inputs/outputs of sys to d.*
Ny = size(sys,1)	*Returns the number of outputs of sys.*
Nu = size(sys,2)	*Returns the number of inputs of sys.*
Sk = size(sys,2+k)	*Returns the length of the k-th dimension of the array when sys is an LTI array.*
Ns = size(sys,'order')	*Returns the order of the (TS, SS, or ZPK) model sys.*
Nf = size(sys,'frequency')	*Returns the frequency of the FRD model sys.*

Model Conversion Commands

Command	Description
sysd = c2d(sys,Ts)	*Converts a continuous model sys to a discrete model sysd using zero-order hold on the inputs and a sample time of Ts seconds.*
sysd = c2d(sys,Ts,method)	*Converts a continuous model sys to a discrete model sysd using zero-order hold on the inputs and a sample time of Ts seconds using the specified method of discretization. The method can be zero-order hold (zoh), triangle approximation (foh), impulse invariant discretization (impulse), Bilinear (Tustin) (tustin) or zero-pole matching (matched).*
[sysd, G] = c2d(sys,Ts,method)	*In addition to the above, returns a matrix G that maps the continuous initial conditions x0 and u0 of the state-space model sys to the discrete-time initial state vector x[0]. The possible methods of discretization are descxribed above.*
sys = chgFreqUnit(sys,units)	*Changes units of the frequency points in sys to new units given by units.*
sysc = d2c(sysd)	*Converts a discrete model sysd to a continuous model sysc using zero-order hold on the inputs.*
sysc = d2c(sysd,method)	*Converts a discrete model sysd to a continuous model sysc using the conversion method given by* method. *The possible methods of conversion are zoh, foh, tustin and matched (see above).*
sys1 = d2d(sys,Ts)	*Resamples the discrete-time model sys to produce an equivalent discrete-time model sys1 with new sample time Ts.*
sys = delay2z(sys)	*Replaces delays of discrete-time TF, SS or ZPK models by poles at z=0, or replaces delays of FRD models by phase shift. [Note: more recent versions of MATLAB have replaced delay2z by absorbDelay.]*
sys = frd(r,f)	*Creates an FRD model sys from the frequency response data stored in the array r. The vector f represents the underlying frequencies for the frequency response data.*
sys = frd(r,f,Ts)	*Creates a discrete-time FRD model with sample time Ts in seconds.*
sys = frd	*Creates an empty FRD model.*
sys = frd(r,f,ltisys)	*Creates an FRD model which inherits the generic properties of the LTI model ltisys.*
sysfrd = frd(sys,f)	*Converts a TF, SS or ZPK model to an FRD model with frequencies f.*
sysfrd = frd(sys,f,units)	*Converts a TF, SS or ZPK model to an FRD model with frequencies f specifying the units ('rad/s' or 'Hz').*

(*continued*)

Command	Description
[num, den] = pade(T,N)	*Returns the Padé approximation of order N of the continuous-time I/O delay exp(–sT) in transfer function form. The row vectors num and den contain the numerator and denominator coefficients in descending powers of s. Both are Nth-order polynomials.*
pade(T,N)	*Plots the step and phase responses of the Nth-order Padé approximation and compares them with the exact responses of the model with I/O delay T.*
sysx = pade(sys,N)	*Produces a delay-free approximation sysx of the continuous delay system sys. All delays are replaced by their Nth-order Padé approximation.*
sysx = pade(sys,Nu,Ny,NINT)	*Specifies independent approximation orders for each input, output, and I/O or internal delay. Here NU, NY and NINT are integer arrays: NU is the vector of approximation orders for the input channel; NY is the vector of approximation orders for the output channel; NINT is the approximation order for I/O delays (TF or ZPK models) or internal delays (state-space models).*
sys = reshape(sys,s1,s2,...,sk) **sys = reshape(sys,[s1s2... sk])**	*Reshapes the LTI model sys to an array of LTI models.*
[r, p, k] = residue(b,a)	*Finds the residues, poles, and direct term of a partial fraction expansion of the ratio of two polynomials, b(s) and a(s), where b and a are the vectors listing the numerator and denominator coefficients, respectively.*
[b,a] = residue(r,p,k)	*Converts the partial fraction expansion back to the polynomials with coefficients in b and a.*
sys = ss(A,B,C,D,E).	*Creates the continuous-time state-space model:* $$E\frac{dx}{dt} = Ax + Bu$$ $$y = Cx + Du$$
sys = ss(A,B,C,D,E,Ts)	*Creates the discrete-time state-space model (with sample time Ts in seconds):* $$Ex[n+1] = Ax[n]Bu[n]$$ $$y[n] = Cx[n] + Du[n]$$
sys = ss(A,B,C,D,E,ltisys)	*Creates a continuous-time state-space model with generic properties inherited from the LTI model ltisys.*
sys = ss(A,B,C,D,E,p1,p2,v1,v2,...)	*Creates a continuous-time state-space model with properties given by the property/value pairs (pi, vi).*
sys= ss(A,B,C,D,E,Ts,p1,v1,p2,v2,...)	*Creates a discrete-time state-space model with sample time Ts and properties given by the property/value pairs (pi, vi).*
sys_ss = ss(sys)	*Converts the (TF or ZPK) model sys to a state-space model.*
sys_ss = ss(sys,'minimal')	*Produces a state-space realization with no uncontrollable or unobservable states*
sys = tf(num,den)	*Creates a continuous-time transfer function with specified numerator and denominator.*
sys = tf(num,den,Ts)	*Creates a discrete-time transfer function with specified numerator and denominator and sample time of Ts seconds.*

(*continued*)

Command	Description
sys = tf(M)	*Creates a static gain M (matrix or scalar).*
sys = tf(num,den,ltisys)	*Creates a transfer function with specified numerator and denominator and generic properties inherited from the LTI model ltisys.*
sys = tf(num,den,p1,v1,p2,v2,...)	*Creates a continuous-time transfer function with specified numerator and denominator and with properties given by the property/value pairs (pi, vi).*
sys = tf(num,den,Ts,p1,v1,p2,v2,...)	*Creates a discrete-time transfer function with specified numerator and denominator, sample time Ts in seconds, and properties given by the property/value pairs (pi, vi).*
s = tf('s')	*Specifies a TF model using a rational function in the Laplace variable s.*
z = tf('z',Ts)	*Specifies a TF model with sample time Ts using a rational function in the discrete-time variable z.*
tfsys = tf(sys)	*Converts a (TF or ZPK) model sys to a transfer function.*
tfsys = tf(sys,'inv')	*Converts a (TF or ZPK) model sys to a transfer function using investment formulas.*
sys = zpk(z,p,k)	*Creates a continuous-time zero-pole-gain model with zeros z, poles p and gains k.*
sys = zpk(z,p,k,Ts)	*Creates a discrete-time zero-pole-gain model with zeros z, poles p, gains k and sample time Ts in seconds.*
sys = zpk(M)	*Specifies a static gain M.*
sys = zpk(z,p,k,ltisys)	*Creates a continuous-time zero-pole-gain model with zeros z, poles p and gains k with generic properties inherited from the LTI model ltisys.*
sys = zpk(z,p,k,p1,v1,p2,v2,...)	*Creates a continuous-time zero-pole-gain model with zeros z, poles p and gains k and properties given by the property/value pairs (pi, vi).*
sys = zpk(z,p,k,Ts,p1,v1,p2,v2,..)	*Creates a discrete-time zero-pole-gain model with zeros z, poles p, gains k and sample time Ts, and properties given by the property/value pairs (pi, vi).*
sys = zpk('s')	*Specifies a continuous-time zero-pole-gain model using a rational function in the Laplace variable s.*
sys = zpk('z',Ts)	*Specifies a discrete-time zero-pole-gain model using a rational function in the discrete-time variable z.*
zsys = zpk(sys)	*Converts an LTI model sys into a zero-pole-gain model.*
zsys = zpk(sys,'inv')	*Converts an LTI model sys into a zero-pole-gain model using investment formulas.*

As a first example, we consider the system:

$$H(s) = \frac{s-1}{s^2+4s+5}$$

with input lag *Td* = 0.35 seconds. The system is discretized using triangular approximation with sampling time *Ts* = 0.1 sec.

```
>> H = tf([1 -1],[1 4 5],'inputdelay',0.35)
```

```
Transfer function:

                    s - 1
exp(-0.35*s) * -------------
               s^2 + 4s + 5

>> Hd = c2d(H,0.1,'foh')

Transfer function:

          0.0115 z^3 + 0.0456 z^2 - 0.0562z - 0.009104
z^(-3) * ---------------------------------------------
                 z^3 - 1.629 z^2 + 0.6703z

Sampling time: 0.1
```

If we want to compare the step response and its discretization (see Figure 3-9) we can use the following command:

```
>> step(H,'-',Hd,'--')
```

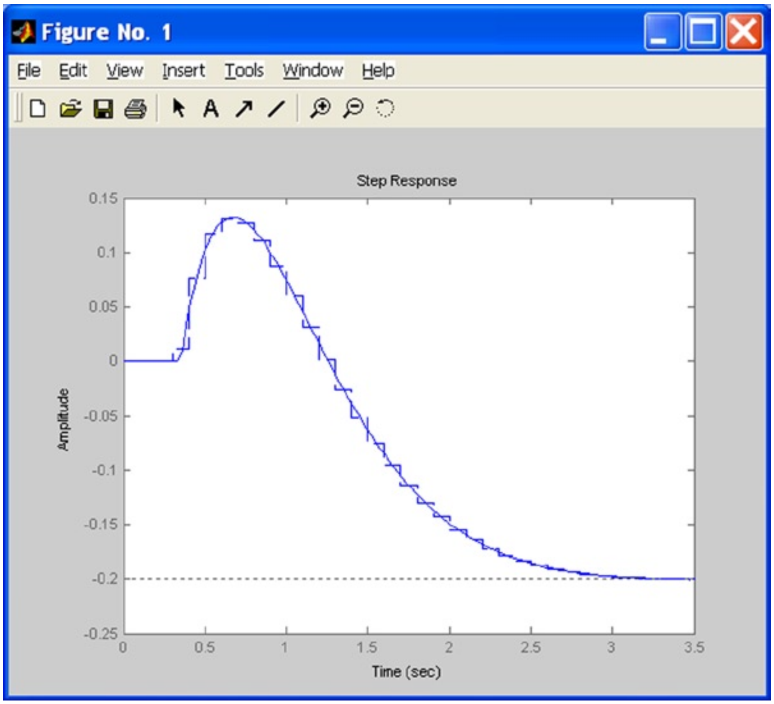


Figure 3-9.

The next example computes a Padé approximation of third order with I/O lag 0.1 seconds and compares the time and frequency response with its approximation (Figure 3-10).

```
>> pade(0.1,3)
Step response of 3rd-order Pade approximation
```

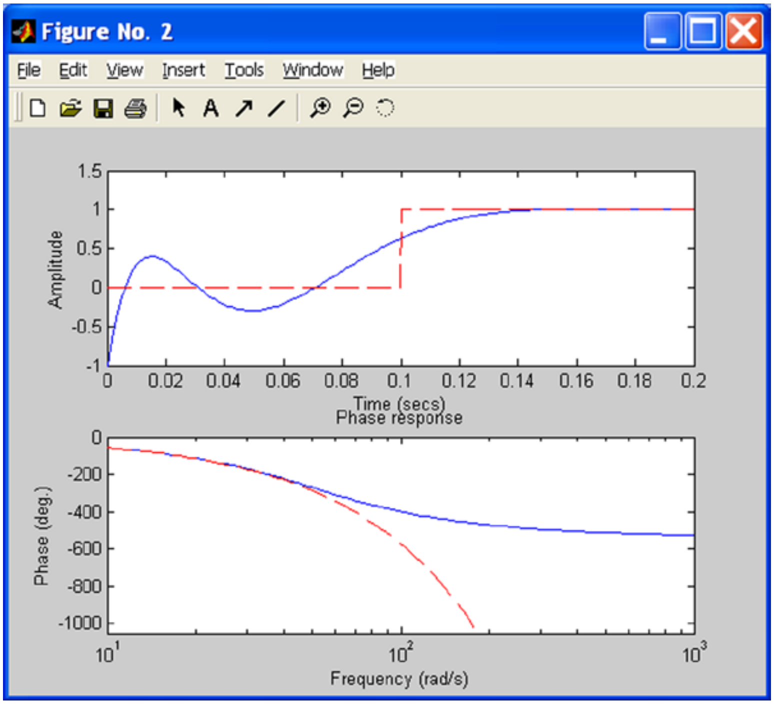


Figure 3-10.

Commands for Reduced Order Models

Command	Description
[sysb,g] = balreal(sys)	*Computes a balanced realization sysb for the stable portion of the LTI model sys. balreal handles both continuous and discrete systems.*
[sysb,g,T,Ti] = balreal(sys)	*In addition returns the vector g containing the diagonal of the balanced gramian, the state similarity transformation $x_b = Tx$ used to convert sys to sysb, and the inverse transformation $Ti = T^{-1}$*

(*continued*)

Command	Description
sysr = minreal(sys)	*Eliminates uncontrollable or unobservable states in state-space models, or cancels pole-zero pairs in transfer functions or zero-pole-gain models.*
sysr = minreal(sys,tol)	*Specifies the tolerance used for state elimination or pole-zero cancellation. The default value is tol = sqrt(eps) and increasing this tolerance forces additional cancellations.*
[sysr,u] = minreal(sys,tol)	*In addition finds an orthogonal matrix U such that (U*A*U',U*B,C*U') is a Kalman decomposition of (A,B,C).*
rsys = modred(sys,elim)	*Reduces the order of a continuous or discrete state-space model sys by eliminating the states found in the vector elim. The full state vector X is partitioned as X = [X1;X2] where X1 is the reduced state vector and X2 is discarded.*
rsys = modred(sys,elim,'method')	*In addition specifies the state elimination method, which can be MatchDC (enforce matching DC gains) or Truncate (delete X2).*
MSYS = sminreal(sys)	*Eliminates the states of the state-space model sys that don't affect the input/output response.*

In the example that follows we consider the zero-pole-gain model defined by *sys* = *zpk*([- 10 - 20.01], [- 5 - 9.9 -20.1], 1) and estimate a balanced realization, presenting the diagonal of the balanced grammian.

```
>> sys = zpk([-10 -20.01],[-5 -9.9 -20.1],1)

Zero/pole/gain:

   (s+10) (s+20.01)
----------------------
(s+5) (s+9.9) (s+20.1)

>> [sysb,g] = balreal(sys)

a =
               x1        x2        x3
   x1       -4.97    0.2399    0.2262
   x2     -0.2399    -4.276    -9.467
   x3      0.2262     9.467    -25.75

b =
                u1
   x1           -1
   x2     -0.02412
   x3      0.02276

c =
               x1        x2        x3
   y1          -1   0.02412   0.02276
```

```
d =
       u1
   y1   0

Continuous-time model.

g =
    0.1006
    0.0001
    0.0000
```

The result shows that the last two states are weakly coupled to the input and output, so it will be convenient to remove them by using the syntax:

```
>> sysr = modred(sysb,[2 3],'del')

a =
          x1
   x1  -4.97

b =
       u1
   x1  -1

c =
       x1
   y1  -1

d =
       u1
   y1   0

Continuous-time model.
```

Now we can compare the answers of the original and reduced models (Figure 3-11) by using the following syntax:

```
>> bode(sys,'-',sysr,'x')
```

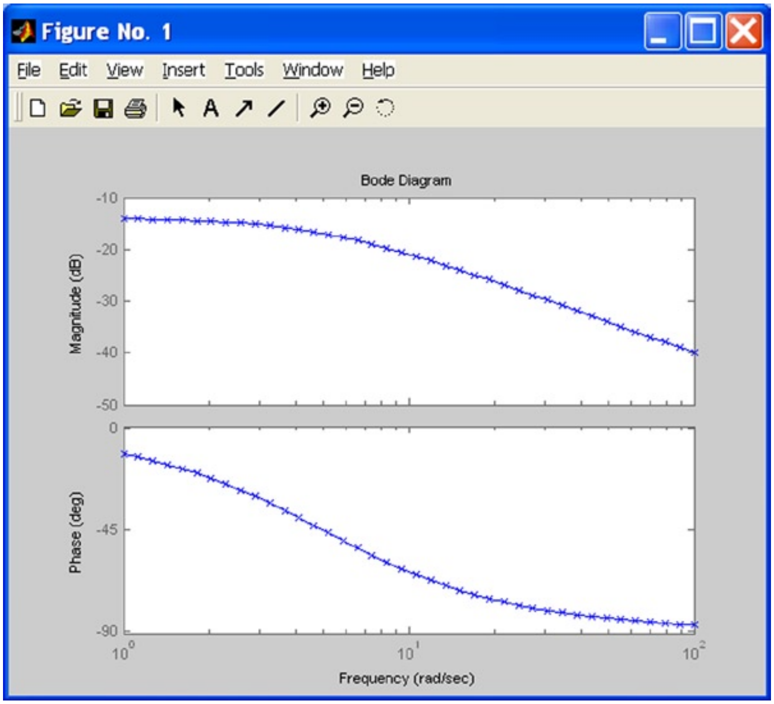


Figure 3-11.

Commands Related to State-Spaces

Command	Description
csys = canon(sys,'*type*')	*Transforms the linear model sys into a canonical state-space model csys. The argument 'type' can be either 'modal' or 'companion.'*
[csys,T] = canon(sys,'*type*')	*In addition returns the state-coordinate transformation T that relates the states of the state-space model sys to the states of csys.*
Co = ctrb(A,B) **Co = ctrb(sys)**	*Returns the controllability matrix for state-space systems.*
[Abar,Bbar,Cbar,T,k] = ctrbf(A,B,C) **[Abar,Bbar,Cbar,T,k] = ctrbf(A,B,C,tol)**	*Decomposes the state-space system represented by A, B, and C into the controllability staircase form, Abar, Bbar, and Cbar. T is the similarity transformation matrix and k is a vector of length n, where n is the order of the system represented by A. The number of non-null values of k indicates the number of iterations needed to calculate T.*
Wc = gram(sys,'c') **Wo = gram(sys,'o')**	*Calculates the controllability and observability grammians of the state-space model sys.*
Ob = obsv(A,B) **Ob = obsv(sys)**	*Calculates the observability matrix for state-space models.*

(*continued*)

Command	Description
[Abar,Bbar,Cbar,T,k] = obsvf(A,B,C) **[Abar,Bbar,Cbar,T,k] = obsvf(A,B,C,tol)**	*Decomposes the state-space system with matrices A, B, and C into the observability staircase form Abar, Bbar, and Cbar. T is the similarity transformation matrix and k is a vector of length n, where n is the order of the system represented by A. The number of non-null values of k indicates the number of iterations needed to calculate T.*
sysT = ss2ss(sys,T)	*Returns the transformed state-space model sysT given sys and the state coordinate transformation T.*
[sysb,T] = ssbal(sys) **[sysb,T] = ssbal(sys,condT)**	*Balances state-space models using a diagonal similarity transformation.*

As a first example we consider the following continuous state-space model:

$$A=\begin{bmatrix}1 & 10^4 & 10^2\\ 0 & 10^2 & 10^5\\ 10 & 1 & 0\end{bmatrix},\ B=\begin{bmatrix}1\\1\\1\end{bmatrix},\ C=\begin{bmatrix}0.1 & 10 & 100\end{bmatrix}$$

We calculate the balanced model as follows:

```
>> a = [1 1e4 1e2; 0 1e2 1e5; 10 1 0];
b = [1; 1; 1];
c = [0.1 10 1e2];
sys ss (a, b, c, 0) =

a =
           x1       x2       x3
   x1       1   1e+004      100
   x2       0      100   1e+005
   x3      10        1        0

b =
       u1
   x1   1
   x2   1
   x3   1

c =
        x1   x2    x3
   y1  0.1   10   100

d =
       u1
   y1   0

Continuous-time model.
```

In the following example we calculate the observability matrix of the ladder system $A = [1, 1; 4, -2]$, $B = [1, -1, 1, -1]$, $C = [0, 1; 1, 0]$

```
>> A = [1, 1; 4, - 2]; B = [1, - 1, 1, - 1]; C = [1,0; 0.1];
>> [Abar, Bbar, Cbar, T, k] = obsvf(A,B,C)

Abar =

     1     1
     4    -2

Bbar =

     1    -1
     1    -1

Cbar =

     1     0
     0     1

T =

     1     0
     0     1

k =

     2     0
```

Below we calculate the controllability matrix of the system in the previous example.

```
>> A = [1, 1; 4, - 2]; B = [1, - 1, 1, - 1]; C = [1,0; 0.1];
>> [Abar, Bbar, Cbar, T, k] = ctrbf(A,B,C)

Abar =

   -3.0000    0.0000
    3.0000    2.0000

Bbar =

         0         0
   -1.4142    1.4142

Cbar =

   -0.7071   -0.7071
    0.7071   -0.7071
```

```
T =

   -0.7071    0.7071
   -0.7071   -0.7071

k =

      1     0
```

Commands for Dynamic Models

Command	Description
[Wn,Z] = damp(sys) **[Wn,Z,P] = damp(sys)**	*Displays a table of the damping ratio, natural frequency, and time constant of the poles of the linear model sys. You can also get the vector P of the poles of sys.*
k = dcgain(sys)	*Calculates the low-frequency (DC) gain of the model sys.*
[P,Q] = covar(sys,W)	*Calculates the stationary covariance of the output of an LTI model sys driven by Gaussian white noise inputs W. P is the steady-state output response covariance and Q is the steady-state state covariance.*
s = dsort(p) **[s,ndx] = dsort(p)**	*Sorts the discrete-time poles contained in the vector p in descending order by magnitude.*
s = esort(p) **[s,ndx] = esort(p)**	*Sorts the continuous-time poles contained in the vector p by real part.*
norm(sys)	*Calculates the H^2 norm of the model sys.*
norm(sys,2)	*Calculates the H^2 norm of the model sys.*
norm(sys,inf)	*Calculates the H_∞ norm of the model sys.*
norm(sys,inf,tol)	*Calculates the H_∞ norm of the model sys with tolerance tol.*
[ninf,fpeak] = norm(sys)	*Calculates, in addition to the H_∞ norm, the frequency fpeak at which the gain reaches its peak value.*
p = pole(sys)	*Calculates the poles of the LTI model sys.*
d = eig(A)	*Returns the vector of eigenvalues of A.*
d = eig(A,B)	*Returns the generalized eigenvalues of the pair(A,B).*
[V,D] = eig(A)	*Returns the eigenvalues and eigenvectors of the matrix A.*
[V,D] = eig(A,'nobalance')	*Returns the eigenvalues and eigenvectors of A without a preliminary balancing step.*
[V,D] = eig(A,B)	*Returns the eigenvalues and generalized eigenvectors of (A,B).*
[V,D] = eig(A,B,flag)	*Returns the eigenvalues and generalized eigenvectors of (A,B). The factorization method ('chol' or 'qz') is specified by flag.*
pzmap(sys) **pzmap(sys1,sys2,...,sysN)** **[p,z] = pzmap(sys)**	*Creates a pole-zero plot of the continuous-time or discrete-time dynamic system sys or of several LTI systems sys1, sys2,..., sysn at the same time. [p, z] gives the poles and zeros and not the graph.*

(*continued*)

Command	Description
rlocus(sys)	*Calculates and plots the root locus of the open-loop SISO model sys.*
rlocus(sys,k)	*Uses the user-specified vector k of gains to plot the root locus.*
rlocus(sys1,sys2,...)	*Calculates and plots the root locus of several systems in a simple graph.*
[r,k] = rlocus(sys)	*Returns the vector k of selected gains and the complex root locations r for these gains.*
r = rlocus(sys,k)	*Returns the root locations r for a system sys with selected gains given by the vector k.*
r = roots(c)	*Returns the roots of the polynomial c as a column vector.*
sgrid	*Generates, for pole-zero and root locus plots, a grid of constant damping factors from zero to one in steps of 0.1 and natural frequencies from zero to 10 rad/sec in steps of one rad/sec, and plots the grid over the current axis.*
zgrid	*Similarly generates a grid from zero to π in steps of π/10, and plots the grid over the current axis.*
z = zero(sys)	*Calculates the zeros of the LTI model sys.*
[z,gain] = zero(sys)	*Returns the zeros and gain of the LTI system sys.*

As a first example, we calculate the eigenvalues, natural frequencies and damping factors of the continuous transfer function model:

$$H(s) = \frac{2s^2 + 5s + 1}{s^2 + 2s + 3}$$

```
>> H = tf([2 5 1],[1 2 3])

Transfer function:

2 s^2 + 5 s + 1
---------------
s^2 + 2 s + 3

>> damp(H)

Eigenvalue              Damping     Freq. (rad/s)

00e - 1 + 000 + 1. 41e + 000i 5. 77e-001 1. 73e + 000
00e - 1 + 000 - 1. 41e + 000i 5. 77e-001 1. 73e + 000
```

In the following example we calculate the DC gain of the MIMO transfer function model:

$$H(s) = \begin{bmatrix} 1 & \dfrac{s-1}{s^2+s+3} \\ \dfrac{1}{s+1} & \dfrac{s+2}{s-3} \end{bmatrix}$$

```
>> H = [1 tf([1 -1],[1 1 3]) ; tf(1,[1 1]) tf([1 2],[1 -3])]
dcgain(H)

Transfer function from input 1 to output...

#1:  1

        1
#2:  -----
     s + 1

Transfer function from input 2 to output...

         s
#1:  -----------
      s^2 + s + 3

     s + 2
#2:  -----
      3s

ans =

1.0000 - 0.3333
1.0000 - 0.6667
```

Next we consider the discrete-time transfer function

$$H(z) = \frac{z^3 - 2.841z^2 + 2.875z - 1.004}{z^3 - 2.417z^2 + 2.003z - 0.5488}$$

with 0.1 second sampling time and calculate the 2-norm and the infinite norm with its optimum value.

```
>> H = tf([1 -2.841 2.875 -1.004],[1 -2.417 2.003 -0.5488],0.1)
norm(H)

Transfer function:

z^3 - 2.841 z^2 + 2.875 z - 1.004
----------------------------------
z^3 - 2.417 z^2 + 2.003 z - 0.5488

Sampling time: 0.1

ans =

1.2438
```

```
>> [ninf,fpeak] = norm(H,inf)

surrounded =

2.5488

fpeak =

3.0844
```

We then confirm the previous values by generating the Bode plot of $H(z)$ (see Figure 3-12).

```
>> bode (H)
```

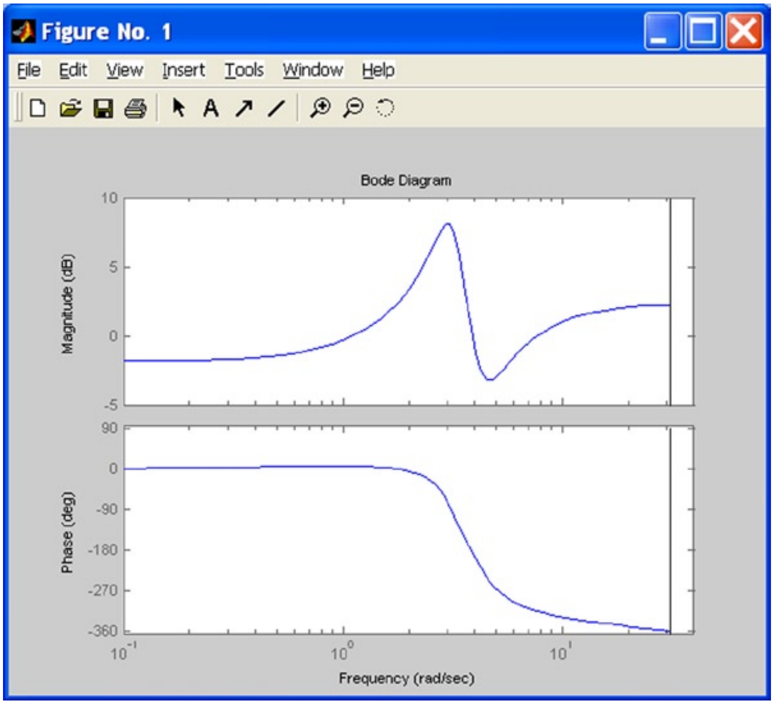


Figure 3-12.

Next we calculate and graph the root locus of the following system (see Figure 3-13):

$$h(s) = \frac{2s^2 + 5s + 1}{s^2 + 2s + 3}$$

```
>> h = tf([2 5 1],[1 2 3]);
rlocus (h)
```

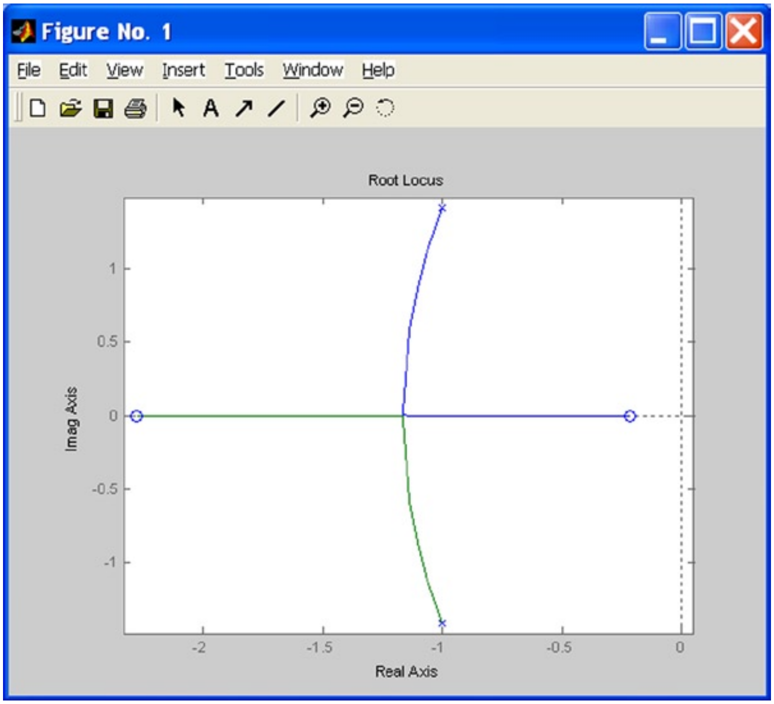


Figure 3-13.

In the example below we plot a *z*-plane grid over the root locus of the following system (see Figure 3-14):

$$H(z) = \frac{2z^2 - 3.4z + 1.5}{z^2 - 1.6z + 0.8}$$

```
> H = tf([2 -3.4 1.5],[1 -1.6 0.8],-1)

Transfer function:

2 z^2 - 3.4 z + 1.5
-------------------
z^2 - 1.6 z + 0.8

Sampling time: unspecified

>> rlocus(H)
zgrid
axis('square')
```

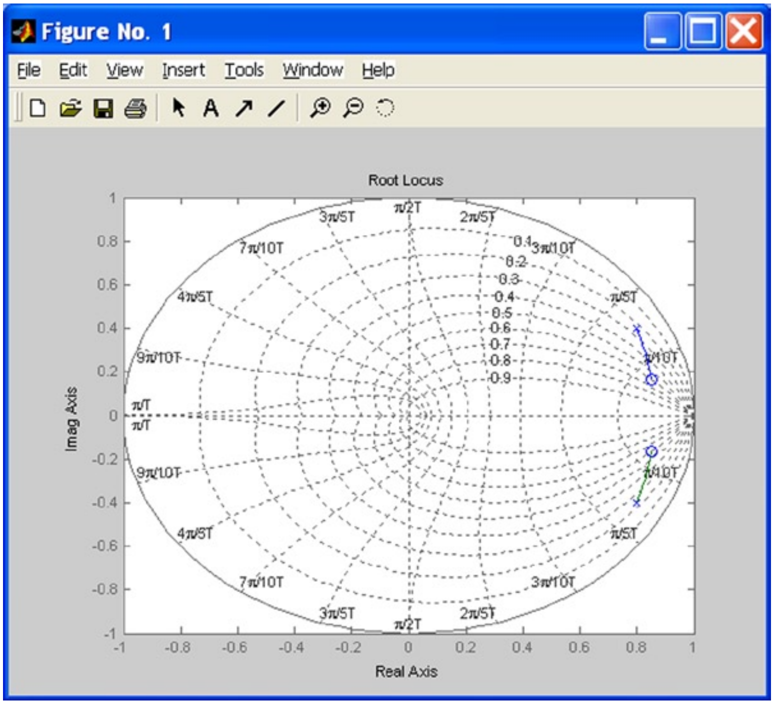


Figure 3-14.

Commands for Interconnecting Models

Command	Description
sys = append(sys1,sys2,...,sysN)	*Combines models in a diagonal configuration block. Groups the models together by appending their inputs and outputs (Figure 3-15).*
asys = augstate (sys)	*Appends the state vector to the output vector.*
sysc = connect(sys,Q,inputs,outputs)	*Connects the subsystems in a block according to a chosen interconnection scheme (given by the connection matrix Q).*
sys = feedback(sys1,sys2) **sys = feedback(sys1,sys2,sign)** **sys = feedback(sys1,sys2,feedin,feedout,sign)**	*Returns a model sys for the negative feedback interconnection of models sys1 and sys2 (see Figure 3-16). May include sign and closed loop (see Figure 3-17).*
sys = lft(sys1,sys2) **sys = lft(sys1,sys2,nu,ny)**	*Forms the linear fractional transformation (LFT) of two models (see Figure 3-18).*
[A,B,C,D] = ord2(wn,z) **[num,den] = ord2(wn,z)**	*Generates continuous second-order systems (wn is the natural frequency and z is the damping factor).*

(*continued*)

Command	Description
sys = parallel(sys1,sys2) **sys = parallel(sys1,sys2,inp1,inp2,out1,out2)**	*Connects two systems in parallel (see Figure 3-19).*
sys = series(sys1,sys2) **sys = series(sys1,sys2,outputs1,inputs2)**	*Connects two systems in series (see Figure 3-20).*
sys = stack(arraydim,sys1,sys2,...)	*Produces an array of dynamic system models by stacking the models sys1,sys2,... along the array dimension arraydim.*

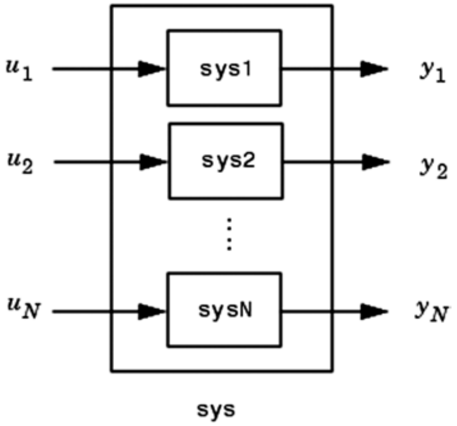


Figure 3-15.

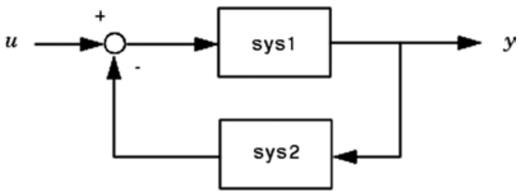


Figure 3-16.

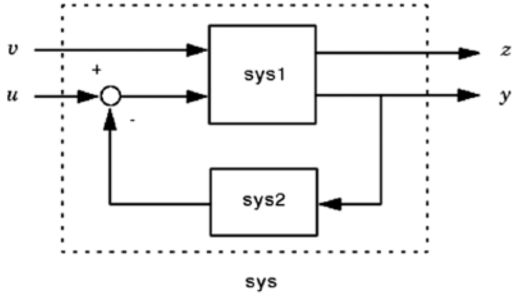


Figure 3-17.

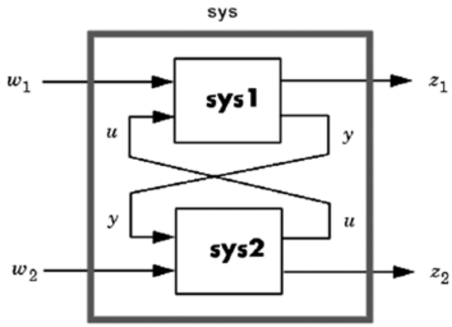


Figure 3-18.

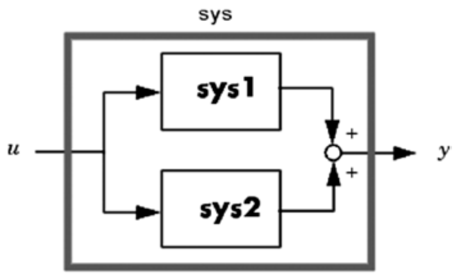


Figure 3-19.

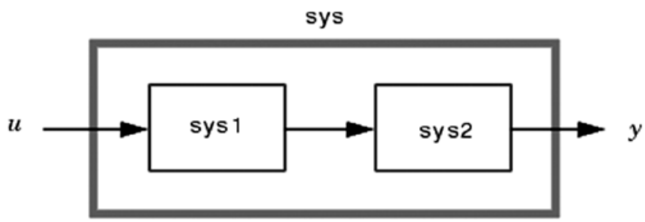


Figure 3-20.

As a first example we will combine the systems *tf*(1, [1 0]) and *ss*(1,2,3,4). We should bear in mind that for systems with transfer functions $H_1(s)$, $H_2(s)$, ..., $H_n(s)$, the resulting combined system has as transfer function:

$$\begin{bmatrix} H_1(s) & 0 & \dots & 0 \\ 0 & H_2(s) & \dots & \dots \\ \dots & \dots & \dots & 0 \\ 0 & \dots & 0 & H_n(s) \end{bmatrix}$$

For two systems *sys*1 and *sys*2 defined by (A_1, B_1, C_1, D_1) and (A_2, B_2, C_2, D_2), their combination *append*(*sys1, sys2*) yields the system:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} A_1 & 0 \\ 0 & A_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} B_1 & 0 \\ 0 & B_2 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} C_1 & 0 \\ 0 & C_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} D_1 & 0 \\ 0 & D_2 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

For our example we have:

```
>> sys1 = tf(1,[1 0])
sys2 = ss(1,2,3,4)
sys = append(sys1,10,sys2)

Transfer function:

1
-
s

a =
        x1
   x1    1

b =
        u1
   x1    2

c =
        x1
   y1    3

d =
        u1
   y1    4

Continuous-time model.

a =
        x1   x2
   x1    0    0
   x2    0    1

b =
        u1   u2   u3
   x1    1    0    0
   x2    0    0    2
```

```
c =
        x1   x2
   y1    1    0
   y2    0    0
   y3    0    3

d =
        u1   u2   u3
   y1    0    0    0
   y2    0   10    0
   y3    0    0    4

Continuous-time model.
```

The following example, illustrated in Figure 3-21, attaches the plant $G(s)$ to the driver $H(s)$, defined below, using negative feedback:

$$G(s) = \frac{2s^2 + 5s + 1}{s^2 + 2s + 3}$$

$$H(s) = \frac{5(s+1)}{s+10}$$

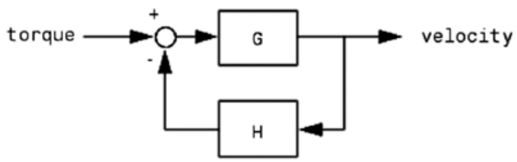


Figure 3-21.

```
>> G = tf([2 5 1],[1 2 3],'inputname','torque',...)
'outputname','velocity');
H = zpk(-2,-10,5)
Cloop = feedback(G,H)

Zero/pole/gain:

5 (s+2)
-------
(s+10)

Zero/pole/gain from input "torque" to output "velocity":

0.18182 (s+10) (s+2. 281) (s+0. 2192)
------------------------------------
(s+3. 419) (s ^ 2 + 1. 763s + 1.064)
```

The following example builds a second-order transfer function with damping factor 0.4 and natural frequency 2.4 rad/sec.

```
>> [num,den] = ord2(2.4,0.4)

num =

1

den =

1.0000    1.9200    5.7600

>> sys = tf(num,den)

Transfer function:

         1
-------------------
s ^ 2 + 1.92 s + 5.76
```

Response Time Commands

Command	Description
[u, t] = gensig(*type*,tau)	*Generates a scalar signal u of class type and with period tau (in seconds). The type can be sine, square or pulse.*
[u, t] = gensig(*type*,tau,Tf,Ts)	*Also specifies the time duration Tf of the signal and the spacing Ts between the time samples t.*
impulse(sys)	*Calculates and plots the impulse response of the model sys.*
impulse(sys,t)	*Uses the user-supplied time vector t for simulation.*
impulse(sys1,sys2,...,sysN)	*Calculates and plots the impulse response of several models.*
impulse(sys1,sys2,...,sysN,t)	*Calculates and plots the impulse response of several models using the user-supplied time vector t for simulation.*
impulse(sys1,'PlotStyle1',...,sysN,'PlotStyleN')	*In addition sets graphics styles.*
[y, t, x] = impulse(sys)	*Returns the length of t, the number of outputs and the number of inputs for the impulse response of the model sys.*
initial(sys,x0) **initial(sys,x0,t)** **initial(sys1,sys2,...,sysN,x0)** **initial(sys1,sys2,...,sysN,x0,t)** **initial(sys1,'PlotStyle1',...,sysN,'PlotStyleN',x0)** **[y, t, x] = initial(sys,x0)**	*Calculates and plots the unforced response of the state-space model sys, or of several models, with initial condition x0. A user-supplied time vector t can be supplied as well as specified graphics styles. You can also obtain the length of t, the number of outputs and the number of inputs for the unforced response of the model sys.*

(*continued*)

Command	Description
lsim(sys,u,t) **lsim(sys,u,t,x0)** **lsim(sys,u,t,x0,'zoh')** **lsim(sys,u,t,x0,'foh')** **lsim(sys1,sys2,...,sysN,u,t)** **lsim(sys1,sys2,...,sysN,u,t,x0)** **lsim(sys1,'PlotStyle1',...,sysN,'PlotStyleN',u,t)** **[y, t, x] = lsim(sys,u,t,x0)**	*Calculates and plots the time response of the state-space model sys, or of several models, with initial condition x0. A user-supplied time sample t can be supplied as well as specified graphics styles. The options zoh and foh specify how the input values should be interpolated between samples (zero-order hold or linear interpolation, respectively). You can also obtain the output response y, the time vector t used for simulation, and the state trajectories x.*
step(sys) **step(sys,t)** **step(sys1,sys2,...,sysN)** **step(sys1,sys2,...,sysN,t)** **step(sys1,'PlotStyle1',...,sysN,'PlotStyleN')** **[y, t, x] = step(sys)**	*Calculates and plots the step response of the LTI model sys, or several models. A user-supplied time sample t can be supplied as well as specified graphics styles. You can also obtain the output response y, the time vector t used for simulation, and the state trajectories x.*
ltiview **ltiview(sys1,sys2,...,sysn)** **ltiview('plottype',sys1,sys2,...,sysn)** **ltiview('plottype',sys,extras)** **ltiview('clear',viewers)** **ltiview('current'sys1,sys2,...,** **sysn,viewers)**	*Opens an LTI Viewer for LTI system response analysis for one or more systems and with different graphics options defined by plottype ('step', 'impulse', 'initial', 'lsim', 'pzmap' 'bode', 'nyquist', 'nichols' and 'sigma').*

As a first example we generate and plot a square signal with period 5 seconds, duration 30 seconds and sampling every 0.1 seconds (see Figure 3-22).

```
>> [u,t] = gensig('square',5,30,0.1);
>> plot(t,u)
axis([0 30-1 2])
```

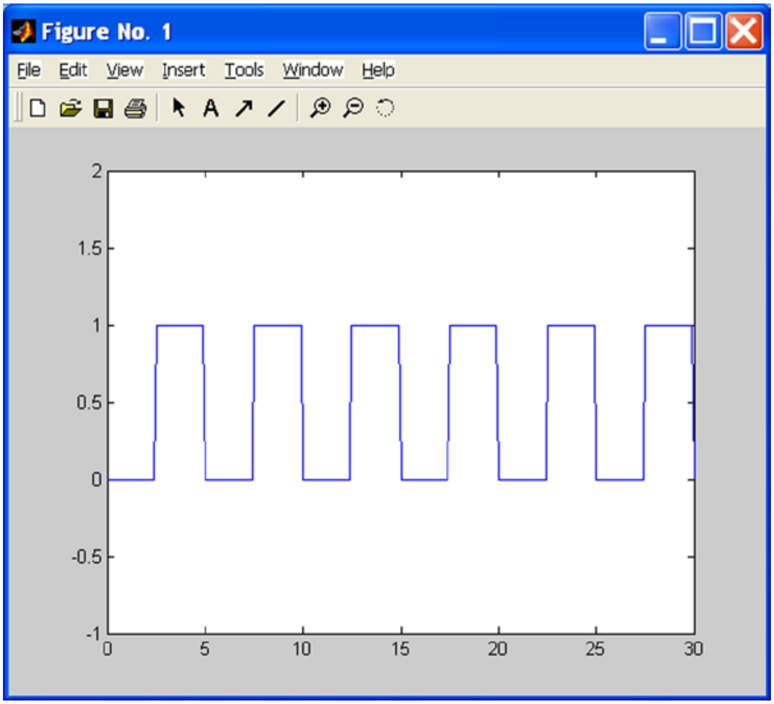


Figure 3-22.

In the example below we generate the response plot for the following state-space model (see Figure 3-23):

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} -0.5572 & -0.7814 \\ 0.7814 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$y = [1.9691 \quad 6.4493] \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

with initial conditions

$$x(0) = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

```
>> a = [-0.5572   -0.7814;0.7814  0];
c = [1.9691  6.4493];
x0 = [1 ; 0]
sys = ss(a,[],c,[]);
initial (sys, x 0)

x 0 =

1
0
```

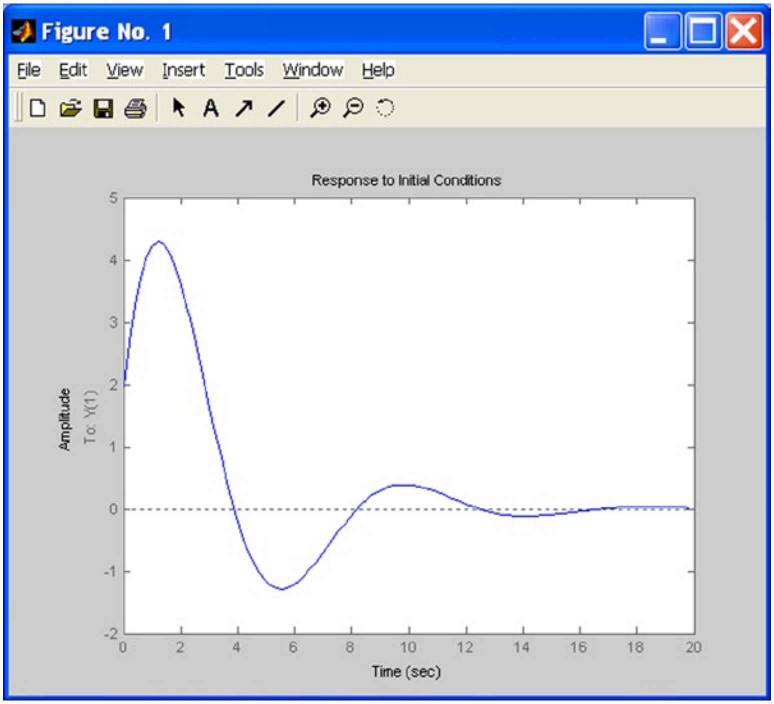


Figure 3-23.

Below we generate the step response plot of the following second order state-space model (see Figure 3-24):

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} -0.5572 & -0.7814 \\ 0.7814 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 1 & -1 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

$$y = [1.9691 \quad 6.4493] \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

The following syntax is used:

```
>> a = [-0.5572   -0.7814;0.7814  0];
b = [1 -1;0 2];
c = [1.9691  6.4493];
sys = ss(a,b,c,0);
step(sys)
```

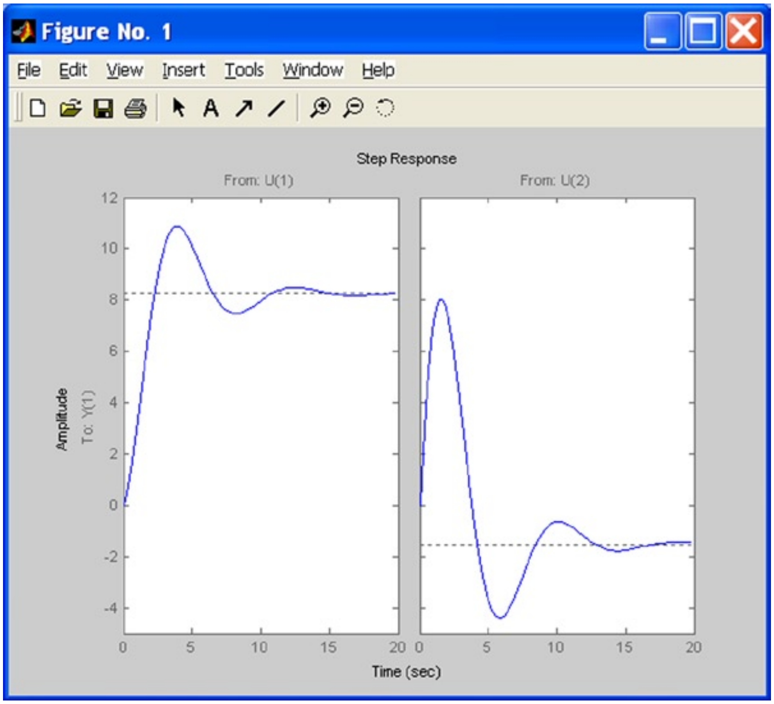


Figure 3-24.

Frequency Response Commands

Command	Description
S = allmargin(sys)	*Computes the gain margin, phase margin, delay margin and the corresponding crossover frequencies of the SISO open-loop model sys.*
bode(sys) **bode(sys,w)** **bode(sys1,sys2,...,sysN)** **bode(sys1,sys2,...,sysN,w)** **bode(sys1,'PlotStyle1',...,** **sysN,'PlotStyleN')** **[mag,phase,w] = bode(sys)**	*Creates a Bode plot of the frequency response of the model sys, or of several systems. The frequency range can be specified by w as well as various graphics options. You can also obtain the magnitude, phase and frequency values of bode(sys).*

(*continued*)

Command	Description
bodemag(sys) **bodemag(sys,{wmin,wmax})** **bodemag(sys,w)** **bodemag(sys1,sys2,...,sysN,w)** **bodemag(sys1,'PlotStyle1',...,** **sysN,'PlotStyleN')**	*Creates a Bode plot of the frequency response of the model sys, or of several models, without the phase diagram. The frequency range and various graphics options can be user-specified.*
frsp = evalfr(sys,f)	*Evaluates the transfer function of the system sys at the complex frequency f.*
H = freqresp(sys,w)	*Returns the frequency response of sys on the real frequency grid specified by the vector w.*
isys = interp(sys,freqs)	*Interpolates the frequency response data contained in the FRD model sys at the frequencies freqs.*
y = linspace(a,b) **y = linspace(a,b,n)**	*Creates a vector with 100 or n values equally spaced between a and b.*
y= logspace(a,b) **y = logspace(a,b,n)** **y = logspace(a,pi,n)**	*Creates a vector with uniform logarithmic spacing between* 10^a *and* 10^b *(50 points between* 10^a *and* 10^b*, n points between* 10^a *and* 10^b *or n points between* 10^a *and* π*).*
[Gm,Pm,Wgm,Wpm] = margin(sys) **[Gm,Pm,Wgm,Wpm] = margin(mag,phase,w)** **margin(sys)**	*Calculates the minimum gain margin, Gm, phase margin, Pm, and associated frequencies Wgm and Wpm of SISO open-loop models. Magnitude, phase and frequency vectors can be specified, and the Bode plot can be generated.*
ngrid	*Superimposes Nichols chart grid lines over the Nichols frequency response of a system.*
nichols(sys) **nichols(sys,w)** **nichols(sys1,sys2,...,sysN)** **nichols(sys1,sys2,...,sysN,w)** **nichols(sys1,'PlotStyle1',...,** **sysN,'PlotStyleN')** **[mag,phase,w] = nichols(sys)** **[mag,phase] = nichols(sys,w)**	*Creates a Nichols chart of the frequency response of a model. The arguments have the same meanings as for the Bode plot.*

(*continued*)

Command	Description
nyquist(sys) **nyquist(sys,w)** **nyquist(sys1,sys2,...,sysN)** **nyquist(sys1,sys2,...,sysN,w)** **nyquist(sys1,'PlotStyle1',...,** **sysN,'PlotStyleN')** **[re,im,w] = nyquist(sys)** **[re,im] = nyquist(sys,w)**	*Creates a Nyquist plot of the frequency response of a model. The arguments have the same meanings as for the Bode plot.*
sigma(sys) **sigma(sys,w)** **sigma(sys,w,type)** **sigma(sys1,sys2,...,sysN)** **sigma(sys1,sys2,...,sysN,w)** **sigma(sys1,sys2,...,sysN,w,type)** **sigma(sys1,'PlotStyle1',...,** **sysN,'PlotStyleN')** **[sv,w] = sigma(sys)** **sv = sigma(sys,w)**	*Calculates the singular values of the frequency response of a model.*

As a first example we generate the Bode plot for the following continuous SISO system (see Figure 3-25):

$$H(s) = \frac{s^2 + 0.1s + 7.5}{s^4 + 0.12s^3 + 9s^2}$$

```
>> g = tf([1 0.1 7.5],[1 0.12 9 0 0]);
bode (g)
```

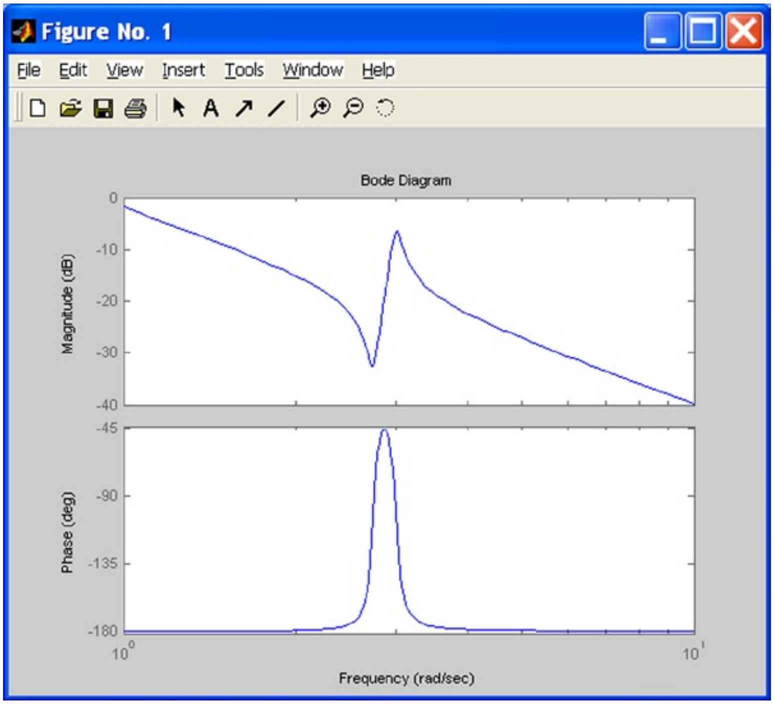


Figure 3-25.

Next we evaluate the following discrete-time transfer function at $z = 1 + i$:

$$H(z) = \frac{z-1}{z^2+z+1}$$

```
>> H = tf([1 -1],[1 1 1],-1)
z = 1+j
evalfr(H,z)

Transfer function:

   z - 1
-----------
z^2 + z + 1

Sampling time: unspecified

z =

1.0000 + 1. 0000i

ans =

0.2308 + 0. 1538i
```

Next we generate the Nichols chart, with grid, for the following system (see Figure 3-26):

$$H(s)=\frac{-4s^4+48s^3-18s^2+250s+600}{s^4+30s^3+282s^2+525s+60}$$

```
>> H = tf([-4 48 -18 250 600],[1 30 282 525 60])

Transfer function:

-4 s^4 + 48 s^3 - 18 s^2 + 250s + 600
--------------------------------------
s^4 + 30 s^3 + 282 s^2 + 525s + 60

>> nichols(H)
>> ngrid
```

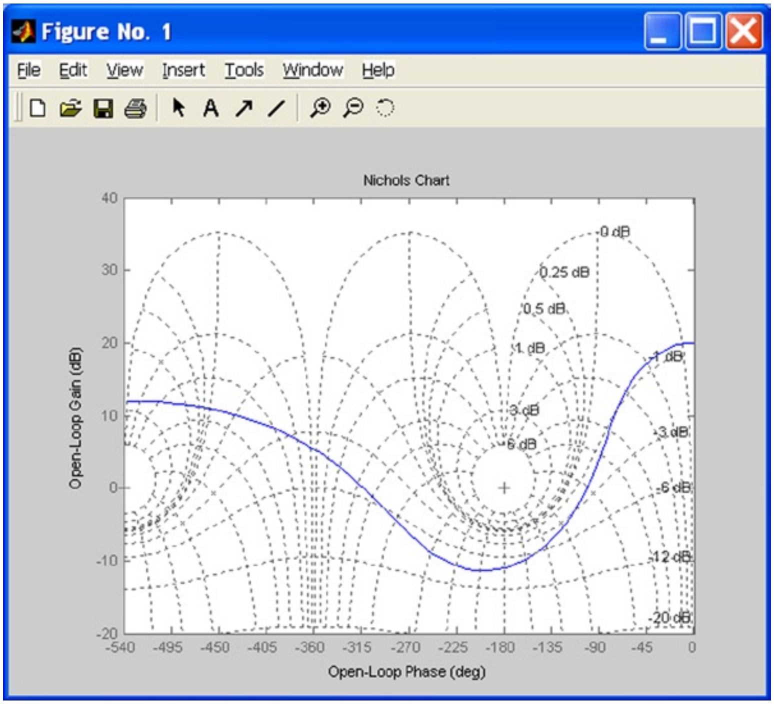


Figure 3-26.

Pole Location Commands

Command	Description
k = acker(A,b,p)	*Given the single input system* $$\frac{dx}{dt} = Ax + bu$$ *and a vector p of desired closed-loop pole locations, using Ackermann's method, k is determined such that the eigenvalues of A – bk match the entries of p (up to ordering).*
K = place(A,B,p)	*Given the single or multi-input system* $$\frac{dx}{dt} = Ax + Bu$$ *and a vector p of desired closed-loop pole locations, k is determined such that the eigenvalues of A – bk match the entries of p (up to ordering).*
est = estim(sys,L) **est = estim(sys,L,sensors,known)**	*Produces a state/output estimator est given the plant state-space model sys and the estimator gain L. The measured outputs (sensors) and the known inputs (known) can be specified.*
rsys = reg(sys,K,L) **rsys = reg(sys,K,L,sensors,known,controls)**	*Forms a dynamic regulator or compensator rsys given a state-space model sys of the plant, a state-feedback gain matrix K, and an estimator gain matrix L. The measured outputs (sensors) and the known inputs (known) can be specified.*

LQG Design Commands

Command	Description
[K, S, e] = lqr(A,B,Q,R) **[K, S, e] = lqr(A,B,Q,R,N)**	*Calculates the LQ-optimal gain for continuous models.*
[K, S, e] = dlqr(a,b,Q,R) **[K, S, e] = dlqr(a,b,Q,R,N)**	*Calculates the LQ-optimal gain for discrete models.*
[K,S,e] = lqry(sys,Q,R) **[K,S,e] = lqry(sys,Q,R,N)**	*Calculates the LQ-optimum gain with weighted output.*
[Kd,S,e] = lqrd(A,B,Q,R,Ts) **[Kd,S,e] = lqrd(A,B,Q,R,N,Ts)**	*Calculates the discrete LQ gain for continuous models.*
[kest,L,P] = kalman(sys,Qn,Rn,Nn) **[kest,L,P,M,Z] = kalman(sys,Qn,Rn,Nn)**	*Computes the Kalman estimator for continuous and discrete models.*
[kest,L,P,M,Z] = kalmd(sys,Qn,Rn,Ts)	*Computes the discrete Kalman estimator for continuous models.*
rlqg = lqgreg(kest,k) **rlqg = lqgreg(kest,k,controls)**	*Forms the linear-quadratic-Gaussian (LQG) regulator by connecting the Kalman estimator designed with kalman and the optimal state-feedback gain designed with lqr, dlqr or lqry.*

Commands for Solving Equations

Command	Description
[X,L,G,rr] = care(A,B,Q) **[X,L,G,rr] = care(A,B,Q,R,S,E)** **[X,L,G,report] = care(A,B,Q,...,'report')** **[X1,X2,L,report] = care(A,B,Q,...,'implicit')**	*Solves algebraic Riccati equations in continuous time.*
[X,L,G,rr] = dare(A,B,Q,R) **[X,L,G,rr] = dare(A,B,Q,R,S,E)** **[X,L,G,report] = dare(A,B,Q,...,'report')** **[X1,X2,L,report] = dare(A,B,Q,...,'implicit')**	*Solves algebraic Riccati equations in discrete time.*
X = lyap(A,Q) **X = lyap(A,B,C)**	*Solves continuous-time Lyapunov equations.*
X = dlyap(A,Q)	*Solves discrete-time Lyapunov equations.*

As an example, we solve the Riccati equation:

$$A^T X + XA - XBR^{-1}B^T X + C^T C = 0$$

where:

$$A = \begin{bmatrix} -3 & 2 \\ 1 & 1 \end{bmatrix} \quad B = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad C = \begin{bmatrix} 1 & -1 \end{bmatrix} \quad R = 3$$

```
>> a = [-3 2;1 1]; b = [0 ; 1]; c = [1 -1]; r = 3;
[x,l,g] = care(a,b,c'*c,r)

x =

0.5895 1.8216
1.8216 8.8188

l =

-3.5026
-1.4370

g =

0.6072 2.9396
```

EXERCISE 3-1

Create the continuous state-space model and compute the realization of the state-space for the transfer function $H(s)$ defined below. Also find a minimal realization of $H(s)$.

$$H(s) = \begin{bmatrix} \dfrac{s+1}{s^3+3s^2+3s+2} \\ \dfrac{s^2+3}{s^2+s+1} \end{bmatrix}$$

```
>> H = [tf([1 1],[1 3 3 2]) ; tf([1 0 3],[1 1 1])];
>> sys = ss(H)

a =
            x1      x2      x3      x4      x5
x1          -3    -1.5      -1       0       0
x2           2       0       0       0       0
x3           0       1       0       0       0
x4           0       0       0      -1    -0.5
x5           0       0       0       2       0

b =
            U1
x1           1
x2           0
x3           0
x4           1
x5           0

c =
            x1      x2      x3      x4      x5
y1           0     0.5     0.5       0       0
y2           0       0       0      -1       1

d =
            U1
y1           0
y2           1

Continuous-time model.

>> size(sys)

State-space model with 2 outputs, 1 input, and 5 states.
```

We have obtained a state-space model with 2 outputs, 1 input and 5 states. A minimal realization of *H*(*s*) is found by using the syntax:

```
>> sys = ss(H,'min')

a =
                    x1           x2           x3
        x1     -1.4183      -1.5188      0.21961
        x2    -0.14192      -1.7933     -0.70974
        x3    -0.44853       1.7658      0.21165

b =
                    u1
        x1     0.19787
        x2      1.4001
        x3     0.02171

c =
                    x1           x2           x3
        y1    -0.15944     0.018224      0.27783
        y2     0.35997     -0.77729      0.78688

d =
                    u1
        y1           0
        y2           1

Continuous-time model.

>> size(sys)

State-space model with 2 outputs, 1 input, and 3 states.
```

A minimal realization is given by a state-space model with 2 outputs, 1 input and 3 states.

This result is in accordance with the following factorization of *H*(*s*) as the composite of a first order system with a second order system:

$$H(s)=\begin{bmatrix}\dfrac{1}{s+2} & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}\dfrac{s+1}{s^2+s+1}\\ \dfrac{s^2+3}{s^2+s+1}\end{bmatrix}$$

EXERCISE 3-2

Find the discrete transfer function of the MIMO system $H(z)$ defined below where the sample time is 0.2 seconds.

$$H(z)=\begin{bmatrix}\dfrac{1}{z+0.3} & \dfrac{z}{z+0.3}\\ \dfrac{-z+2}{z+0.3} & \dfrac{3}{z+0.3}\end{bmatrix}$$

```
>> nums = {1 [1 0];[-1 2] 3}
Ts = 0.2
H = tf(nums,[1 0.3],Ts)

nums =

    [       1.00]    [1x2 double]
    [1x2 double]    [       3.00]

Ts =

          0.20

Transfer function from input 1 to output...
         1
 #1:  -------
      z + 0.3

      -z + 2
 #2:  -------
      z + 0.3

Transfer function from input 2 to output...
         z
 #1:  -------
      z + 0.3

         3
 #2:  -------
      z + 0.3

Sampling time: 0.2
```

EXERCISE 3-3

Given the zero-pole-gain model

$$H(z) = \frac{z-0.7}{z-0.5}$$

with sample time 0.01 seconds, perform a resampling to 0.05 seconds. Then undo the resampling and verify that you obtain the original model.

```
>> H = zpk(0.7,0.5,1,0.1)
H2 = d2d(H,0.05)

Zero/pole/gain:

(z-0.7)
-------
(z-0.5)

Sampling time: 0.1

Zero/pole/gain:

(z-0.8243)
----------
(z-0.7071)

Sampling time: 0.05
```

We reverse the resampling in the following way:

```
>> d2d(H2,0.1)

Zero/pole/gain:

(z-0.7)
-------
(z-0.5)

Sampling time: 0.1
```

Thus the original model is obtained.

EXERCISE 3-4

Consider the continuous fourth-order model given by the transfer function *h*(*s*) defined below. Reduce the order by eliminating the states corresponding to small values of the diagonal balanced grammian vector g. Compare the original and reduced models.

$$h(s)=\frac{s^3+11s^2+36s+26}{s^4+14.6s^3+74.96s^2+153.7s+99.65}$$

We start by defining the model and computing a balanced state-space realization as follows:

```
>> h = tf([1 11 36 26],[1 14.6 74.96 153.7 99.65])
[hb,g] = balreal(h)
g'

Transfer function:

          s^3 + 11 s^2 + 36s + 26
--------------------------------------------
s^4 + 14.6 s^3 + 74.96 s^2 + 153.7s + 99.65

a =
              x1         x2         x3         x4
   x1     -3.601    -0.8212    -0.6163    0.05831
   x2     0.8212     -0.593     -1.027    0.09033
   x3    -0.6163      1.027     -5.914      1.127
   x4   -0.05831    0.09033     -1.127     -4.492

b =
               u1
   x1      -1.002
   x2      0.1064
   x3    -0.08612
   x4   -0.008112



c =
              x1         x2         x3         x4
   y1     -1.002    -0.1064   -0.08612   0.008112

d =
        u1
   y1   0

Continuous-time model.
```

```
g =

    0.1394
    0.0095
    0.0006
    0.0000
ans =

    0.1394    0.0095    0.0006    0.0000
```

We now remove the three states corresponding to the last three values of *g* using two different methods.

```
>> hmdc = modred(hb,2:4,'mdc')
hdel = modred(hb,2:4,'del')

a =
           x1
   x1  -4.655

b =
           u1
   x1  -1.139

c =
           x1
   y1  -1.139

d =
             u1
   y1  -0.01786

Continuous-time model.

a =
           x1
   x1  -3.601

b =
           u1
   x1  -1.002

c =
           x1
   y1  -1.002

d =
       u1
   y1   0

Continuous-time model.
```

Next we compare the responses with the original model (see Figure 3-27).

```
>> bode(h,'-',hmdc,'x',hdel,'*')
```

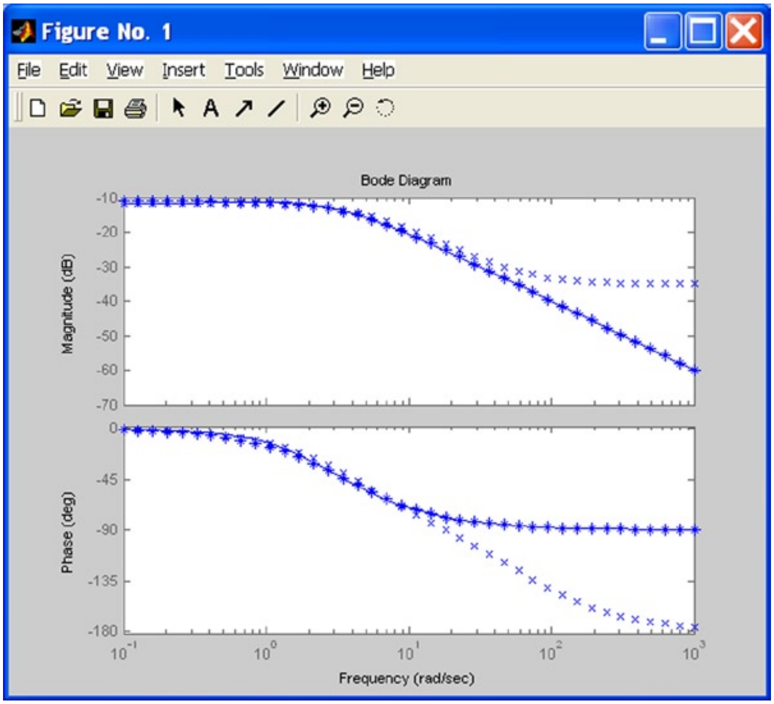


Figure 3-27.

We see that in both cases the reduced model is better than the original. We now compare the step responses (see Figure 3-28)

```
>> step(h,'-',hmdc,'-.',hdel,'--')
```

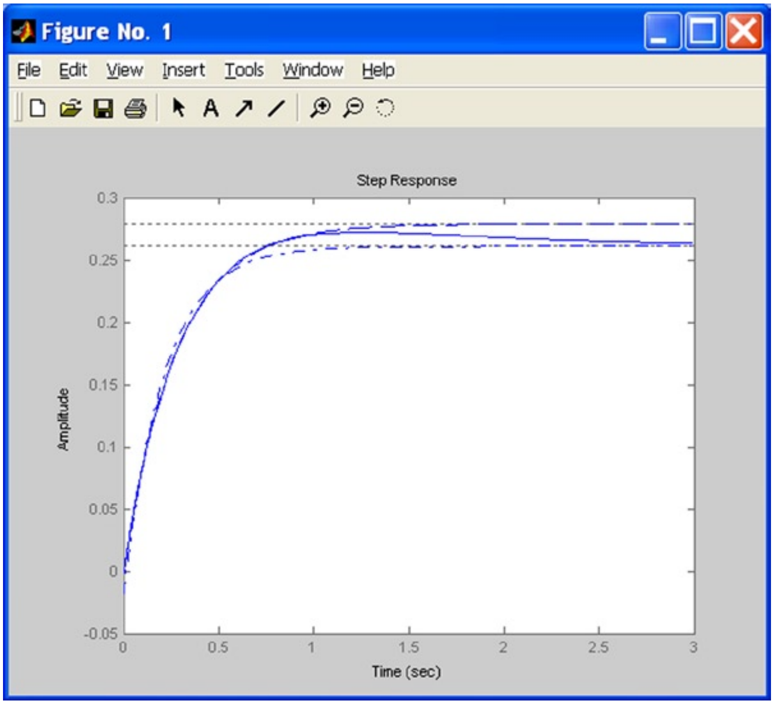


Figure 3-28.

EXERCISE 3-5

Calculate the covariance of response of the discrete SISO system defined by $H(z)$ and T_s below, corresponding to a Gaussian white noise of intensity $W = 5$.

$$H(z)=\frac{2z+1}{z^2+0.2z+0.5},\ T_s=0.1$$

```
>> sys = tf([2 1],[1 0.2 0.5],0.1)

Transfer function:
      2 z + 1
-----------------
z^2 + 0.2 z + 0.5

Sampling time: 0.1
>>p = covar(sys,5)

p =

30.3167
```

EXERCISE 3-6

Plot the poles and zeros of the continuous-time transfer function system defined by

$$H(s) = \frac{2s^2 + 5s + 1}{s^2 + 2s + 3}.$$

```
>> H = tf([2 5 1],[1 2 3])
Transfer function:

2 s^2 + 5s + 1
--------------
s ^ 2 + 2s + 3

>> pzmap (H)
>> sgrid
```

Figure 3-29 shows the result.

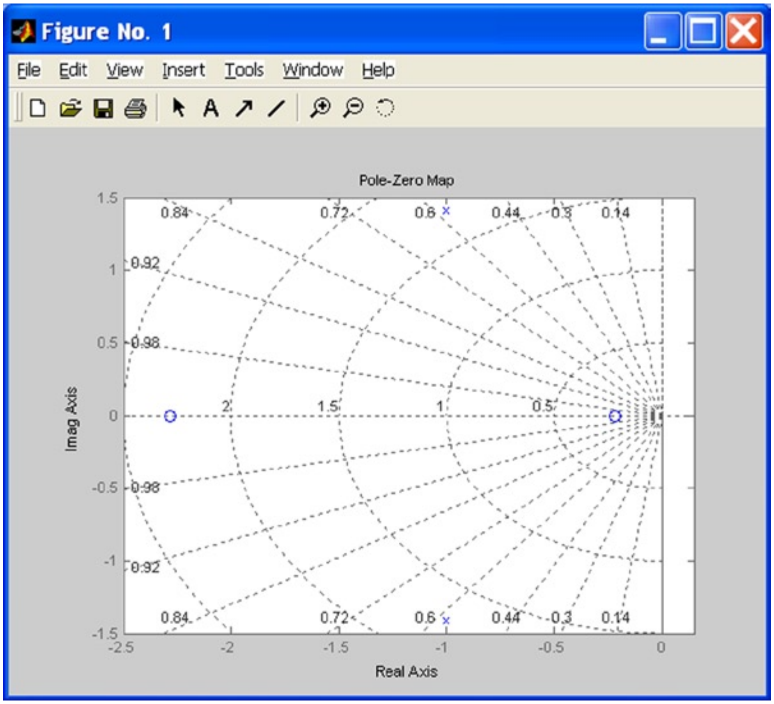


Figure 3-29.

EXERCISE 3-7

Consider the diagram in Figure 3-30 in which the matrices of the state-space model sys2 are given by:

$$A=[-9.0201, 17.7791; -1.6943, 3.2138];$$
$$B=[-.5112,\ .5362; -0.002, -1.8470];$$
$$C=[-3.2897, 2.4544; -13.5009, 18.0745];$$
$$D=[-.5476, -.1410; -.6459, .2958].$$

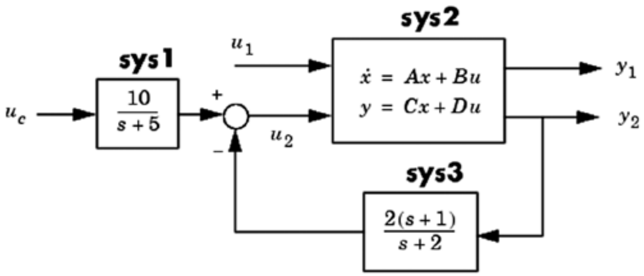


Figure 3-30.

First join the unconnected blocks, and secondly find the state-space model for the global interconnection given by the matrix Q = [3.1, – 4; 4,3,0] with inputs = [1,2] and outputs = [2,3].

The blocks are joined using the following syntax:

```
>> A = [ -9.0201,  17.7791; -1.6943  3.2138 ];
B = [ -.5112,  .5362;  -.002  -1.8470];
C = [ -3.2897,  2.4544;  -13.5009  18.0745];
D = [-.5476,  -.1410;  -.6459  .2958 ];
>> sys1 = tf(10,[1 5],'inputname','uc')
sys2 = ss(A,B,C,D,'inputname',{'u1' 'u2'},...
'outputname',{'y1' 'y2'})
sys3 = zpk(-1,-2,2)

Transfer function from input "uc" to output:

  10
-----
s + 5

a =
            x1       x2
   x1    -9.02    17.78
   x2   -1.694    3.214

b =
             u1        u2
   x1   -0.5112    0.5362
   x2    -0.002    -1.847
```

```
c =
          x1     x2
   y1  -3.29  2.454
   y2  -13.5  18.07

d =
            u1       u2
   y1  -0.5476   -0.141
   y2  -0.6459   0.2958

Continuous-time model.

Zero/pole/gain:

2 (s+1)
-------
(s+2)
```

The union of the unconnected blocks is created as follows:

```
sys = append(sys1,sys2,sys3)
a =
           x1       x2       x3       x4
   x1      -5        0        0        0
   x2       0    -9.02    17.78        0
   x3       0   -1.694    3.214        0
   x4       0        0        0       -2

b =
            uc        u1        u2         ?
   x1        4         0         0         0
   x2        0   -0.5112    0.5362         0
   x3        0    -0.002    -1.847         0
   x4        0         0         0     1.414

c =
           x1       x2       x3       x4
   ?      2.5        0        0        0
   y1       0    -3.29    2.454        0
   y2       0    -13.5    18.07        0
   ?        0        0        0   -1.414

d =
            uc        u1        u2         ?
   ?         0         0         0         0
   y1        0   -0.5476    -0.141         0
   y2        0   -0.6459    0.2958         0
   ?         0         0         0         2

Continuous-time model.
```

We then obtain the state-space model for the global interconnection.

```
>> Q = [3, 1, -4; 4, 3, 0];
>> inputs = [1 2];
>> outputs = [2 3];
>> sysc = connect(sys,Q,inputs,outputs)

a =
             x1        x2        x3        x4
   x1        -5         0         0         0
   x2    0.8422   0.07664     5.601    0.4764
   x3    -2.901    -33.03     45.16    -1.641
   x4    0.6571       -12     16.06    -1.628

b =
             uc        u1
   x1         4         0
   x2         0    -0.076
   x3         0    -1.501
   x4         0   -0.5739

c =
             x1        x2        x3        x4
   y1   -0.2215    -5.682     5.657   -0.1253
   y2    0.4646    -8.483     11.36    0.2628

d =
             uc        u1
   y1         0    -0.662
   y2         0   -0.4058

Continuous-time model.
```

EXERCISE 3-8

Plot the unit impulse response of the second-order state-space model defined below and store the results in an array with output response and simulation time.

The model is defined as follows:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} -0.5572 & -0.7814 \\ 0.7814 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 1 & -1 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

The requested plot is obtained by using the following syntax (see Figure 3-31):

```
>> a = [-0.5572 -0.7814;0.7814  0];
b = [1 -1;0 2];
c = [1.9691  6.4493];
sys = ss(a,b,c,0);
impulse (sys)
```

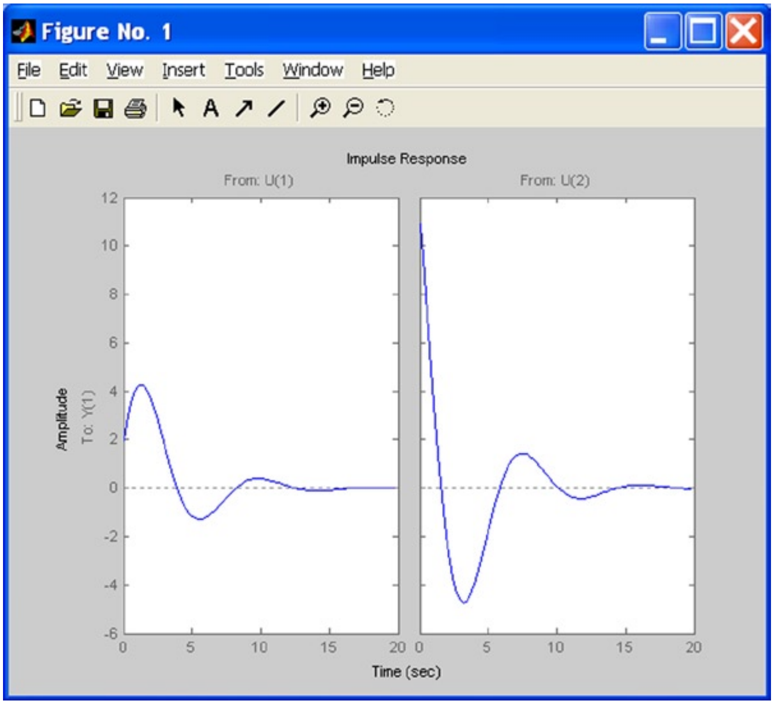


Figure 3-31.

The output response and simulation time are obtained using the syntax:

```
>> [y t] = impulse (sys)

y(:,:,1) =

1.9691
2.6831
3.2617
3.7059
4.0197
4.2096
      .
      .

y(:,:,2) =

10.9295
9.4915
7.9888
6.4622
4.9487
      .
      .
```

EXERCISE 3-9

Graph and simulate the response of the system with transfer function *H*(*s*) defined below to a square signal of period 4 seconds, sampling every 0.1 seconds and every 10 seconds.

$$H(s)=\begin{bmatrix}\dfrac{2s^2+5s+1}{s^2+2s+3}\\ \dfrac{s-1}{s^2+s+5}\end{bmatrix}$$

We begin by generating the square signal with *gensys* and then perform the simulation using *lsim* (see Figure 3-32) as follows:

```
>> [u,t] = gensig('square',4,10,0.1);
>> H = [tf([2 5 1],[1 2 3]) ; tf([1 -1],[1 1 5])]
lsim(H,u,t)
```

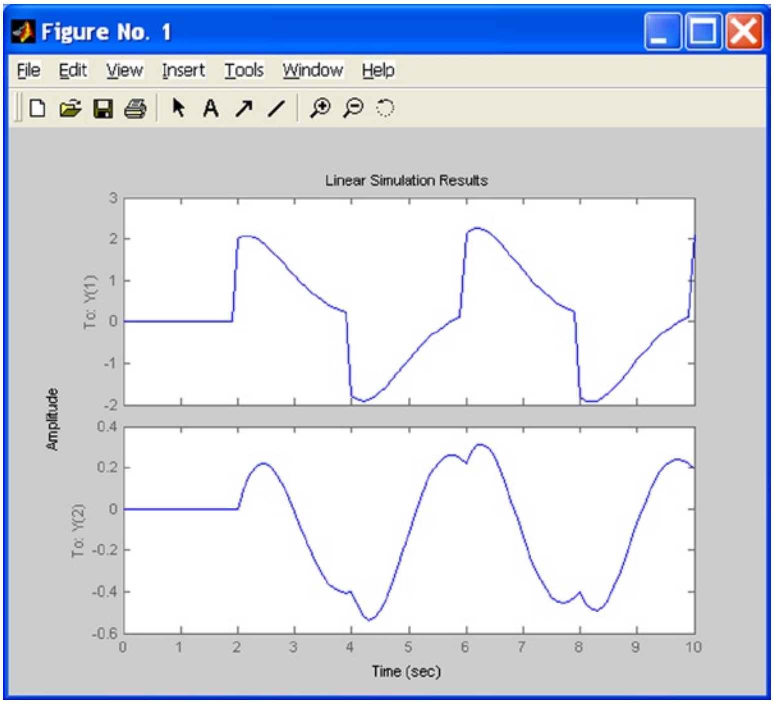


Figure 3-32.

```
Transfer function from input to output...

    2 s ^ 2 + 5 s + 1
#1:  ---------------
     s ^ 2 + 2 s + 3

          s 1
#2:  -----------
    s ^ 2 + s + 5
```