Control System Toolbox™

Getting Started Guide

R2014**b**
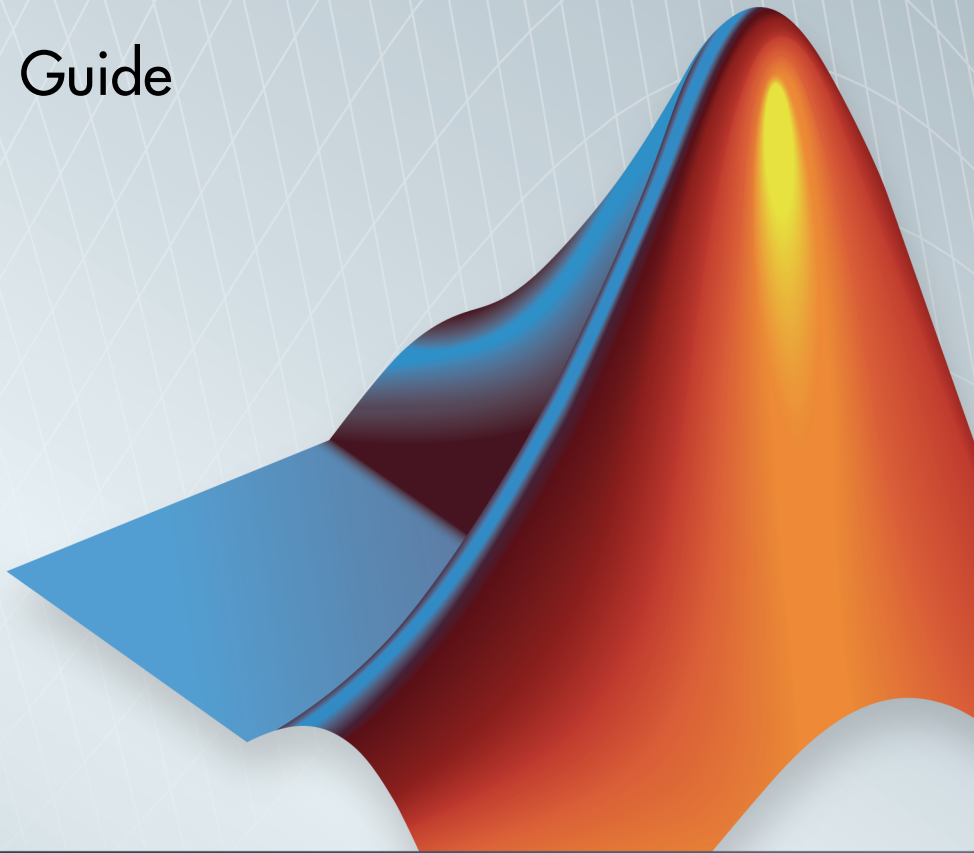
MATLAB®
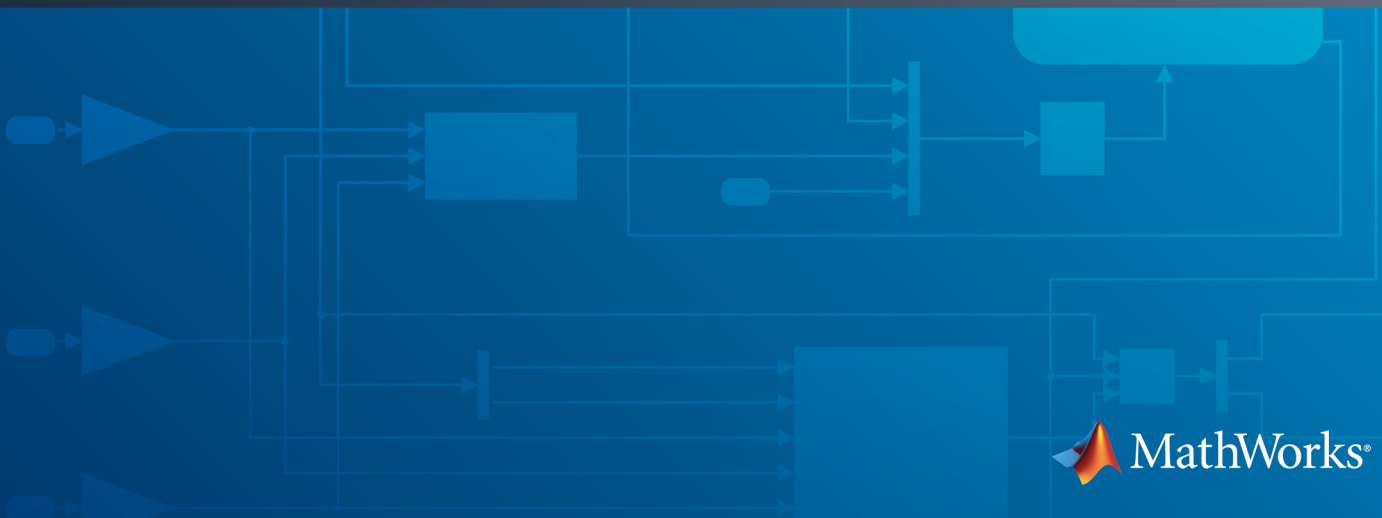
◢ MathWorks®

How to Contact MathWorks

Latest news: www.mathworks.com

Sales and services: www.mathworks.com/sales_and_services

User community: www.mathworks.com/matlabcentral

Technical support: www.mathworks.com/support/contact_us

Phone: 508-647-7000

The MathWorks, Inc.
3 Apple Hill Drive
Natick, MA 01760-2098

Control System Toolbox™ Getting Started Guide

© COPYRIGHT 2000–2014 by The MathWorks, Inc.

Trademarks

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See www.mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

Patents

MathWorks products are protected by one or more U.S. patents. Please see www.mathworks.com/patents for more information.

Revision History

November 2002	First printing	New Version 5.0 (Release 12)
June 2001	Second printing	Revised for Version 5.1 (Release 12.1)
July 2002	Online only	Revised for Version 5.2 (Release 13)
June 2004	Online only	Revised for Version 6.0 (Release 14)
March 2005	Online only	Revised for Version 6.2 (Release 14SP2)
September 2005	Online only	Revised for Version 6.2.1 (Release 14SP3)
March 2006	Online only	Revised for Version 7.0 (Release 2006a)
September 2006	Online only	Revised for Version 7.1 (Release 2006b)
March 2007	Online only	Revised for Version 8.0 (Release 2007a)
September 2007	Online only	Revised for Version 8.0.1 (Release 2007b)
March 2008	Online only	Revised for Version 8.1 (Release 2008a)
October 2008	Third printing	Revised for Version 8.2 (Release 2008b)
March 2009	Online only	Revised for Version 8.3 (Release 2009a)
September 2009	Online only	Revised for Version 8.4 (Release 2009b)
March 2010	Online only	Revised for Version 8.5 (Release 2010a)
September 2010	Online only	Revised for Version 9.0 (Release 2010b)
April 2011	Online only	Revised for Version 9.1 (Release 2011a)
September 2011	Online only	Revised for Version 9.2 (Release 2011b)
March 2012	Online only	Revised for Version 9.3 (Release 2012a)
September 2012	Online only	Revised for Version 9.4 (Release 2012b)
March 2013	Online only	Revised for Version 9.5 (Release 2013a)
September 2013	Online only	Revised for Version 9.6 (Release 2013b)
March 2014	Online only	Revised for Version 9.7 (Release 2014a)
October 2014	Online only	Revised for Version 9.8 (Release 2014b)

Contents

Analyzing Models

3

Designing Compensators

4

x

Product Overview

Control System Toolbox Product Description

Design and analyze control systems

Control System Toolbox provides industry-standard algorithms and apps for systematically analyzing, designing, and tuning linear control systems. You can specify your system as a transfer function, state-space, zero-pole-gain, or frequency-response model. Apps and functions, such as step response plot and Bode plot, let you visualize system behavior in time domain and frequency domain. You can tune compensator parameters using automatic PID controller tuning, Bode loop shaping, root locus method, LQR/LQG design, and other interactive and automated techniques. You can validate your design by verifying rise time, overshoot, settling time, gain and phase margins, and other requirements.

Key Features

- Transfer-function, state-space, zero-pole-gain, and frequency-response models of linear systems
- Series, parallel, feedback, and general block-diagram connection of linear models
- Step response, Nyquist plot, and other time- and frequency-domain tools for analyzing stability and performance measures
- Root locus, Bode diagrams, LQR, LQG, and other classical and state-space control system design techniques
- Automatic tuning of PID controllers
- Model representation conversion, continuous-time model discretization, and low-order approximation of high-order systems
- LAPACK and SLICOT algorithms optimized for accuracy and performance

2

Building Models

Linear (LTI) Models

What Is a Plant?

Typically, control engineers begin by developing a mathematical description of the dynamic system that they want to control. The system to be controlled is called a *plant*. As an example of a plant, this section uses the DC motor. This section develops the differential equations that describe the electromechanical properties of a DC motor with an inertial load. It then shows you how to use the Control System Toolbox functions to build linear models based on these equations.

Linear Model Representations

You can use Control System Toolbox functions to create the following model representations:

- State-space models (SS) of the form

$$\frac{dx}{dt} = Ax + Bu$$
$$y = Cx + Du$$

where A, B, C, and D are matrices of appropriate dimensions, x is the state vector, and u and y are the input and output vectors.

- Transfer functions (TF), for example,

$$H(s) = \frac{s+2}{s^2 + s + 10}$$

- Zero-pole-gain (ZPK) models, for example,

$$H(z) = 3\frac{(z+1+j)(z+1-j)}{(z+0.2)(z+0.1)}$$

- Frequency response data (FRD) models, which consist of sampled measurements of a system's frequency response. For example, you can store experimentally collected frequency response data in an FRD model.

Note The design of FRD models is a specialized subject that this guide does not address. See "Frequency Response Data (FRD) Models" for a discussion of this topic.

SISO Example: The DC Motor

A simple model of a DC motor driving an inertial load shows the angular rate of the load, $\omega(t)$, as the output and applied voltage, $v_{app}(t)$, as the input. The ultimate goal of this example is to control the angular rate by varying the applied voltage. This figure shows a simple model of the DC motor.
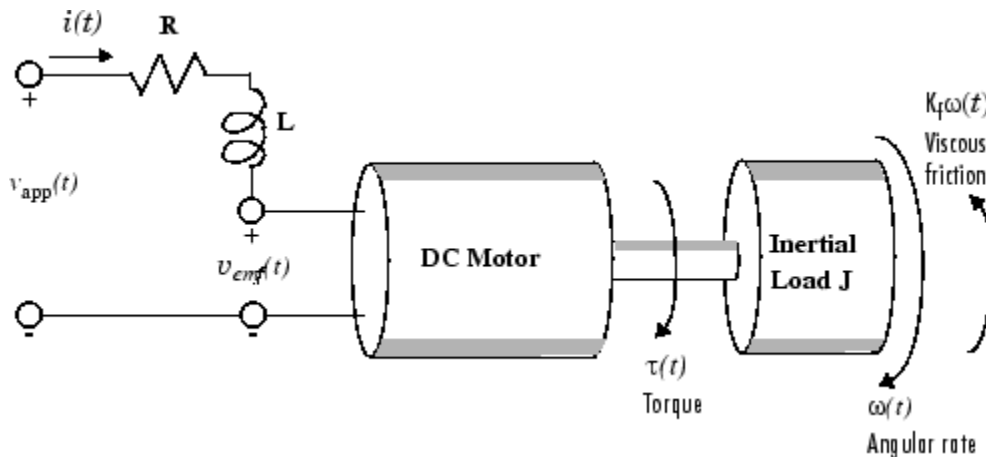
A Simple Model of a DC Motor Driving an Inertial Load

In this model, the dynamics of the motor itself are idealized; for instance, the magnetic field is assumed to be constant. The resistance of the circuit is denoted by R and the self-inductance of the armature by L. If you are unfamiliar with the basics of DC motor modeling, consult any basic text on physical modeling. With this simple model and basic laws of physics, it is possible to develop differential equations that describe the behavior of this electromechanical system. In this example, the relationships between electric potential and mechanical force are Faraday's law of induction and Ampère's law for the force on a conductor moving through a magnetic field.

Mathematical Derivation

The torque τ seen at the shaft of the motor is proportional to the current i induced by the applied voltage,

$$\tau(t) = K_m i(t)$$

where K_m, the armature constant, is related to physical properties of the motor, such as magnetic field strength, the number of turns of wire around the conductor coil, and so on. The back (induced) electromotive force, υ_{emf} , is a voltage proportional to the angular rate ω seen at the shaft,

$$\upsilon_{emf}(t) = K_b \omega(t)$$

where K_b, the emf constant, also depends on certain physical properties of the motor.

The mechanical part of the motor equations is derived using Newton's law, which states that the inertial load J times the derivative of angular rate equals the sum of all the torques about the motor shaft. The result is this equation,

$$J \frac{dw}{dt} = \sum \tau_i = -K_f \omega(t) + K_m i(t)$$

where $K_f \omega$ is a linear approximation for viscous friction.

Finally, the electrical part of the motor equations can be described by

$$\upsilon_{app}(t) - \upsilon_{emf}(t) = L \frac{di}{dt} + Ri(t)$$

or, solving for the applied voltage and substituting for the back emf,

$$\upsilon_{app}(t) = L\frac{di}{dt} + Ri(t) + K_b\omega(t)$$

This sequence of equations leads to a set of two differential equations that describe the behavior of the motor, the first for the induced current,

$$\frac{di}{dt} = -\frac{R}{L}i(t) - \frac{K_b}{L}\omega(t) + \frac{1}{L}\upsilon_{app}(t)$$

and the second for the resulting angular rate,

$$\frac{d\omega}{dt} = -\frac{1}{J}K_f\omega(t) + \frac{1}{J}K_mi(t)$$

State-Space Equations for the DC Motor

Given the two differential equations derived in the last section, you can now develop a state-space representation of the DC motor as a dynamic system. The current i and the angular rate ω are the two states of the system. The applied voltage, υ_{app}, is the input to the system, and the angular velocity ω is the output.

$$\frac{d}{dt}\begin{bmatrix} i \\ \omega \end{bmatrix} = \begin{bmatrix} -\dfrac{R}{L} & -\dfrac{K_b}{L} \\ \dfrac{K_m}{J} & -\dfrac{K_f}{J} \end{bmatrix} \cdot \begin{bmatrix} i \\ \omega \end{bmatrix} + \begin{bmatrix} \dfrac{1}{L} \\ 0 \end{bmatrix} \cdot \upsilon_{app}(t)$$

$$y(t) = \begin{bmatrix} 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} i \\ \omega \end{bmatrix} + \begin{bmatrix} 0 \end{bmatrix} \cdot \upsilon_{app}(t)$$

State-Space Representation of the DC Motor Example

Building SISO Models

After you develop a set of differential equations that describe your plant, you can construct SISO models using simple commands. The following sections discuss

• Constructing a state-space model of the DC motor

- Converting between model representations
- Creating transfer function and zero/pole/gain models

Constructing a State-Space Model of the DC Motor

Enter the following nominal values for the various parameters of a DC motor.

```
R= 2.0 % Ohms
L= 0.5 % Henrys
Km = .015 % torque constant
Kb = .015 % emf constant
Kf = 0.2 % Nms
J= 0.02 % kg.m^2
```

Given these values, you can construct the numerical state-space representation using the `ss` function.

```
A = [-R/L -Kb/L; Km/J -Kf/J]
B = [1/L; 0];
C = [0 1];
D = [0];
sys_dc = ss(A,B,C,D)
```

These commands return the following result:

```
a =
                    x1          x2
        x1          -4       -0.03
        x2        0.75         -10


b =
                    u1
        x1           2
        x2           0


c =
                    x1          x2
        y1           0           1


d =
                    u1
        y1           0
```

Converting Between Model Representations

Now that you have a state-space representation of the DC motor, you can convert to other model representations, including transfer function (TF) and zero/pole/gain (ZPK) models.

Transfer Function Representation

You can use `tf` to convert from the state-space representation to the transfer function. For example, use this code to convert to the transfer function representation of the DC motor.

```
sys_tf = tf(sys_dc)

Transfer function:
      1.5
------------------
s^2 + 14 s + 40.02
```

Zero/Pole/Gain Representation

Similarly, the `zpk` function converts from state-space or transfer function representations to the zero/pole/gain format. Use this code to convert from the state-space representation to the zero/pole/gain form for the DC motor.

```
sys_zpk = zpk(sys_dc)

Zero/pole/gain:
       1.5
------------------
(s+4.004) (s+9.996)
```

Note The state-space representation is best suited for numerical computations. For highest accuracy, convert to state space prior to combining models and avoid the transfer function and zero/pole/gain representations, except for model specification and inspection.

Constructing Transfer Function and Zero/Pole/Gain Models

In the DC motor example, the state-space approach produces a set of matrices that represents the model. If you choose a different approach, you can construct the corresponding models using `tf`, `zpk`, `ss`, or `frd`.

```
sys = tf(num,den)              % Transfer function
sys = zpk(z,p,k)               % Zero/pole/gain
sys = ss(a,b,c,d)              % State-space
sys = frd(response,frequencies) % Frequency response data
```

For example, you can create the transfer function by specifying the numerator and denominator with this code.

```
sys_tf = tf(1.5,[1 14 40.02])

Transfer function:
       1.5
------------------
s^2 + 14 s + 40.02
```

Alternatively, if you want to create the transfer function of the DC motor directly, use these commands.

```
s = tf('s');
sys_tf = 1.5/(s^2+14*s+40.02)
```

These commands result in this transfer function.

```
Transfer function:
        1.5
-------------------
s^2 + 14 s + 40.02
```

To build the zero/pole/gain model, use this command.

```
sys_zpk = zpk([],[-9.996 -4.004], 1.5)
```

This command returns the following zero/pole/gain representation.

```
Zero/pole/gain:
       1.5
------------------
(s+9.996) (s+4.004)
```

Constructing Discrete Time Systems

The Control System Toolbox software provides full support for discrete-time systems. You can create discrete systems in the same way that you create analog systems; the

only difference is that you must specify a sample time period for any model you build. For example,

```
sys_disc = tf(1, [1 1], .01);
```

creates a SISO model in the transfer function format.

```
Transfer function:
  1
-----
z + 1

Sampling time: 0.01
```

Adding Time Delays to Discrete-Time Models

You can add time delays to discrete-time models by specifying an input delay, output delay, or I/O delay when building the model. The time delay must be a nonnegative integer that represents a multiple of the sampling time. For example,

```
sys_delay = tf(1, [1 1], 0.01,'ioDelay',5)
```

returns a system with an I/O delay of 5 s.

```
Transfer function:
             1
z^(-5) * -----
           z + 1

Sampling time: 0.01
```

Adding Delays to Linear Models

You can add time delays to linear models by specifying an input delay, output delay, or I/O delay when building a model. For example, to add an I/O delay to the DC motor, use this code.

```
sys_tfdelay = tf(1.5,[1 14 40.02],'ioDelay',0.05)
```

This command constructs the DC motor transfer function, but adds a 0.05 second delay.

```
Transfer function:
                   1.5
```

```
exp(-0.05*s) * -----------------
               s^2 + 14 s + 40.02
```

For a complete description of adding time delays to models and closing loops with time delays, see "Models with Time Delays".

LTI Objects

For convenience, the Control System Toolbox software uses custom data structures called *LTI objects* to store model-related data. For example, the variable sys_dc created for the DC motor example is called an *SS object*. There are also TF, ZPK, and FRD objects for transfer function, zero/pole/gain, and frequency data response models respectively. The four LTI objects encapsulate the model data and enable you to manipulate linear systems as single entities rather than as collections of vectors or matrices.

To see what LTI objects contain, use the get command. This code describes the contents of sys_dc from the DC motor example.

```
get(sys_dc)
                a: [2x2 double]
                b: [2x1 double]
                c: [0 1]
                d: 0
                e: []
        StateName: {2x1 cell}
    InternalDelay: [0x1 double]
               Ts: 0
       InputDelay: 0
      OutputDelay: 0
        InputName: {''}
       OutputName: {''}
       InputGroup: [1x1 struct]
      OutputGroup: [1x1 struct]
             Name: ''
            Notes: {}
         UserData: []
```

You can manipulate the data contained in LTI objects using the set command; see the Control System Toolbox online reference pages for descriptions of set and get.

Another convenient way to set or retrieve LTI model properties is to access them directly using dot notation. For example, if you want to access the value of the A matrix, instead of using get, you can type

```
sys_dc.a
```

at the MATLAB® prompt. This notation returns the A matrix.

```
ans =

   -4.0000    -0.0300
    0.7500   -10.0000
```

Similarly, if you want to change the values of the A matrix, you can do so directly, as this code shows.

```
A_new = [-4.5 -0.05; 0.8 -12.0];
sys_dc.a = A_new;
```

MIMO Models

In this section...
"MIMO Example: State-Space Model of Jet Transport Aircraft" on page 2-12
"Constructing MIMO Transfer Functions" on page 2-14
"Accessing I/O Pairs in MIMO Systems" on page 2-16

MIMO Example: State-Space Model of Jet Transport Aircraft

You can use the same functions that apply to SISO systems to create multiple-input, multiple-output (MIMO) models. This example shows how to build a MIMO model of a jet transport. Because the development of a physical model for a jet aircraft is lengthy, only the state-space equations are presented here. See any standard text in aviation for a more complete discussion of the physics behind aircraft flight.

The jet model during cruise flight at MACH = 0.8 and H = 40,000 ft. is

```
A = [-0.0558   -0.9968     0.0802     0.0415
      0.5980   -0.1150    -0.0318          0
     -3.0500    0.3880    -0.4650          0
           0    0.0805     1.0000         0];

B = [ 0.0073          0
     -0.4750     0.0077
      0.1530     0.1430
           0          0];

C = [0     1     0     0
     0     0     0     1];

D = [0     0
     0     0];
```

Use the following commands to specify this state-space model as an LTI object and attach names to the states, inputs, and outputs.

```
states = {'beta' 'yaw' 'roll' 'phi'};
inputs = {'rudder' 'aileron'};
outputs = {'yaw rate' 'bank angle'};

sys_mimo = ss(A,B,C,D,'statename',states,...
```

```
'inputname',inputs,...
'outputname',outputs);
```

You can display the LTI model by typing `sys_mimo`.

```
sys_mimo

a =
                    beta          yaw         roll          phi
        beta     -0.0558      -0.9968       0.0802       0.0415
         yaw       0.598       -0.115      -0.0318            0
        roll       -3.05        0.388       -0.465            0
         phi           0       0.0805            1            0


b =
                  rudder      aileron
        beta      0.0073            0
         yaw      -0.475       0.0077
        roll       0.153        0.143
         phi           0            0


c =
                    beta          yaw         roll          phi
    yaw rate           0            1            0            0
  bank angle           0            0            0            1


d =
                  rudder      aileron
    yaw rate           0            0
  bank angle           0            0

Continuous-time model.
```

The model has two inputs and two outputs. The units are radians for `beta` (sideslip angle) and `phi` (bank angle) and radians/sec for `yaw` (yaw rate) and `roll` (roll rate). The rudder and aileron deflections are in degrees.

As in the SISO case, use `tf` to derive the transfer function representation.

```
tf(sys_mimo)

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

```
                 -0.475 s^3 - 0.2479 s^2 - 0.1187 s - 0.05633
 yaw rate:  -------------------------------------------------
            s^4 + 0.6358 s^3 + 0.9389 s^2 + 0.5116 s + 0.003674

                      0.1148 s^2 - 0.2004 s - 1.373
 bank angle:  -------------------------------------------------
              s^4 + 0.6358 s^3 + 0.9389 s^2 + 0.5116 s + 0.003674

Transfer function from input "aileron" to output...
            0.0077 s^3 - 0.0005372 s^2 + 0.008688 s + 0.004523
 yaw rate:  -------------------------------------------------
            s^4 + 0.6358 s^3 + 0.9389 s^2 + 0.5116 s + 0.003674

                      0.1436 s^2 + 0.02737 s + 0.1104
 bank angle:  -------------------------------------------------
              s^4 + 0.6358 s^3 + 0.9389 s^2 + 0.5116 s + 0.003674
```

Constructing MIMO Transfer Functions

MIMO transfer functions are two-dimensional arrays of elementary SISO transfer functions. There are two ways to specify MIMO transfer function models:

- Concatenation of SISO transfer function models
- Using `tf` with cell array arguments

Concatenation of SISO Models

Consider the following single-input, two-output transfer function.

$$H(s) = \begin{bmatrix} \dfrac{s-1}{s+1} \\ \dfrac{s+2}{s^2+4s+5} \end{bmatrix}$$

You can specify $H(s)$ by concatenation of its SISO entries. For instance,

```
h11 = tf([1 -1],[1 1]);
h21 = tf([1 2],[1 4 5]);
```

or, equivalently,

```
s = tf('s')
```

```
h11 = (s-1)/(s+1);
h21 = (s+2)/(s^2+4*s+5);
```

can be concatenated to form *H*(*s*).

```
H = [h11; h21]
```

This syntax mimics standard matrix concatenation and tends to be easier and more readable for MIMO systems with many inputs and/or outputs.

Using the tf Function with Cell Arrays

Alternatively, to define MIMO transfer functions using `tf`, you need two cell arrays (say, `N` and `D`) to represent the sets of numerator and denominator polynomials, respectively. See "Cell Arrays" in the MATLAB documentation for more details on cell arrays.

For example, for the rational transfer matrix *H*(*s*), the two cell arrays `N` and `D` should contain the row-vector representations of the polynomial entries of

$$N(s) = \begin{bmatrix} s-1 \\ s+2 \end{bmatrix} \qquad D(s) = \begin{bmatrix} s+1 \\ s^2 + 4s + 5 \end{bmatrix}$$

You can specify this MIMO transfer matrix *H*(*s*) by typing

```
N = {[1 -1];[1 2]};   % Cell array for N(s)
D = {[1 1];[1 4 5]}; % Cell array for D(s)
H = tf(N,D)
```

These commands return the following result:

```
Transfer function from input to output...
      s - 1
 #1:  -----
      s + 1

         s + 2
 #2:  -------------
      s^2 + 4 s + 5
```

Notice that both `N` and `D` have the same dimensions as *H*. For a general MIMO transfer matrix *H*(*s*), the cell array entries `N{i,j}` and `D{i,j}` should be row-vector representations of the numerator and denominator of *H*$_{ij}$(*s*), the *ij*th entry of the transfer matrix *H*(*s*).

Accessing I/O Pairs in MIMO Systems

After you define a MIMO system, you can access and manipulate I/O pairs by specifying input and output pairs of the system. For instance, if `sys_mimo` is a MIMO system with two inputs and three outputs,

```
sys_mimo(3,1)
```

extracts the subsystem, mapping the first input to the third output. Row indices select the outputs and column indices select the inputs. Similarly,

```
sys_mimo(3,1) = tf(1,[1 0])
```

redefines the transfer function between the first input and third output as an integrator.

Arrays of Linear Models

You can specify and manipulate collections of linear models as single entities using LTI arrays. For example, if you want to vary the Kb and Km parameters for the DC motor and store the resulting state-space models, use this code.

```
K = [0.1 0.15 0.2]; % Several values for Km and Kb
A1 = [-R/L -K(1)/L; K(1)/J -Kf/J];
A2 = [-R/L -K(2)/L; K(2)/J -Kf/J];
A3 = [-R/L -K(3)/L; K(3)/J -Kf/J];
sys_lti(:,:,1)= ss(A1,B,C,D);
sys_lti(:,:,2)= ss(A2,B,C,D);
sys_lti(:,:,3)= ss(A3,B,C,D);
```

The number of inputs and outputs must be the same for all linear models encapsulated by the LTI array, but the model order (number of states) can vary from model to model within a single LTI array.

The LTI array sys_lti contains the state-space models for each value of K. Type sys_lti to see the contents of the LTI array.

```
Model sys_lti(:,:,1,1)
======================

  a =
                        x1           x2
            x1          -4          -0.2
            x2           5          -10
.
.
.
Model sys_lti(:,:,2,1)
======================

  a =
                        x1           x2
            x1          -4          -0.3
            x2         7.5          -10
.
.
.
Model sys_lti(:,:,3,1)
======================
```

```
  a =
                              x1              x2
            x1              -4              -0.4
            x2              10               -10
.
.
.
3x1 array of continuous-time state-space models.
```

You can manipulate the LTI array like any other object. For example,

```
step(sys_lti)
```

produces a plot containing step responses for all three state-space models.
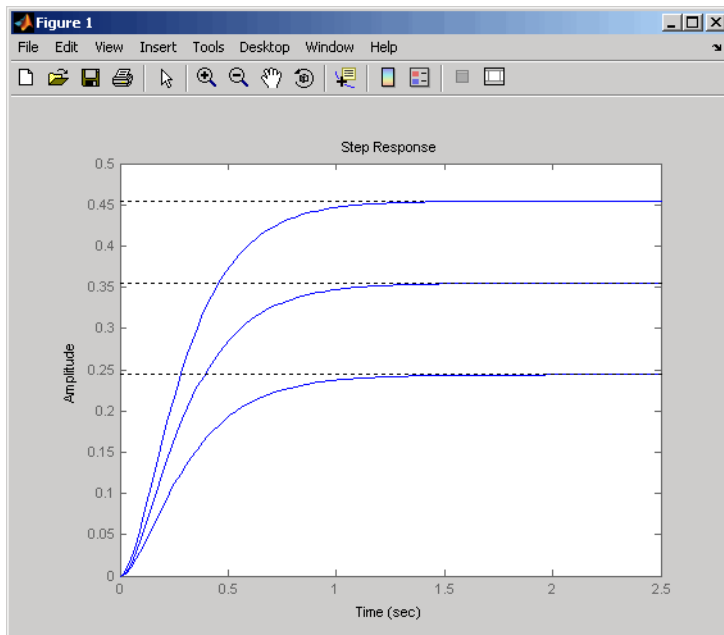
Step Responses for an LTI Array Containing Three Models

LTI arrays are useful for performing batch analysis on an entire set of models. For more information, see "Model Arrays".

Model Characteristics

You can use the Control System Toolbox commands to query model characteristics such as the I/O dimensions, poles, zeros, and DC gain. These commands apply to both continuous- and discrete-time models. Their LTI-based syntax is summarized in the table below.

Commands to Query Model Characteristics

Command	Description
size(model_name)	Number of inputs and outputs
ndims(model_name)	Number of dimensions
isct(model_name)	Returns 1 for continuous systems
isdt(model_name)	Returns 1 for discrete systems
hasdelay(model_name)	True if system has delays
pole(model_name)	System poles
zero(model_name)	System (transmission) zeros
dcgain(model_name)	DC gain
norm(model_name)	System norms (H_2 and L_∞)
covar(model_name,W)	Covariance of response to white noise
bandwidth(model_name)	Frequency response bandwidth
order(model_name)	LTI model order
pzmap(model_name)	Compute pole-zero map of LTI models
damp(model_name)	Natural frequency and damping of system poles
class(model_name)	Create object or return class of object
isa(model_name)	Determine whether input is object of given class
isempty(model_name)	Determine whether LTI model is empty
isproper(model_name)	Determine whether LTI model is proper
issiso(model_name)	Determine whether LTI model is single-input/single-output (SISO)
isstable(model_name)	Determine whether system is stable
reshape(model_name)	Change shape of LTI array

Interconnecting Linear Models

In this section...
"Arithmetic Operations for Interconnecting Models" on page 2-20
"Feedback Interconnections" on page 2-21

Arithmetic Operations for Interconnecting Models

You can perform arithmetic on LTI models, such as addition, multiplication, or concatenation. Addition performs a parallel interconnection. For example, typing

```
tf(1,[1 0]) + tf([1 1],[1 2])   % 1/s + (s+1)/(s+2)
```

produces this transfer function.

```
Transfer function:
s^2 + 2 s + 2
-------------
  s^2 + 2 s
```

Multiplication performs a series interconnection. For example, typing

```
2 * tf(1,[1 0])*tf([1 1],[1 2])   % 2*1/s*(s+1)/(s+2)
```

produces this cascaded transfer function.

```
Transfer function:
2 s + 2
---------
s^2 + 2 s
```

If the operands are models of different types, the resulting model type is determined by precedence rules; see "Rules That Determine Model Type" for more information.

For more information about model arithmetic functions, see "Catalog of Model Interconnections".

You can also use the `series` and `parallel` functions as substitutes for multiplication and addition, respectively.

Equivalent Ways to Interconnect Systems

Operator	Function	Resulting Transfer Function
`sys1 + sys2`	`parallel(sys1,sys2)`	Systems in parallel
`sys1 - sys2`	`parallel(sys1,-sys2)`	Systems in parallel
`sys1 * sys2`	`series(sys2,sys1)`	Cascaded systems

Feedback Interconnections

You can use the `feedback` and `lft` functions to derive closed-loop models. For example,

```
sys_f = feedback(tf(1,[1 0]), tf([1 1],[1 2])
```

computes the closed-loop transfer function from *r* to *y* for the feedback loop shown below. The result is

```
Transfer function:
    s + 2
-------------
s^2 + 3 s + 1
```

This figure shows the interconnected system in block diagram format.
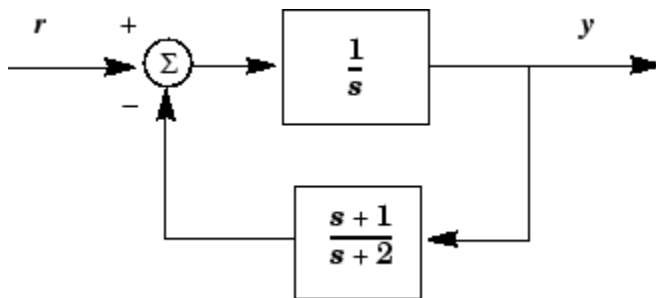
Feedback Interconnection

You can use the `lft` function to create more complicated feedback structures. This function constructs the linear fractional transformation of two systems. See the reference page for more information.

Converting Between Continuous- and Discrete- Time Systems

In this section...

Available Commands for Continuous/Discrete Conversion

The commands c2d, d2c, and d2d perform continuous to discrete, discrete to continuous, and discrete to discrete (resampling) conversions, respectively.

```
sysd = c2d(sysc,Ts)   % Discretization w/ sample period Ts
sysc = d2c(sysd)      % Equivalent continuous-time model
sysd1= d2d(sysd,Ts)   % Resampling at the period Ts
```

Available Methods for Continuous/Discrete Conversion

Various discretization/interpolation methods are available, including zero-order hold (default), first-order hold, Tustin approximation with or without prewarping, and matched zero-pole. For example,

```
sysd = c2d(sysc,Ts,'foh')  % Uses first-order hold
sysc = d2c(sysd,'tustin')  % Uses Tustin approximation
```

Digitizing the Discrete DC Motor Model

You can digitize the DC motor plant using the c2d function and selecting an appropriate sample time. Choosing the right sample time involves many factors, including the performance you want to achieve, the fastest time constant in your system, and the speed at which you expect your controller to run. For this example, choose a time constant of 0.01 second. See "SISO Example: The DC Motor" on page 2-3 for the construction of the SS object sys_dc.

```
Ts=0.01;
sysd=c2d(sys_dc,Ts)
```

```
a =
                        x1           x2
         x1       0.96079   -0.00027976
         x2      0.006994       0.90484


b =
                        u1
         x1      0.019605
         x2   7.1595e-005




c =
                        x1           x2
         y1             0            1


d =
                        u1
         y1             0

Sampling time: 0.01
Discrete-time model.
```

To see the discrete-time zero-pole gain for the digital DC motor, use `zpk` to convert the model.
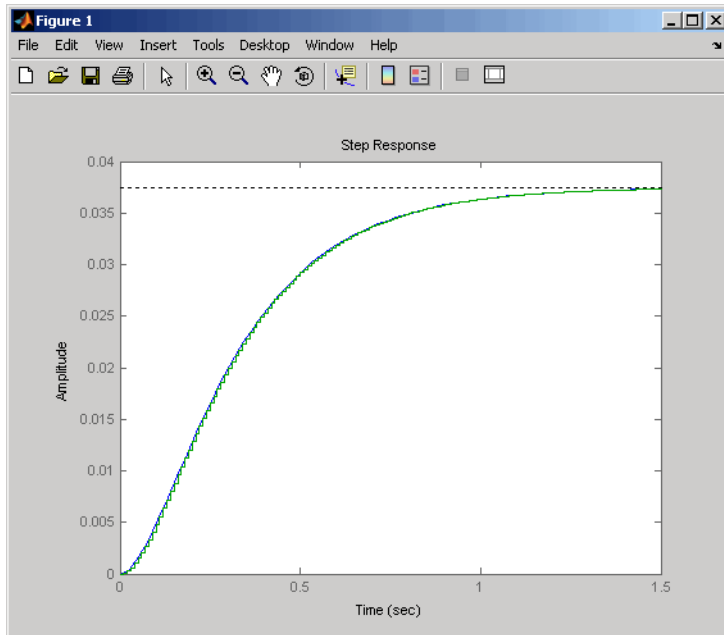
```
fd=zpk(sysd)

Zero/pole/gain:
7.1595e-005 (z+0.9544)
-----------------------
 (z-0.9608) (z-0.9049)

Sampling time: 0.01
```

You can compare the step responses of sys_dc and sysd by typing

```
step(sys_dc,sysd)
```
This figure shows the result.

Note the step response match. Continuous and FOH-discretized step responses match for models without internal delays.

Reducing Model Order

In this section...

Model Order Reduction Commands

You can derive reduced-order SISO and MIMO models with the commands shown in the following table.

Model Order Reduction Commands	
hsvd	Compute Hankel singular values of LTI model
balred	Reduced-order model approximation
minreal	Minimal realization (pole/zero cancellation)
balreal	Compute input/output balanced realization
modred	State deletion in I/O balanced realization
sminreal	Structurally minimal realization

Techniques for Reducing Model Order

To reduce the order of a model, you can perform any of the following actions:

• Eliminate states that are structurally disconnected from the inputs or outputs using sminreal.

  Eliminating structurally disconnected states is a good first step in model reduction because the process is cheap and does not involve any numerical computation.

• Compute a low-order approximation of your model using balred.

• Eliminate cancelling pole/zero pairs using minreal.

Example: Gasifier Model

This example presents a model of a gasifier, a device that converts solid materials into gases. The original model is nonlinear.

Loading the Model

To load a linearized version of the model, type

```
load ltiexamples
```

at the MATLAB prompt; the gasifier example is stored in the variable named `gasf`. If you type

```
size(gasf)
```

you get in return

```
State-space model with 4 outputs, 6 inputs, and 25 states.
```

SISO Model Order Reduction

You can reduce the order of a single I/O pair to understand how the model reduction tools work before attempting to reduce the full MIMO model as described in "MIMO Model Order Reduction" on page 2-30.

This example focuses on a single input/output pair of the gasifier, input 5 to output 3.
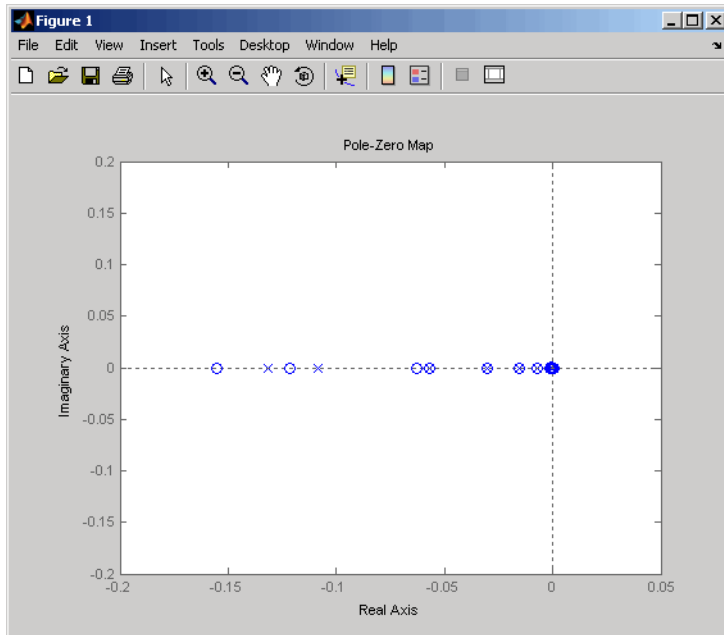
```
sys35 = gasf(3,5);
```

Before attempting model order reduction, inspect the pole and zero locations by typing

```
pzmap(sys35)
```

Zoom in near the origin on the resulting plot using the zoom feature or by typing

```
axis([-0.2 0.05 -0.2 0.2])
```

The following figure shows the results.

**Pole-Zero Map of the Gasifier Model (Zoomed In)**

Because the model displays near pole-zero cancellations, it is a good candidate for model reduction.

To find a low-order reduction of this SISO model, perform the following steps:

1   Select an appropriate order for your reduced system by examining the relative amount of energy per state using a Hankel singular value (HSV) plot. Type the command

    ```
    hsvd(sys35)
    ```
    to create the HSV plot.

    Changing to log scale using the right-click menu results in the following plot.
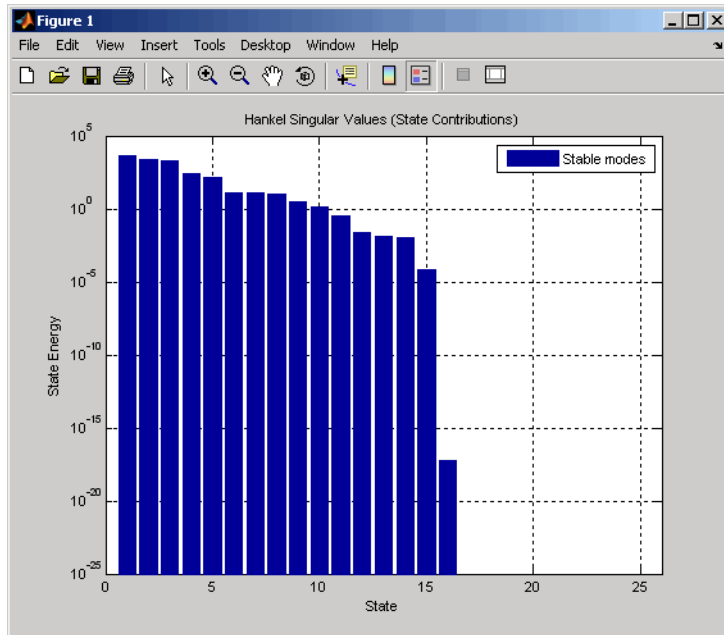
Small Hankel singular values indicate that the associated states contribute little to the I/O behavior. This plot shows that discarding the last 10 states (associated with the 10 smallest Hankel singular values) has little impact on the approximation error.

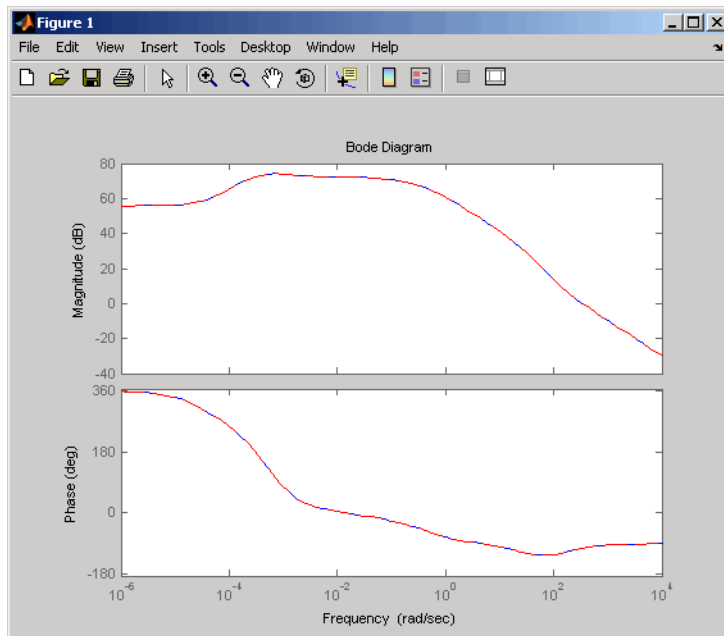**2** To remove the last 10 states and create a 15th order approximation, type

```
rsys35 = balred(sys35,15);
```
You can type `size(rsys35)` to see that your reduced system contains only 15 states.

3 Compare the Bode response of the full-order and reduced-order models using the bode command:

```
bode(sys35,'b',rsys35,'r--')
```
This figure shows the result.

As the overlap of the curves in the figure shows, the reduced model provides a good approximation of the original system.

You can try reducing the model order further by repeating this process and removing more states. Reduce the `gasf` model to 5th, 10th, and 15th orders all at once by typing the following command

```
rsys35 = balred(sys35,[5 10 15]);
```
Plot a bode diagram of these three reduced systems along with the full order system by typing

```
bode(sys35,'b',rsys35,'r--')
```
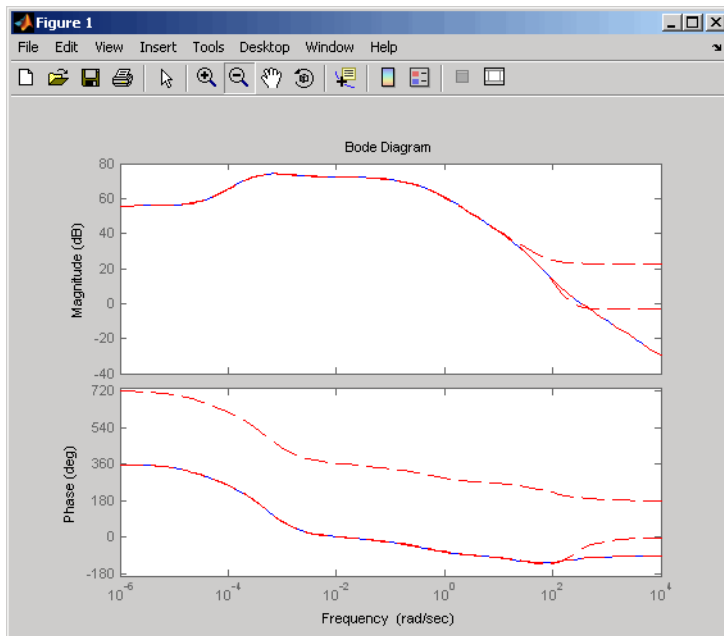
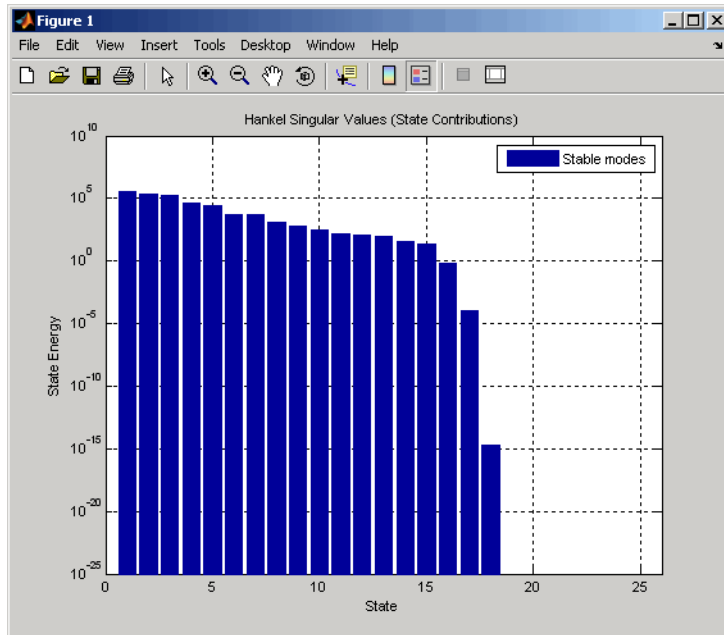Observe how the error increases as the order decreases.

MIMO Model Order Reduction

You can approximate MIMO models using the same steps as SISO models as follows:

1   Select an appropriate order for your reduced system by examining the relative
    amount of energy per state using a Hankel singular value (HSV) plot.

    Type

    ```
    hsvd(gasf)
    ```
    to create the HSV plot.

Discarding the last 8 states (associated with the 8 smallest Hankel singular values) should have little impact on the error in the resulting 17th order system.

2 To remove the last 8 states and create a 17th order MIMO system, type
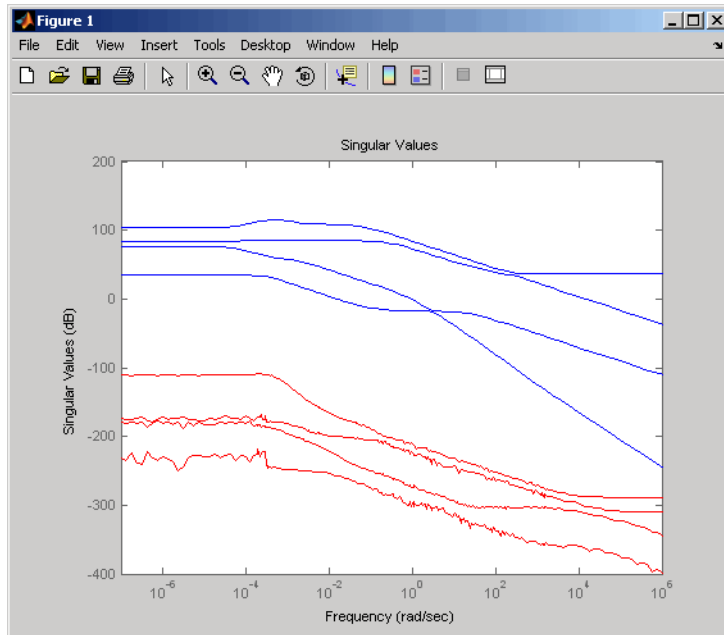
```
rsys=balred(gasf,17);
```
You can type `size(gasf)` to see that your reduced system contains only 17 states.

3 To facilitate visual inspection of the approximation error, use a singular value plot rather than a MIMO Bode plot. Type

```
sigma(gasf,'b',gasf-rsys,'r')
```
to create a singular value plot comparing the original system to the reduction error.

The reduction error is small compared to the original system so the reduced order model provides a good approximation of the original model.

Acknowledgment

3

Analyzing Models

Linear Analysis Using the LTI Viewer

In this example, you learn how to analyze the time- and frequency-domain responses of one or more linear models using the LTI Viewer GUI.

Before you can perform the analysis, you must have already created linear models in the MATLAB workspace. For information on how to create a model, see "Basic Models".

To perform linear analysis:

1   Open the LTI Viewer showing one or more models using the following syntax:

```
ltiview(model1,model2,...,modelN)
```
By default, this syntax opens a step response plot of your models, as shown in the following figure.
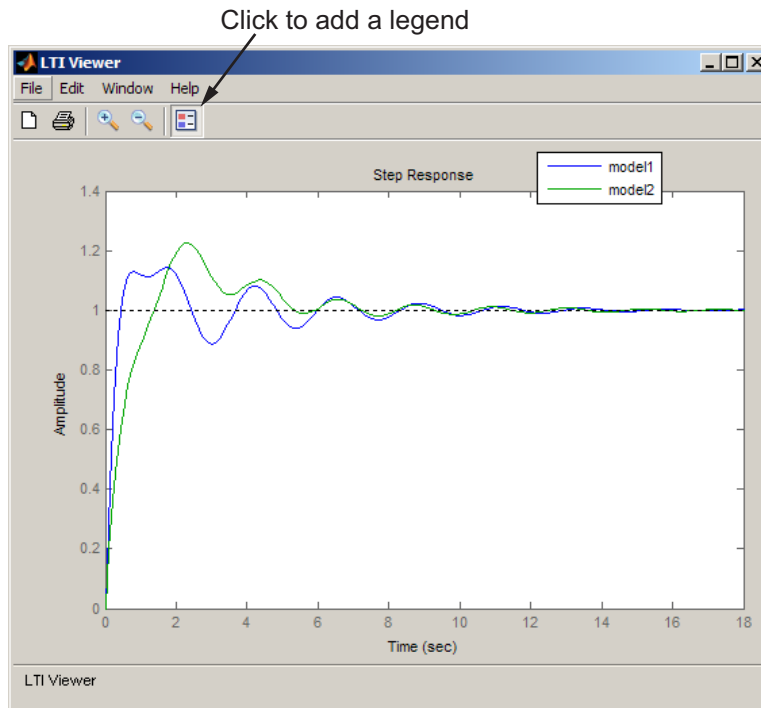
2   Add more plots to the LTI Viewer.

a Select **Edit** > **Plot Configurations**.

b In the Plot Configurations dialog box, select the number of plots to open.
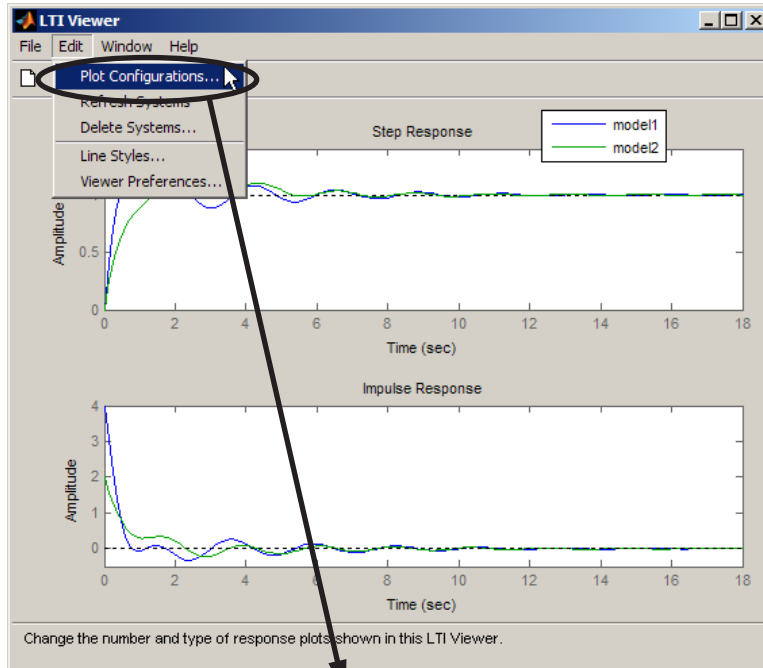
Select the number of plots to open
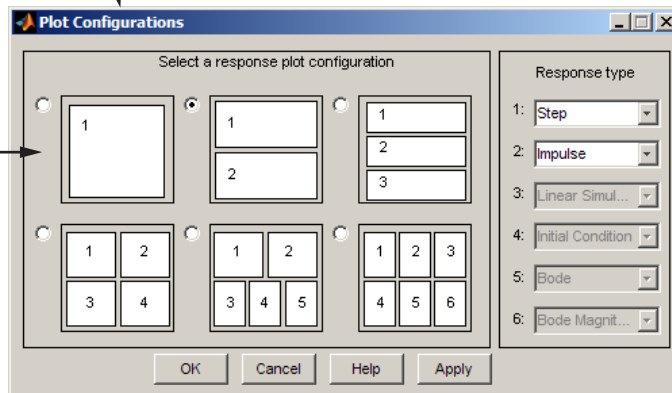
3 To view a different type of response on a plot, right-click and select a plot type.
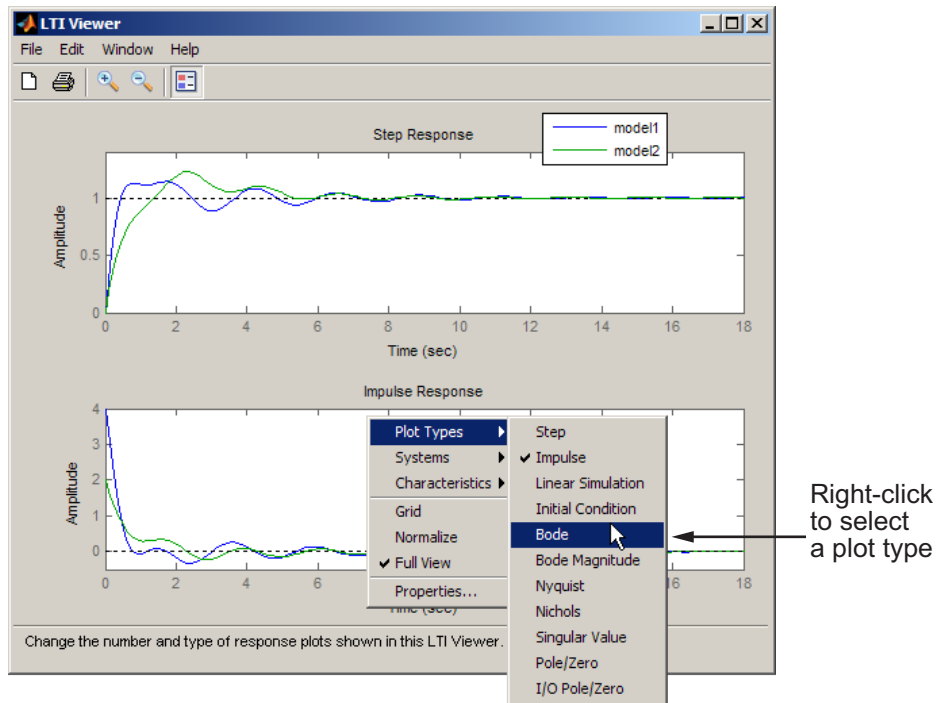
4 Analyze system performance. For example, you can analyze the peak response in the Bode plot and settling time in the step response plot.

a Right-click to select performance characteristics.

b Click on the dot that appears to view the characteristic value.
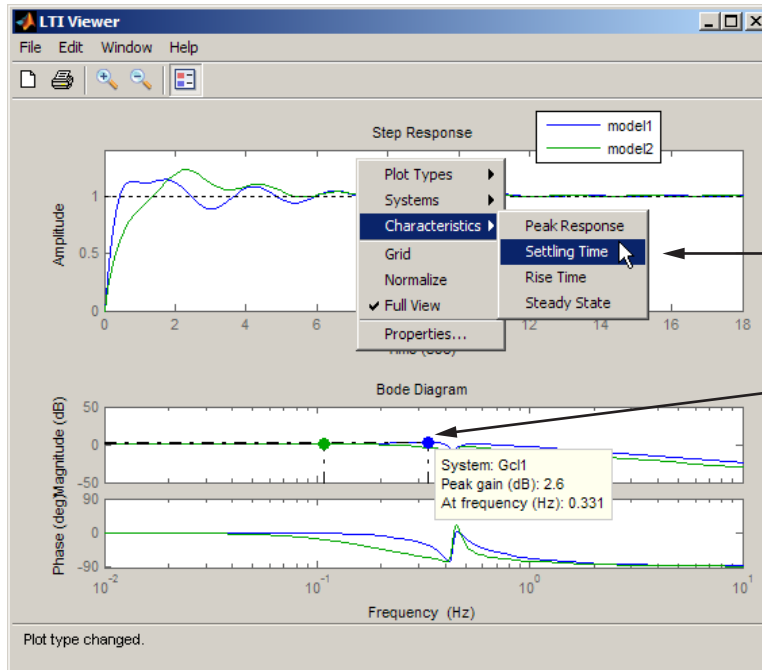
Right-click to show performance characteristics

Click dot to view value

LTI Viewer

In this section...

Plot Types Available in the LTI Viewer

The LTI Viewer is a GUI for viewing and manipulating the response plots of linear models. You can display the following plot types for linear models using the LTI Viewer:

- Step and impulse responses
- Bode and Nyquist plots
- Nichols plots
- Singular values of the frequency response
- Pole/zero plots
- Response to a general input signal
- Unforced response starting from given initial states (only for state-space models)

Note that time responses and pole/zero plots are not available for FRD models.

Note The LTI Viewer displays up to six different plot types simultaneously. You can also analyze the response plots of several linear models at once.

This figure shows an LTI Viewer with two response plots.

Click on these icons to open a new viewer, print, and zoom in and out, respectively.

Use the **File** menu to import models and the **Edit** menu to delete existing ones.

Right-click anywhere in a plot region to open a menu of options for that plot.

Left-click directly on a curve for information about the curve at that particular point.
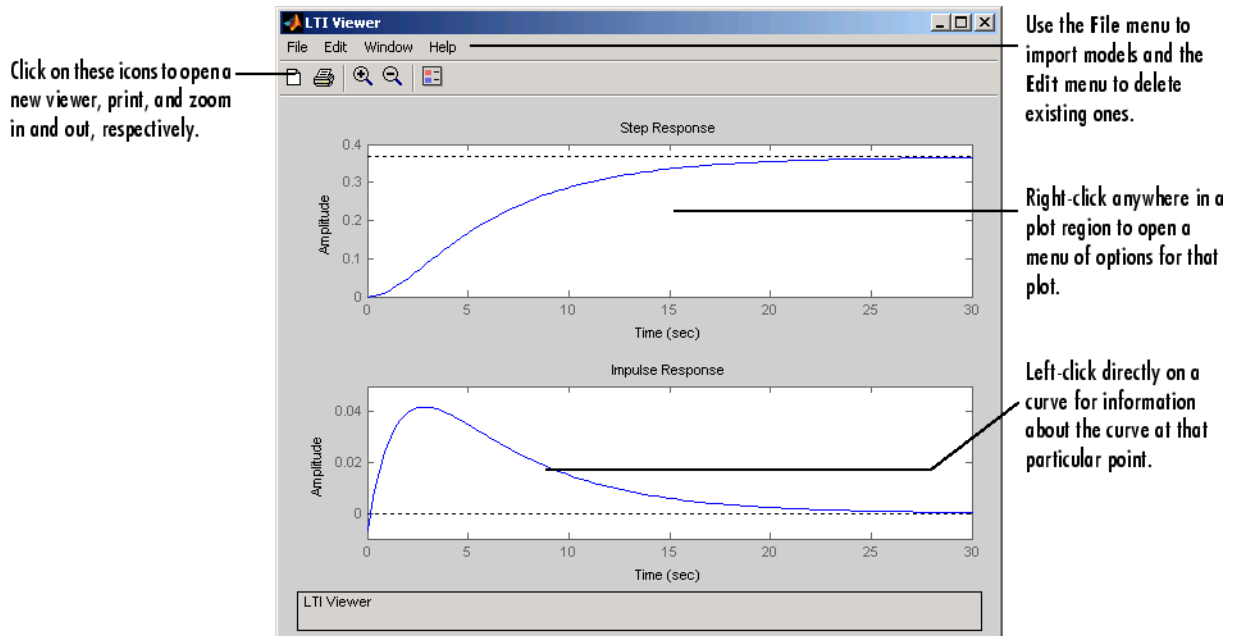
The LTI Viewer with Step and Impulse Response Plots

The next section presents an example that shows you how to import a system into the LTI Viewer and how to customize the viewer to fit your requirements.

Example: Time and Frequency Responses of the DC Motor

"SISO Example: The DC Motor" on page 2-3 presents a DC motor example. If you have not yet built that example, type

```
load ltiexamples
```

at the MATLAB prompt. This loads several LTI models, including a state-space representation of the DC motor called sys_dc.

Opening the LTI Viewer

To open the LTI Viewer, type

```
ltiview
```

This opens an LTI Viewer with an empty step response plot window by default.

Importing Models into the LTI Viewer

To import the DC motor model, select **Import** under the **File** menu. This opens the Import System Data dialog box, which lists all the models available in your MATLAB workspace.
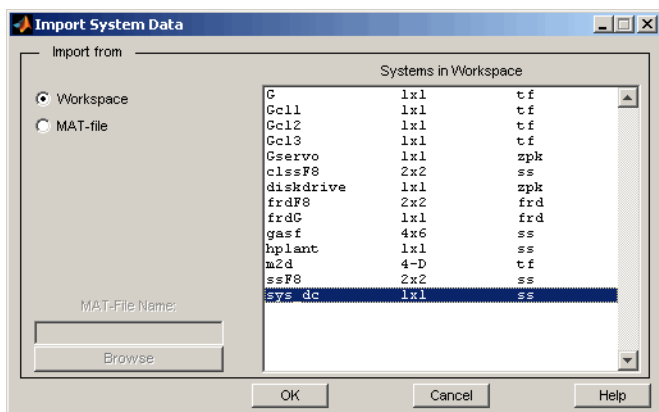
Import System Data Dialog Box with the DC Motor Model Selected

Select sys_dc from the list of available models and click **OK** to close the browser. This imports the DC motor model into the LTI Viewer.

To select more than one model at a time, do the following:

- To select individual (noncontiguous) models, select one model and hold down the **Ctrl** key while selecting additional models. To clear any models, hold down the **Ctrl** key while you click the highlighted model names.
- To select a list of contiguous models, select the first model and hold down the **Shift** key while selecting the last model you want in the list.

The figure below shows the LTI Viewer with a step response for the DC motor example.

This dashed line shows the steady-state value of the step response

The status bar describes the last action you have taken. It also provides information about accessing LTI Viewer menu options.
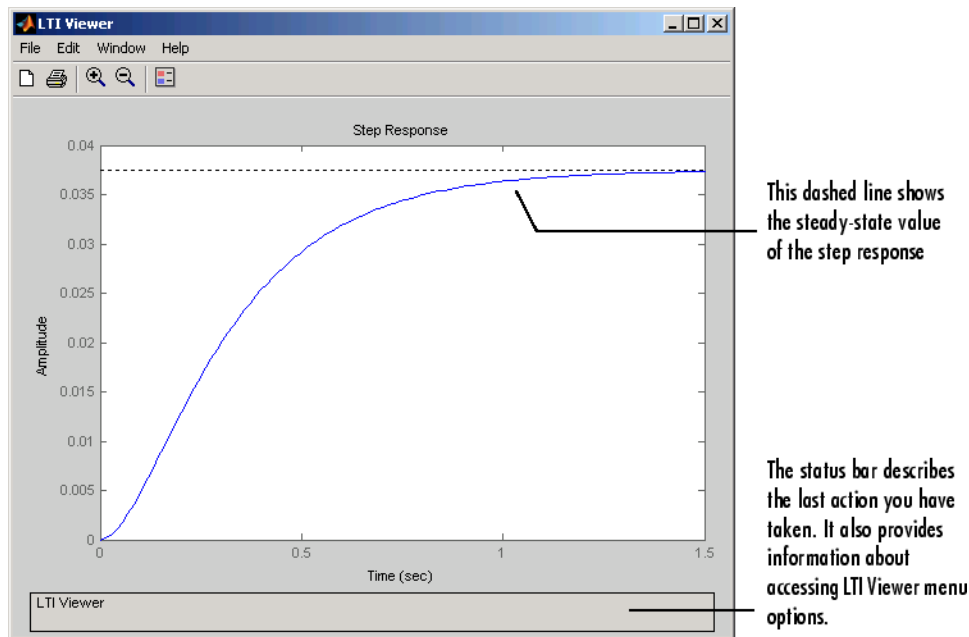
Step Response for the DC Motor Example in the LTI Viewer

Alternatively, you can open the LTI Viewer and import the DC motor example directly from the MATLAB prompt.

```
ltiview('step', sys_dc)
```

See the `ltiview` reference page for a complete list of options.

Right-Click Menus

The LTI Viewer provides a set of controls and options that you can access by right-clicking your mouse. Once you have imported a model into the LTI Viewer, the options you can select include

- **Plot Types** — Change the plot type. Available types include step, impulse, Bode, Bode magnitude, Nichols, Nyquist, and singular values plots.
- **Systems** — Select or clear any models that you included when you created the response plot.

- **Characteristics** — Add information about the plot. The characteristics available change from plot to plot. For example, Bode plots have stability margins available, but step responses have rise time and steady-state values available.

- **Grid** — Add grids to your plots.

- **Normalize** — Scale responses to fit the view (only available for time-domain plot types).

- **Full View** — Use automatic limits to make the entire curve visible.

- **Properties** — Open the Property Editor.
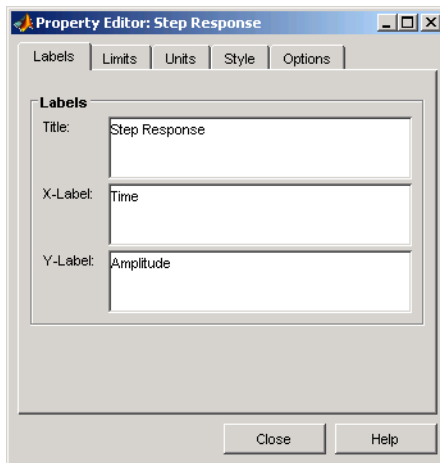
You can use this editor to customize various attributes of your plot. See "Customizing Plot Properties and Preferences" for a full description of the Property Editor.

Alternatively, you can open the Property Editor by double-clicking in an empty region of the response plot.

Displaying Response Characteristics on a Plot

For example, to see the rise time for the DC motor step response, right-click and select **Characteristics > Rise Time**.
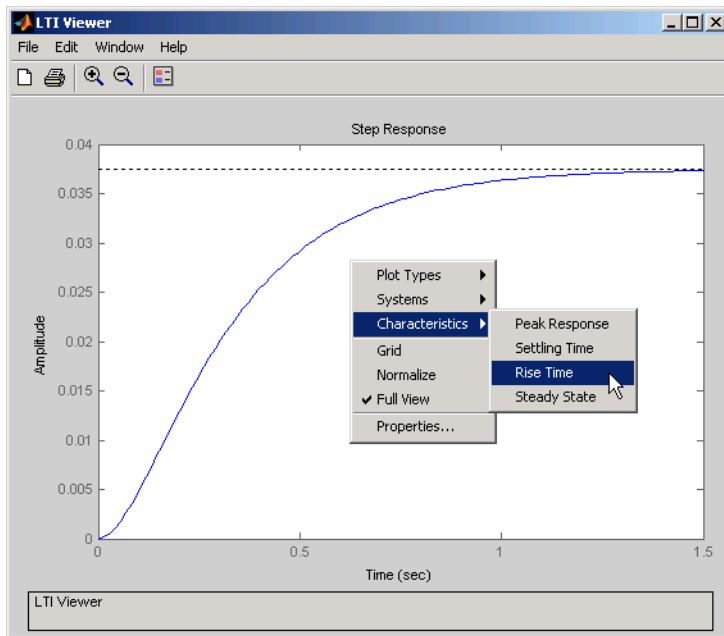
Using Right-Click Menus to Display the Rise Time for a Step Response

By default, the rise time is defined as the amount of time it takes the step response to go from 10% to 90% of the steady-state value. You can change this range in the **options** tab of the property editor. For more information on the property editor, see "Customizing Plot Properties and Preferences".

The LTI Viewer calculates and displays the rise time for the step response.

The dot marks the 90% value for the rise time.

This line shows when the response reaches 10% of its steady-state value.

This line shows when the response reaches 90% of its steady-state value.
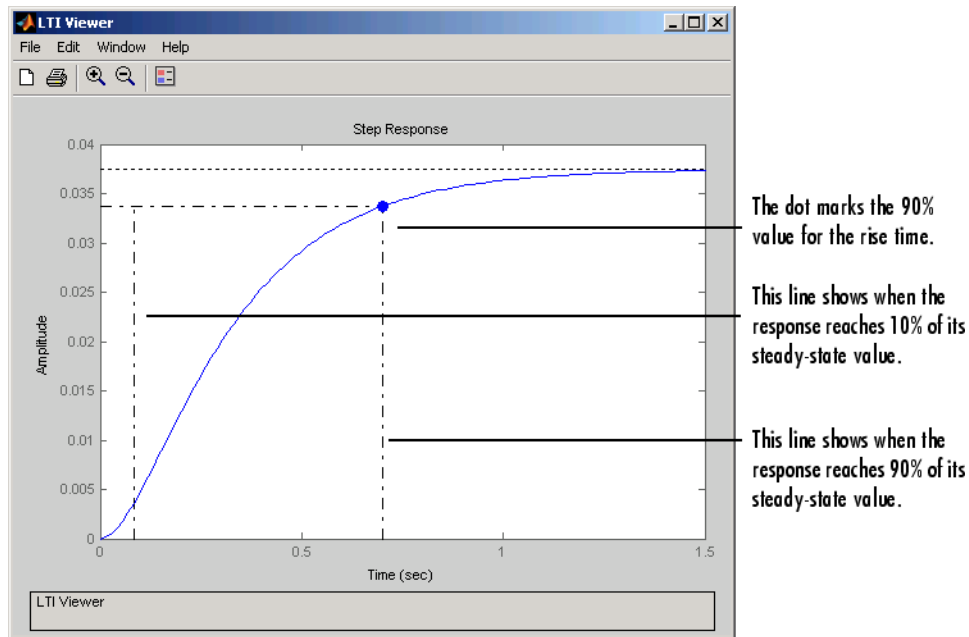
DC Motor Step Response with the Rise Time Displayed

To display the values of any plot characteristic marked on a plot, place your mouse on the blue dot that marks the characteristic. This opens a *data marker* with the relevant information displayed. To make the marker persistent, left-click the blue dot.

For example, this figure shows the rise time value for the DC motor step response.
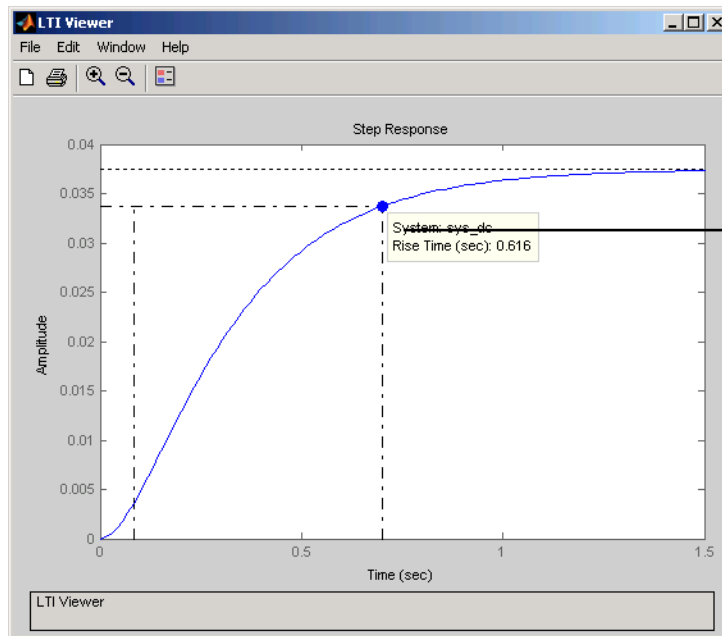
Place your cursor over the blue dot to display a data marker with the system name and the 90% rise time value.

Left-click the blue dot to make the data marker persistent.

Using Your Mouse to Get the Rise Time Values

Note that you can left-click anywhere on a particular plot line to see the response values of that plot at that point. You must either place your cursor over the blue dot or left-click, however, if you want to see the rise time value.

For more information about data markers, see "Data Markers" on page 3-40.

Changing Plot Type

You can view other plots using the right-click menus in the LTI Viewer. For example, if you want to see the open loop Bode plots for the DC motor model, select **Plot Type** and then **Bode** from the right-click menu.
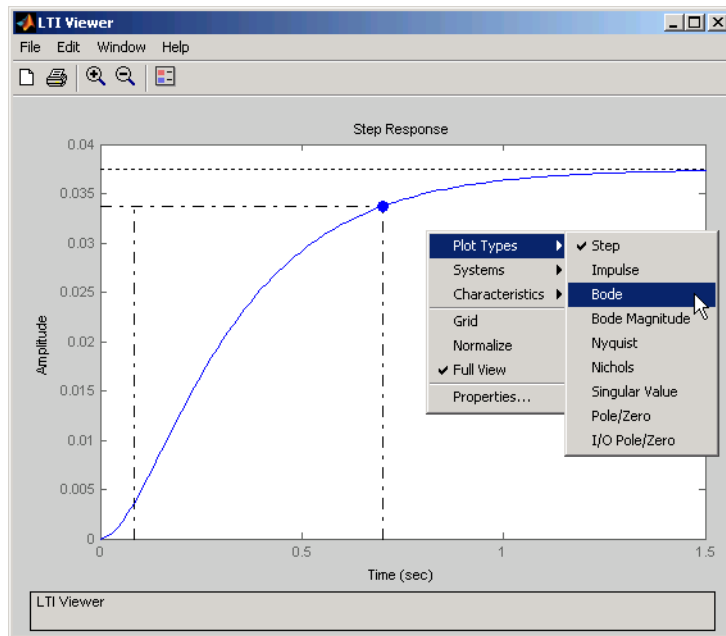
Changing the Step Response to a Bode Plot

Selecting **Bode** changes the step response to a Bode plot for the DC motor model.
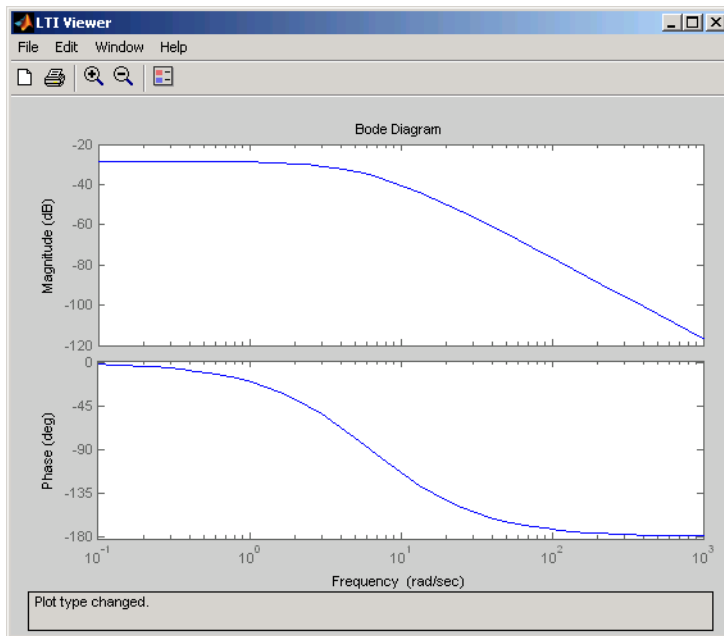
Bode Plot for the DC Motor Model

Showing Multiple Response Types

If you want to see, for example, both a step response and a Bode plot at the same time, you have to reconfigure the LTI Viewer. To view different response types in a single LTI Viewer, select **Plot Configuration**s under the **Edit** menu. This opens the Plot Configurations dialog box.
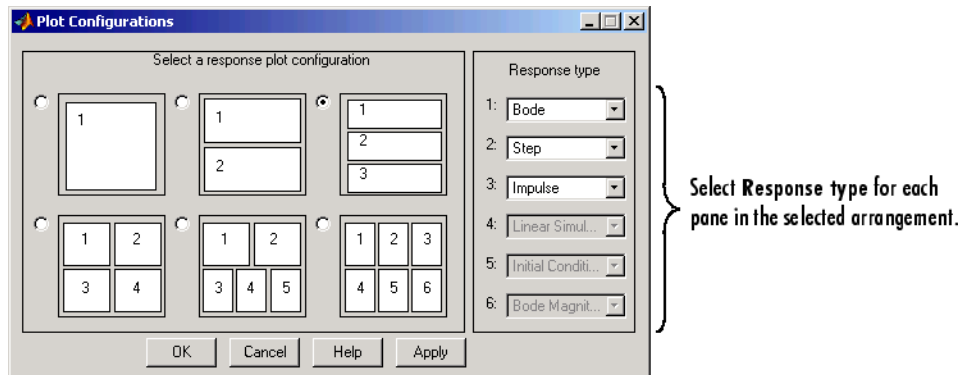
Using the Plot Configurations Dialog Box to Reconfigure the LTI Viewer

You can select up to six plots in one viewer. Choose the **Response type** for each plot area from the right-side menus. There are nine available plot types:

- Step
- Impulse
- Linear Simulation
- Initial Condition
- Bode (magnitude and phase)
- Bode Magnitude (only)
- Nyquist
- Nichols
- Singular Value
- Pole/Zero
- I/O pole/zero

Comparing Multiple Models

This section shows you how to import and manipulate multiple models in one LTI Viewer. For example, if you have designed a set of compensators to control a system, you can compare the closed-loop step responses and Bode plots using the LTI Viewer.

A sample set of closed-loop transfer function models is included (along with some other models) in the MAT-file `ltiexamples.mat`. Type

```
load ltiexamples
```

to load the provided transfer functions. The three closed-loop transfer function models, `Gcl1`, `Gcl2`, and `Gcl3`, are for a satellite attitude controller.

In this example, you analyze the response plots of the `Gcl1` and `Gcl2` transfer functions.

Initializing the LTI Viewer with Multiple Plots

To load the two models `Gcl1` and `Gcl2` into the LTI Viewer, select **Import** under the **File** menu and select the desired models in the LTI Browser. See "Importing Models into the LTI Viewer" on page 3-8 for a description of how to select groups of models. If necessary, you can reconfigure the viewer to display both the step responses and the Bode plots of the two systems using the Viewer Configuration dialog box. See "Showing Multiple Response Types" on page 3-15 for a discussion of this feature.

Alternatively, you can open an LTI Viewer with both systems and both the step responses and Bode plots displayed. To do this, type

```
ltiview({'step';'bode'},Gcl1,Gcl2)
```

Either approach opens the following LTI Viewer.
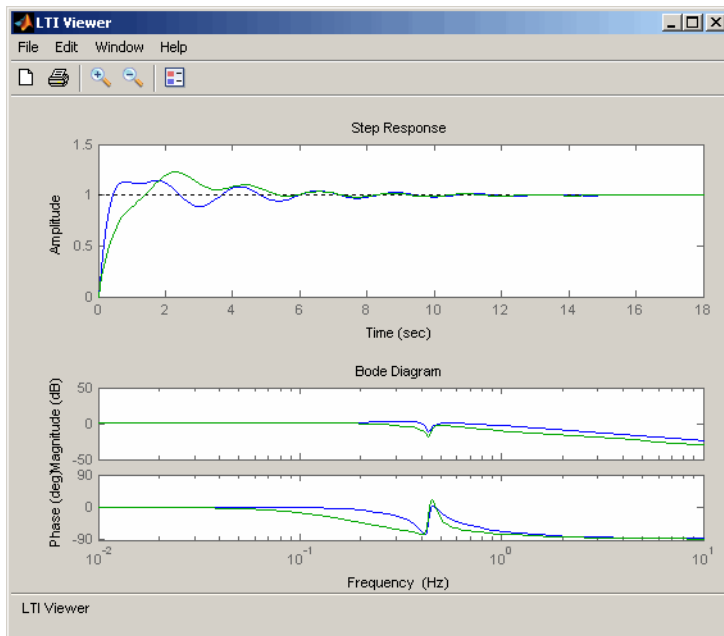
Multiple Response Plots in a Single LTI Viewer

Inspecting Response Characteristics

To mark the settling time on the step responses presented in this example, do the following:

- Right-click anywhere in the plot region of the step response plots. This opens the right-click menu list in the plot region.

- Select **Characteristics** > **Settling Time**.

To mark the stability margins of the Bode plot in this example, right-click and select **Characteristics** > **Minimum Stability Margins**.

Your LTI Viewer should now look like this.
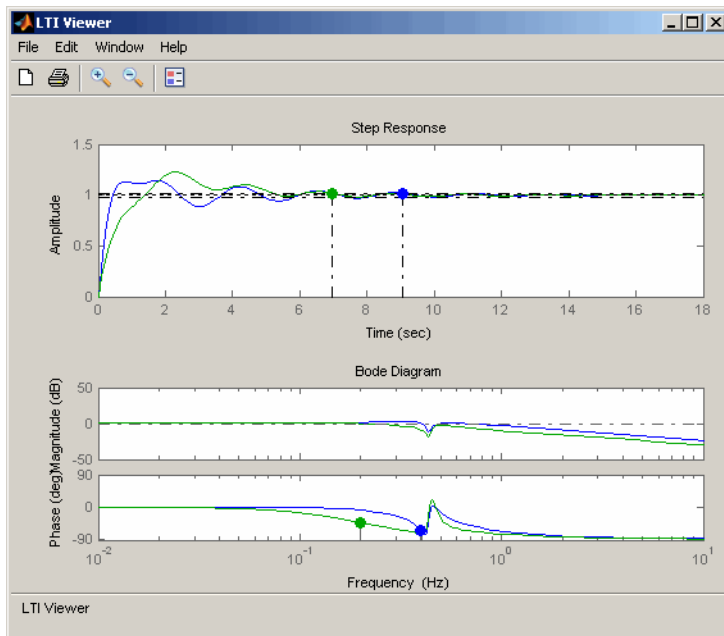
Multiple Plots with Response Characteristics Added

The minimum stability margins, meaning the smallest magnitude phase and gain margins, are displayed as green and blue markers on the Bode phase diagram. If you want to see all the gain and phase margins of a system, right-click and select **Characteristics** > **Minimum Stability Margins**.

Toggling Model Visibility

If you have imported more than one model, you can select and clear which models to plot in the LTI Viewer using right-click menus. For example, if you import the following three models into the viewer, you can choose to view any combination of the three you want.

```
s=tf('s');
sys1=1/(s^2+s+1);
sys2=1/(s^2+s+2);
sys3=1/(s^2+s+3);
```

This figure shows how to clear the second of the three models using right-click menu options.

Using Right-Click Menus to Select/Clear Plotted Systems

The **Systems** menu lists all the imported models. A system is selected if a check mark is visible to the left of the system name.

Simulate Models with Arbitrary Inputs and Initial Conditions

In this section...
"What is the Linear Simulation Tool?" on page 3-21
"Opening the Linear Simulation Tool" on page 3-21
"Working with the Linear Simulation Tool" on page 3-22
"Importing Input Signals" on page 3-24
"Example: Loading Inputs from a Microsoft Excel Spreadsheet" on page 3-26
"Example: Importing Inputs from the Workspace" on page 3-27
"Designing Input Signals" on page 3-31
"Specifying Initial Conditions" on page 3-33

What is the Linear Simulation Tool?

You can use the Linear Simulation Tool to simulate linear models with arbitrary input signals and initial conditions.

The Linear Simulation Tool lets you do the following:

- Import input signals from the MATLAB workspace.
- Import input signals from a MAT-file, Microsoft® Excel® spreadsheet, ASCII flat-file, comma-separated variable file (CSV), or text file.
- Generate arbitrary input signals in the form of a sine wave, square wave, step function, or white noise.
- Specify initial states for state-space models.

  Default initial states are zero.

Opening the Linear Simulation Tool

To open the Linear Simulation Tool, do one of the following:

- In the LTI Viewer, right-click the plot area and select **Plot Types > Linear Simulation**.
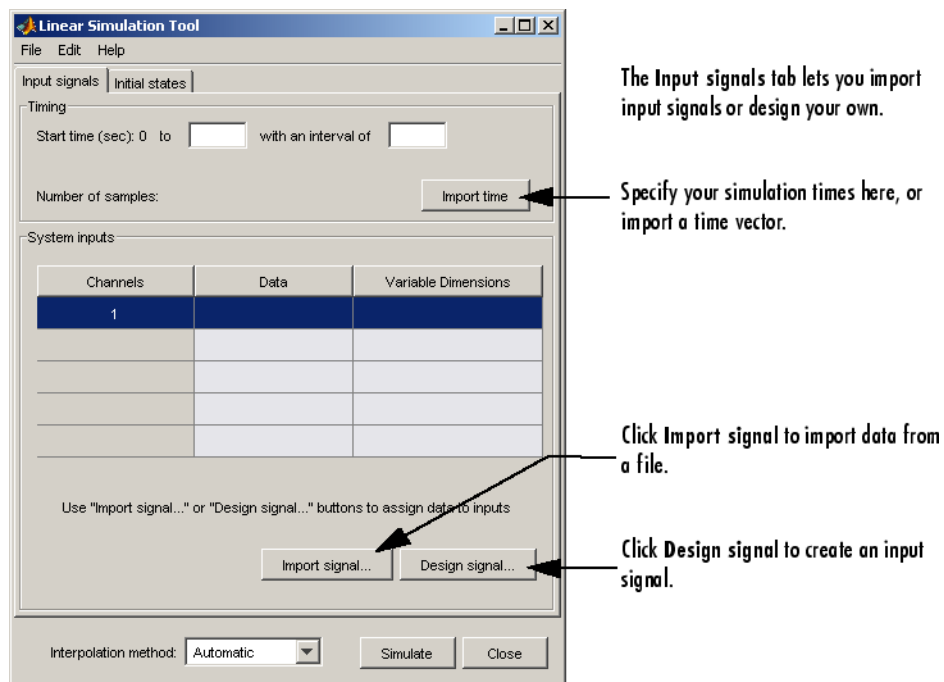- Use the lsim function at the MATLAB prompt:

```
lsim(modelname)
```

- In the MATLAB Figure window, right-click a response plot and select **Input data**.

Working with the Linear Simulation Tool

The Linear Simulation Tool contains two tabs, **Input signals** and **Initial states**.

After opening the Linear Simulation Tool (as described in "Opening the Linear Simulation Tool" on page 3-21), follow these steps to simulate your model:

1 Click the **Input signals** tab, if it is not displayed.

The **Input signals** tab lets you import input signals or design your own.

Specify your simulation times here, or import a time vector.

Click **Import signal** to import data from a file.
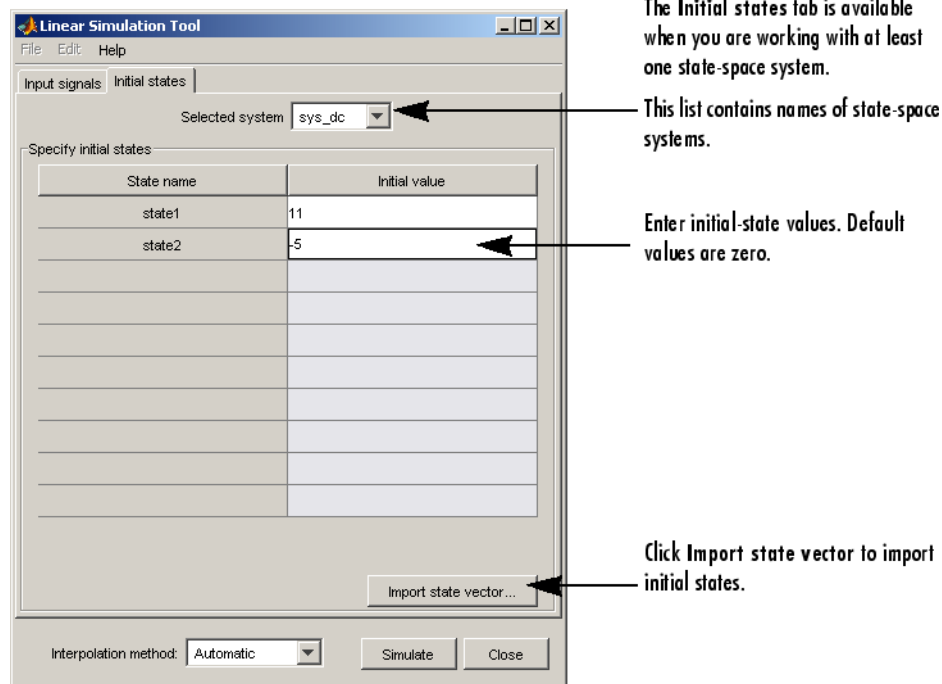
Click **Design signal** to create an input signal.

2 In the **Timing** area, specify the simulation time vector by doing one of the following:

- Import the time vector by clicking **Import time**.
- Enter the end time and the time interval in seconds. The start time is set to 0 seconds.

3 Specify the input signal by doing one of the following:

- Click **Import signal** to import it from the MATLAB workspace or a file. For more information, see "Importing Input Signals" on page 3-24.

- Click **Design signal** to create your own inputs. For more information, see "Designing Input Signals" on page 3-31.

4 If you have a state-space model and want to specify initial conditions, click the **Initial states** tab. By default, all initial states are set to zero.

You can either enter state values in the **Initial value** column, or import values by clicking **Import state vector**. For more information about entering initial states, see "Specifying Initial Conditions" on page 3-33.

The **Initial states** tab is available when you are working with at least one state-space system.

This list contains names of state-space systems.

Enter initial-state values. Default values are zero.

Click **Import state vector** to import initial states.

5 For a continuous model, select one of the following interpolation methods in the **Interpolation method** list to be used by the simulation solver:

- Zero order hold

- First order hold (linear interpolation)
- Automatic (Linear Simulation Tool selects first order hold or zero order hold automatically, based on the smoothness of the input)

Note The interpolation method is not used when simulating discrete models.

---

6 Click **Simulate**.

Importing Input Signals

You can import input signals from the MATLAB workspace after opening the Linear Simulation Tool (see "Opening the Linear Simulation Tool" on page 3-21). You can also import inputs from a MAT-file, Microsoft Excel spreadsheet, ASCII flat-file, comma-separated variable file (CSV), or text file.

For information about creating your own inputs, see "Designing Input Signals" on page 3-31. For an overview of working with the Linear Simulation Tool, see "Working with the Linear Simulation Tool" on page 3-22.

To import one or more input signals:

1 In the Linear Simulation Tool, click the **Input signals** tab, if it is not displayed.

2 Specify the simulation time in the **Timing** area.

3 Select one or more rows for the input channels you want to import. The following figure shows an example with two selected channels.
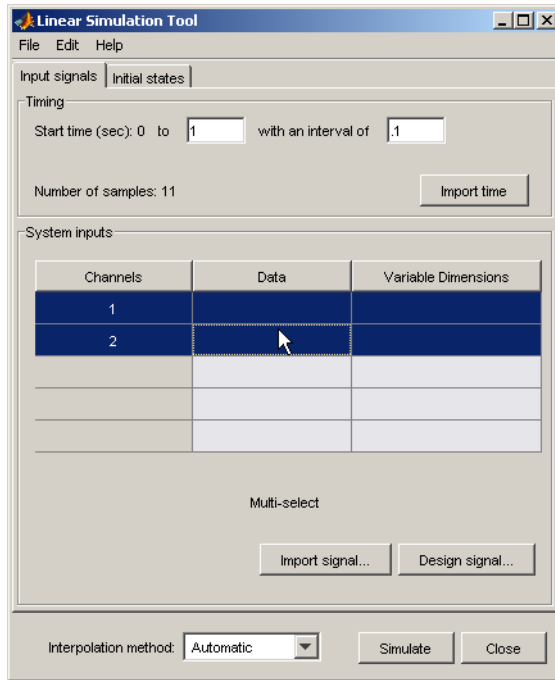
4 Click **Import signal** to open the Data Import dialog box. The following figure shows an example of the Data Import dialog box.
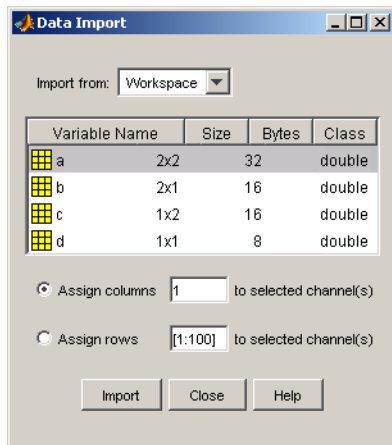
5 In the **Import from** list, select the source of the input signals. It can be one of the following:

- `Workspace`
- `MAT file`
- `XLS file`
- `CSV file`
- `ASCII file`

6 Select the data you want to import. The Data Import dialog box contains different options depending on which source format you selected.

7 Click **Import**.

For an example of importing input signals, see the following:

- "Example: Loading Inputs from a Microsoft Excel Spreadsheet" on page 3-26
- "Example: Importing Inputs from the Workspace" on page 3-27

Example: Loading Inputs from a Microsoft Excel Spreadsheet

To load inputs from a Microsoft Excel (XLS) spreadsheet:

1 In the Linear Simulation Tool, click **Import signal** in the **Input signals** tab to open the Data Import dialog box.

2 Select **XLS file** in the **Import from** list.

3 Click **Browse**.

4 Select the file you want to import and click **Open**. This populates the Data Import dialog box with the data from the Microsoft Excel spreadsheet.

If your spreadsheet has multiple worksheets, select one from this list.

Click a column to select it. Press **Shift** to select multiple contiguous columns; press **Ctrl** to select nonadjacent columns.

Enter the number of the first row that contains data.

Bad data contains strings or missing values. Select to skip cells or rows, linearly interpolate between good data values, or use zero-order hold.
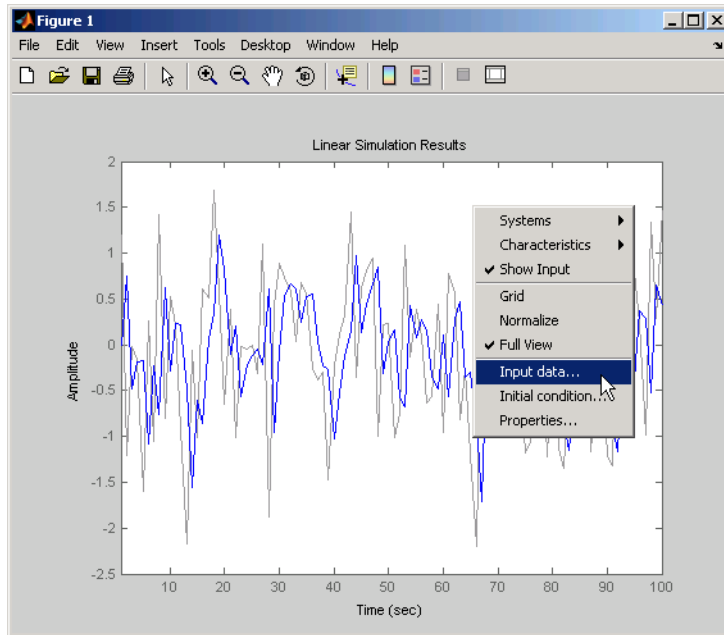
Example: Importing Inputs from the Workspace

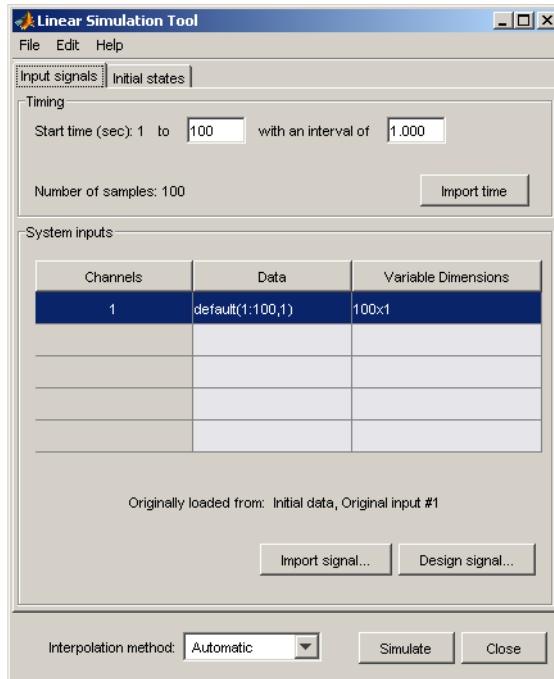To load an input signal from the MATLAB workspace:

1 Enter this code to open a response plot with a second-order system:

```
s=tf('s');
ss=(s+2)/(s^2+3*s+2);
lsim(ss,randn(100,1),1:100);
```

2 Right-click the plot background and select **Input data**.

This opens the Linear Simulation Tool with default input data.

3 Create an input signal for your system in the MATLAB Command Window, such as the following:

```
new_signal=[-3*ones(1,20) 2*ones(1,30) 0.5*ones(1,50)]';
```

4 In the Linear Simulation Tool, click **Import signal**.

5 In the Data Import dialog box, click, **Assign columns** to assign the first column of the input signal to the selected channel.
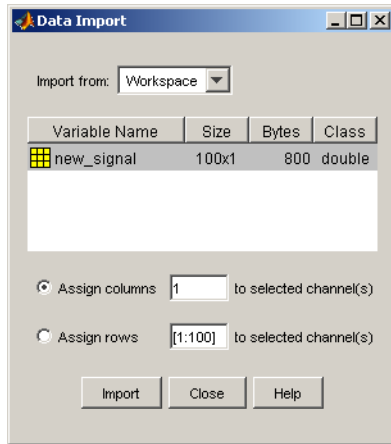
6 Click **Import**. This imports the new signal into the Linear Simulation Tool.
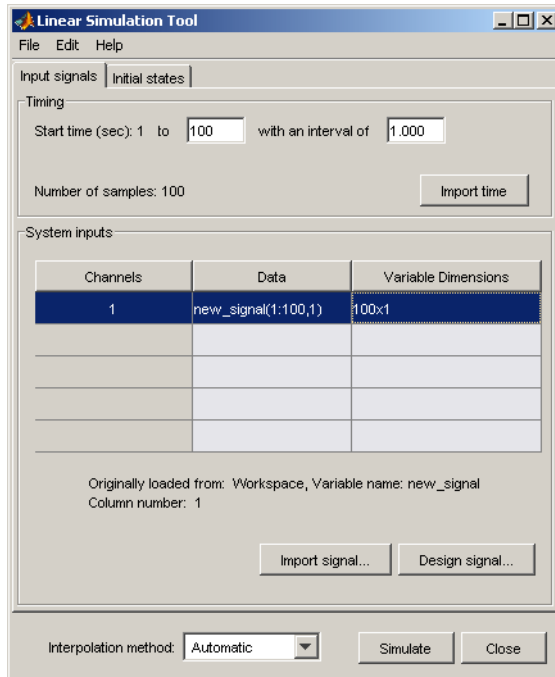
7 Click **Simulate** to see the response of your second-order system to the imported signal.
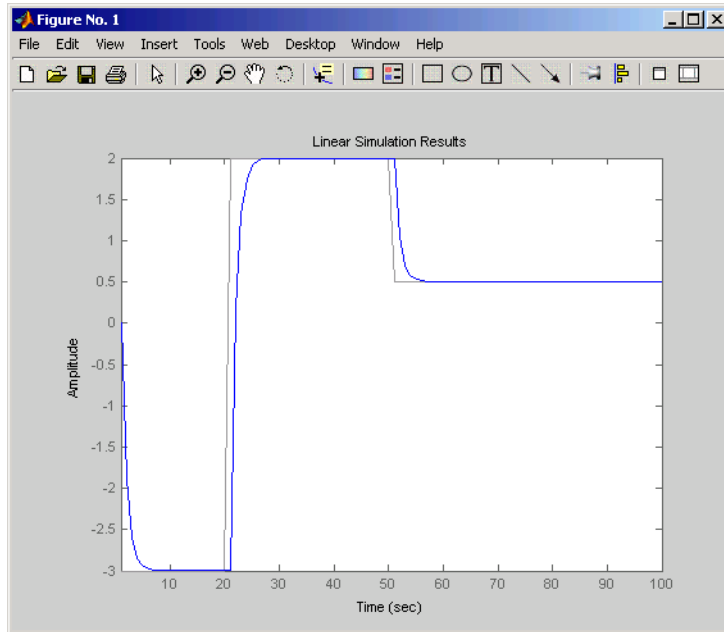

Designing Input Signals

You can generate arbitrary input signals in the form of a sine wave, square wave, step function, or white noise after opening the Linear Simulation Tool (see "Opening the Linear Simulation Tool" on page 3-21).

For information about importing inputs from the MATLAB workspace or from a file, see "Importing Input Signals" on page 3-24. For an overview of working with the Linear Simulation Tool, see "Working with the Linear Simulation Tool" on page 3-22.

To design one or more input signals:

1 In the Linear Simulation Tool, click the **Input signals** tab (if it is not displayed).

2 Specify the simulation time in the **Timing** area. The time interval (in seconds) is used to evaluate the input signal you design in later steps of this procedure.

3 Select one or more rows for the signal channels you want to design. The following figure shows an example with two selected channels.
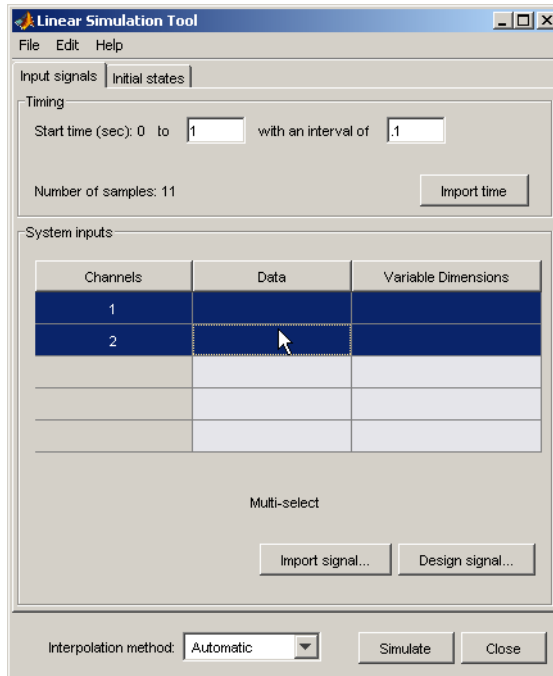
4 Click **Design signal** to open the Signal Designer dialog box. The following figure shows an example of the Signal Designer dialog box.
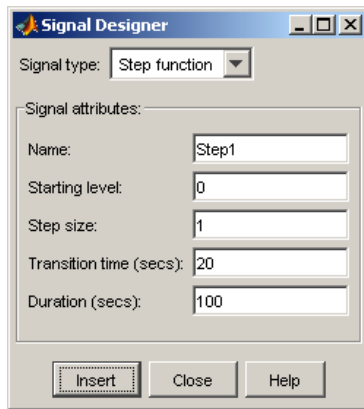
5 In the **Signal type** list, select the type of signal you want to create. It can be one of the following:

- `Sine wave`
- `Square wave`
- `Step function`
- `White noise`

6 Specify the signal characteristics. The Signal Designer dialog box contains different options depending on which signal type you selected.

7 Click **Insert**. This brings the new signal into the Linear Simulation Tool.

8 Click **Simulate** in the Linear Simulation Tool to view the system response.

Specifying Initial Conditions

If your system is in state-space form, you can enter or import initial states after opening the Linear Simulation Tool (see "Opening the Linear Simulation Tool" on page 3-21).

For an overview of working with the Linear Simulation Tool, see "Working with the Linear Simulation Tool" on page 3-22.

You can also import initial states from the MATLAB workspace.

To import one or more initial states:

1. In the Linear Simulation Tool, click the **Initial states** tab (if it is not already displayed).

2. In the **Selected system** list, select the system for which you want to specify initial conditions.

3. You can either enter state values in the **Initial value** column, or import values from the MATLAB workspace by clicking **Import state vector**. The following figure shows an example of the import window:
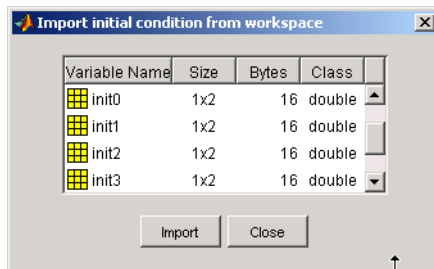
Note For *n*-states, your initial-condition vector must have *n* entries.

4. After specifying the initial states, click **Simulate** in the Linear Simulation Tool to view the system response.

Functions for Time and Frequency Response

In this section...

When to Use Functions for Time and Frequency Response

You can use the LTI Viewer GUI for a wide range of applications. There are situations, however, where you may want a more open and extensible environment. You can use the Control System Toolbox functions for basic time and frequency domain analysis plots used in control system engineering. These functions apply to any kind of linear model (continuous or discrete, SISO or MIMO, or arrays of models). You can only apply the frequency domain analysis functions to FRD models.

Use the LTI Viewer when a GUI-driven environment is desirable. On the other hand, use functions when you want customized plots. If you want to include data unrelated to your models, you must use functions instead of the LTI Viewer (which only plots model data).

The next sections discuss time and frequency response functions and how to use these functions to create customized plots of linear model responses.

Time and Frequency Response Functions

Time responses investigate the time-domain transient behavior of linear models for particular classes of inputs and disturbances. You can determine such system characteristics as rise time, settling time, overshoot, and steady-state error from the time response. You can use the Control System Toolbox functions for step response, impulse response, initial condition response, and general linear simulations. For example, you can simulate the response to white noise inputs using `lsim` and the MATLAB function `randn`.

In addition to time-domain analysis, you can use the Control System Toolbox functions for frequency-domain analysis using the following standard plots:

- Bode
- Nichols
- Nyquist
- Singular value

This table lists available time and frequency response functions and their use.
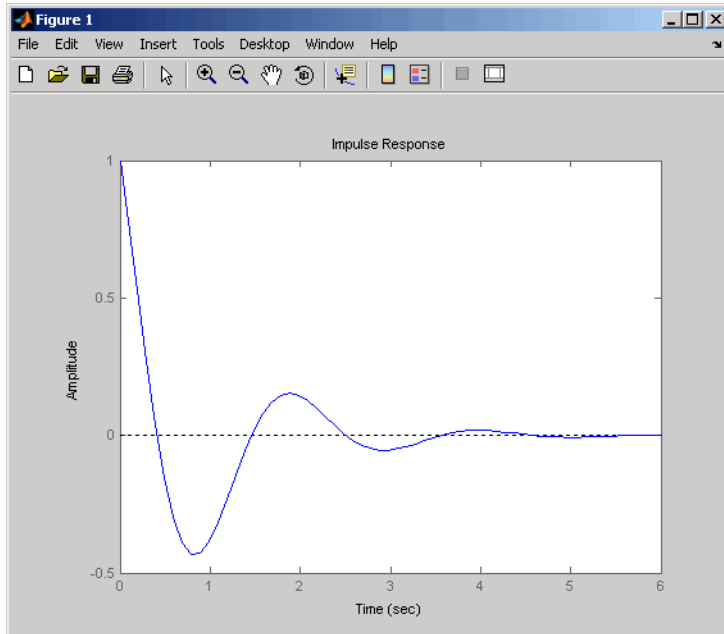
Functions for Frequency and Time Response

Functions	Description
bode	Bode plot
evalfr	Computes the frequency response at a single complex frequency (not for FRD models)
freqresp	Computes the frequency response for a set of frequencies
gensig	Input signal generator (for lsim)
impulse	Impulse response plot
initial	Initial condition response plot
iopzmap	Pole-zero map for each I/O pair of an LTI model
lsim	Simulation of response to arbitrary inputs
margin	Computes and plots gain and phase margins
nichols	Nichols plot
nyquist	Nyquist plot
pzmap	Pole-zero map
step	Step response plot
hsvd	Compute Hankel singular values of LTI model
bodemag	Bode magnitude response of LTI models

These functions can be applied to single linear models or LTI arrays.

The functions step, impulse, and initial automatically generate an appropriate simulation horizon for the time response plots. For example,

```
h = tf([1 0],[1 2 10])
impulse(h)
```
produces the following plot.

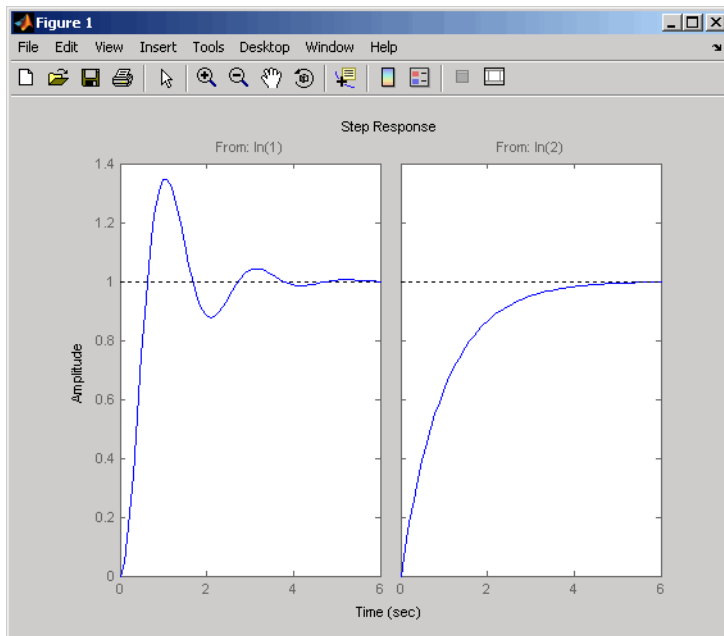**Impulse Response of a SISO Model**

Frequency-domain plots automatically generate an appropriate frequency range as well.

Plotting MIMO Model Responses

For MIMO models, time and frequency response functions produce an array of plots with one plot per I/O channel (or per output for `initial` and `lsim`). For example,

```
h = [tf(10,[1 2 10]) , tf(1,[1 1])]
step(h)
```

produces the following plot.

Step Responses for a MIMO Model

The simulation horizon is automatically determined based on the model dynamics. You can override this automatic mode by specifying a final time,

```
step(h,10) % Simulates from 0 to 10 seconds
```

or a vector of evenly spaced time samples.

```
t = 0:0.01:10    % Time samples spaced every 0.01 second
step(h,t)
```

Right-Click Menus

All the time and frequency response functions provide right-click menus that allow you to customize your plots. For more information on using the LTI Viewer right-click menus, see "Using the Right-Click Menu in the LTI Viewer". This figure shows the plots from Step Responses for a MIMO Model, with the right-click menu open.
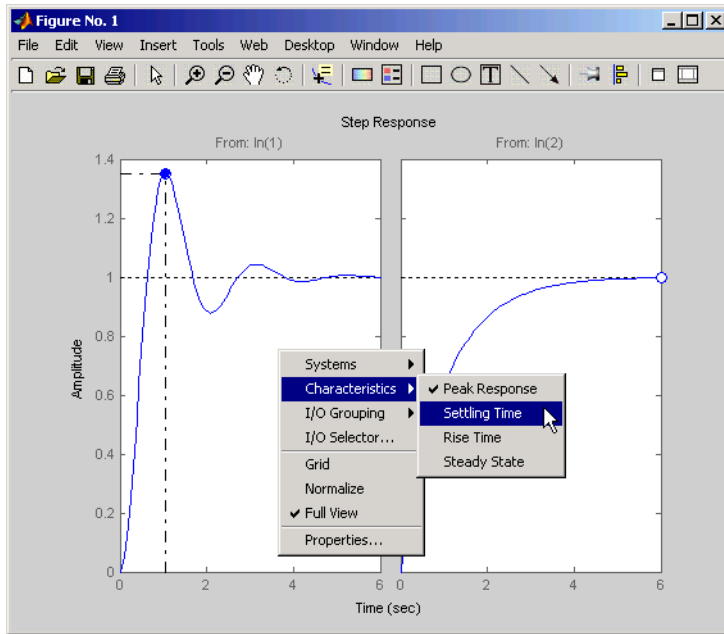
Using the Right-Click Menu in a Step Response Plot

The options you can select include

- **Systems** — Select or clear any models that you included when you created the response plot.
- **Characteristics** — Add information about the plot. The characteristics available change from plot to plot. For example, Bode plots have stability margins available, but step responses have rise time and steady-state values available.
- **Axes Grouping** — Change the grouping of your plots. Available options are All, None, Inputs, and Outputs. You can group all the plots together, place each in a separate plot region (none), or group the inputs or outputs together.
- **I/O Selector** — Open the **I/O Selector** dialog box.
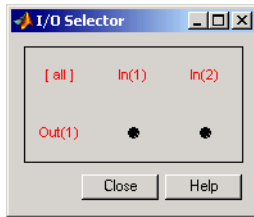
Use this dialog box to select/clear which inputs and outputs to plot.

- **Normalize** — Scale responses to fit the view (only available for time-domain plot types).
- **Full View** — Use automatic limits to make the entire curve visible.
- **Grid** — Add grids to your plots.
- **Properties** — Open the Property Editor, which you can use to customize various attributes of your plot. See "Customization" for a full description of the Property Editor.

  Alternatively, you can open the Property Editor by double-clicking in an empty region of the response plot.

Data Markers

In addition to right-click menus, you can use plot data markers. These allow you to identify key data points on your plots. This figure, using the same plot as Step Responses for a MIMO Model, shows markers on the plots.
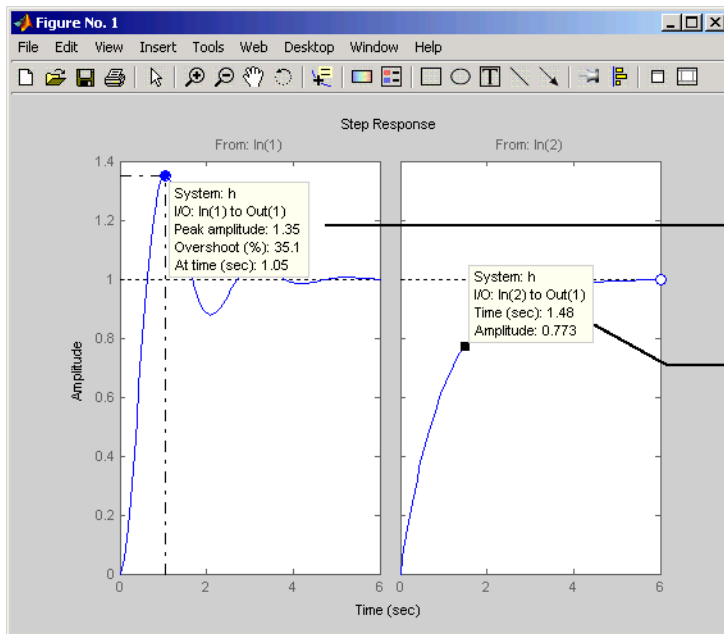
Place your cursor on an active characteristic, shown as a blue dot here, to see its name and numerical value.

Left-click anywhere on a plot to see the system name, I/O labels, time, and value at that point.

Using Plot Markers to Identify Data Points

You can move a data marker by

- Grabbing the black square located at the corner of the marker
- Dragging the marker with your mouse

The time and amplitude values will change as you move the marker. This does not apply to markers that display plot characteristics (e.g., peak value or rise time). In the case of plot characteristic data markers, you can view them by placing your cursor over the dot that represents the active characteristic. To make the data marker persistent, left-click the marker.

Note Data markers do not apply to the SISO Design Tool, which displays data about plot characteristics in the status pane at the bottom of the SISO Design Tool window.
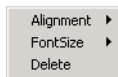
Right-Click Menus

Right-click on any data marker to open a property menu for the marker.

Property options for the marker include

- **Alignment** — Change the position of the marker. Available options are top-right, top-left, bottom-right, and bottom-left.
- **FontSize** — Change the font size.
- **Movable** — By default, you can move data markers by clicking and dragging. Clearing **Movable** forces the marker to remain at a fixed data point.
- **Delete** — Remove the selected marker. Alternatively, left-click anywhere in the empty plot region to delete all markers in the plot
- **Interpolation** — By default, data markers linearly interpolate between points along the plotted curve. Select **None** to force the markers to snap to nearest points along the plotted curve.

Since characteristic data markers are by definition fixed, the right-click menus for them have fewer options.

These options work the same as they do for the full right-click menu.

Plotting and Comparing Multiple Systems

You can use the command-line response-plotting functions to plot the response of continuous and discrete linear models on a single plot. To do so, invoke the corresponding command-line function using the list sys1,..., sysN of models as the inputs.

```
stepplot(sys1,sys2,...,sysN)
impulseplot(sys1,sys2,...,sysN)
...
bodeplot(sys1,sys2,...,sysN)
nicholsplot(sys1,sys2,...,sysN)
...
```

All models in the argument lists of any of the response plotting functions (except for `sigma`) must have the same number of inputs and outputs. To differentiate the plots easily, you can also specify a distinctive color/linestyle/marker for each system just as you would with the `plot` command. For example,

```
bodeplot(sys1,'r',sys2,'y--',sys3,'gx')
```

plots `sys1` with solid red lines, `sys2` with yellow dashed lines, and `sys3` with green `x` markers.

You can plot responses of multiple models on the same plot. These models do not need to be all continuous-time or all discrete-time.
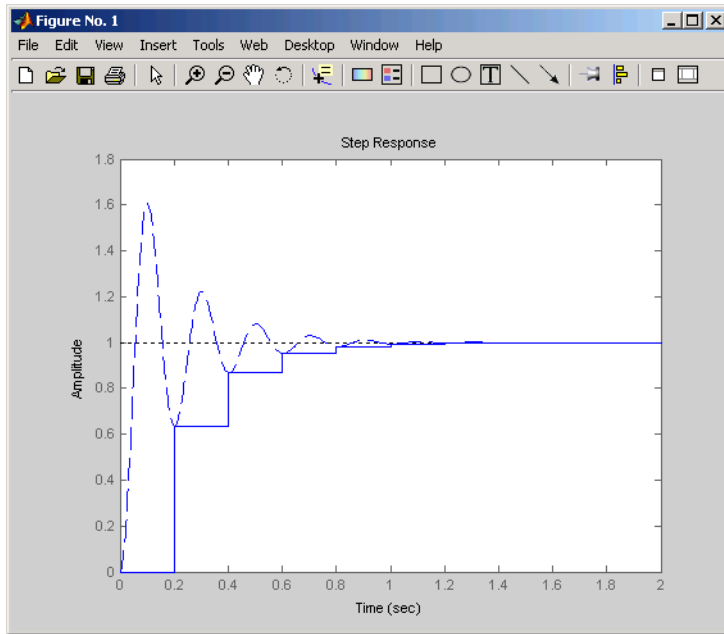
Example: Comparing Continuous and Discretized Systems

The following example compares a continuous model with its zero-order-hold discretization.

```
sysc = tf(1000,[1 10 1000])
sysd = c2d(sysc,0.2)
% ZOH sampled at 0.2 second

stepplot(sysc,'--',sysd,'-')
% Compare step responses
```

These commands produce the plot shown below.
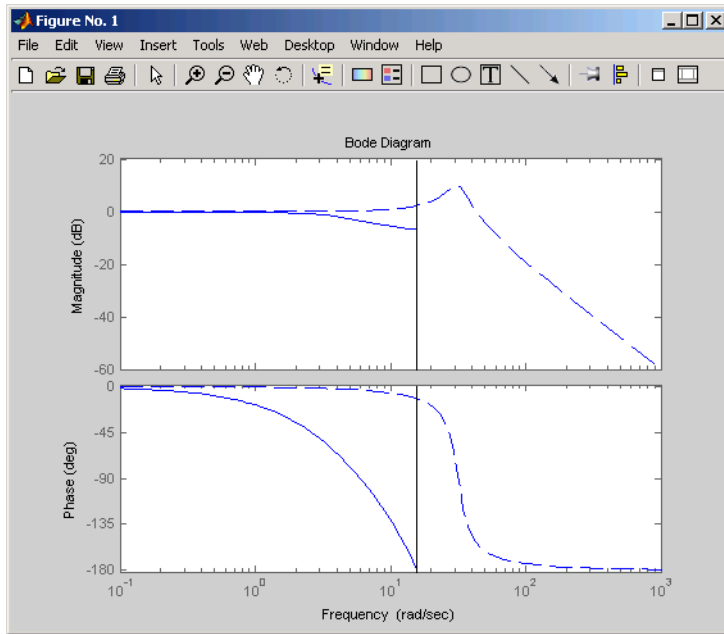
Comparison of a Continuous Model to Its Discretized Version

Use this command to compare the Bode plots of the two systems.

```
bodeplot(sysc,'--',sysd,'-')
% Compare Bode responses
```

The following plot results from the last command.

Comparison of Bode Plots for a Continuous Model and Its Discretized Version

A comparison of the continuous and discretized responses reveals a drastic undersampling of the continuous-time system. Specifically, there are hidden oscillations in the discretized time response and aliasing conceals the continuous-time resonance near 30 rad/sec.

Creating Custom Plots

Time and frequency response commands are useful for creating custom plots. You can mix model response plots with other data views using response commands together with MATLAB plotting commands such as `plot`, `subplot`, `hold`.

Example: Custom Plots

For example, the following sequence of commands displays the Bode plot, step response, pole/zero map, and some additional data in a single figure window.
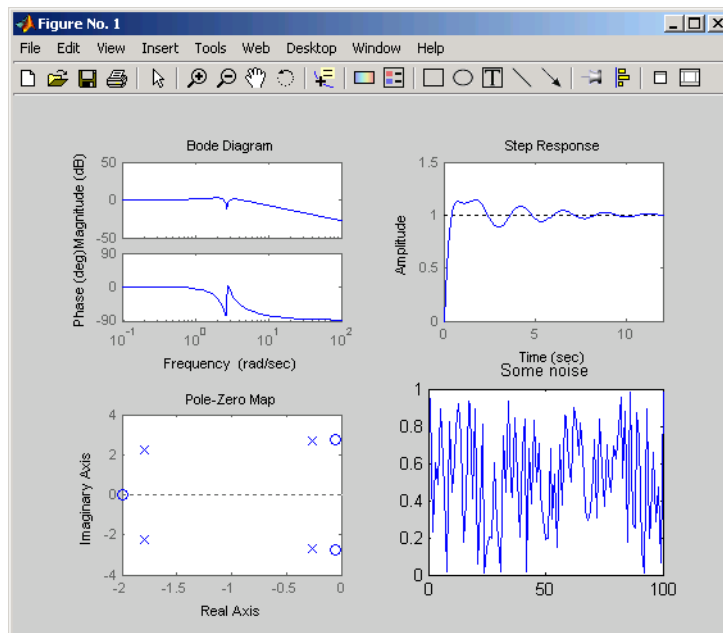
```
h = tf([4 8.4 30.8 60],[1 4.12 17.4 30.8 60]);
subplot(221)
```

```
bodeplot(h)
subplot(222)
stepplot(h)
subplot(223)
pzplot(h)
subplot(224)
plot(rand(1, 100))      % Any data can go here
title('Some noise')
```

Your plot should look similar to this illustration.

Example of Creating a Custom Plot
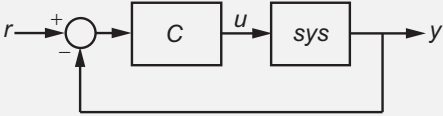
Note Each of the plots generated by response analysis functions in Example of Creating a Custom Plot (`bodeplot`, `stepplot`, and `pzplot`) has its own right-click menu (similar to those in the LTI Viewer). To activate the right-click menus, place your mouse in the plot region and right-click. The menu contents depend on what type of plot you have selected.

4

Designing Compensators

Choosing a PID Controller Design Tool

Control System Toolbox software gives you several tools for designing PID controllers.

Tool	When to use
PID Tuner	• Automatic, interactive tuning of SISO PID controller in the feed-forward path of single-loop, unity-feedback control configuration. • Interactive fitting of a plant model from measured SISO response data and automatic tuning of PID controller for the resulting model (requires System Identification Toolbox™ software).
"SISO Design Tool"	Tuning PID controllers in any other loop configuration.
Command-line PID tuning	Programmatic tuning of SISO PID controllers.

Designing PID Controllers with the PID Tuner

In this section...
"PID Tuner Overview" on page 4-3
"PID Controller Type" on page 4-4
"PID Controller Form" on page 4-5

PID Tuner Overview

Use the PID Tuner to interactively design a SISO PID controller in the feed-forward path of single-loop, unity-feedback control configuration.
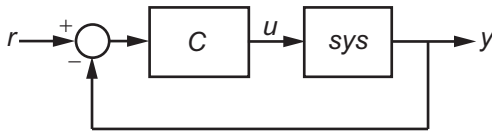
The PID Tuner automatically designs a controller for your plant. You specify the controller type (P, I, PI, PD, PDF, PID, PIDF) and form (parallel or standard). You can analyze the design using a variety of response plots, and interactively adjust the design to meet your performance requirements.

To launch the PID Tuner, use the `pidTuner` command:

```
pidTuner(sys,type)
```

where `sys` is a linear model of the plant you want to control, and `type` is a string indicating the controller type to design.

Alternatively, enter

```
pidTuner(sys,Cbase)
```

where `Cbase` is a baseline controller, allowing you to compare the performance of the designed controller to the performance of `Cbase`.

For more information about `sys` and `Cbase`, see the `pidTuner` reference page.

Note: You can also open PID Tuner from the MATLAB desktop, in the **Apps** tab. When you do so, use the **Plant** menu in PID Tuner to specify your plant model.

PID Controller Type

The PID Tuner can tune up to seven types of controllers. To select the controller type, use one of these methods:

- Provide the `type` argument to the launch command `pidTuner`.
- Provide a baseline controller `Cbase` to the launch command `pidTuner`. The PID Tuner designs a controller of the same type as `Cbase`.
- In PID Tuner, use the **Type** menu to change controller types.

`type` input to `pidTune`	Entry in Type menu	Controller Type	Continuous-time Controller Formula (parallel form)	Discrete-time Controller Formula (parallel form, `ForwardEuler` integrator formulas)
`'p'`	P	Proportional only	K_p	K_p
`'i'`	I	Integral only	$\dfrac{K_i}{s}$	$K_i \dfrac{T_s}{z-1}$
`'pi'`	PI	Proportional and integral	$K_p + \dfrac{K_i}{s}$	$K_p + K_i \dfrac{T_s}{z-1}$
`'pd'`	PD	Proportional and derivative	$K_p + K_d s$	$K_p + K_d \dfrac{z-1}{T_s}$
`'pdf'`	PDF	Proportional and derivative with first-order filter on derivative term	$K_p + \dfrac{K_d s}{T_f s + 1}$	$K_p + K_d \dfrac{1}{T_f + \dfrac{T_s}{z-1}}$
`'pid'`	PID	Proportional, integral, and derivative	$K_p + \dfrac{K_i}{s} + K_d s$	$K_p + K_i \dfrac{T_s}{z-1} + K_d \dfrac{z-1}{T_s}$
`'pidf'`	PIDF	Proportional, integral, and derivative with first-order filter on derivative term	$K_p + \dfrac{K_i}{s} + \dfrac{K_d s}{T_f s + 1}$	$K_p + K_i \dfrac{T_s}{z-1} + K_d \dfrac{1}{T_f + \dfrac{T_s}{z-1}}$

If sys is a discrete-time model with sampling time Ts, the PID Tuner designs a discrete-time `pid` controller using the `ForwardEuler` discrete integrator formula. To design

a controller that has different discrete integrator formulas, use one of the following methods:

- Provide a discrete-time baseline controller `Cbase` to the launch command `pidTuner`. The PID Tuner designs a controller that has the same discrete integrator formulas as `Cbase`.
- After launching the PID Tuner, click **Options** to open the **Controller Options** dialog box. Select discrete integrator formulas from the **Integral Formula** and **Derivative Formula** menus.

For more information about discrete integrator formulas, see the `pid` and `pidstd` reference pages.

PID Controller Form

When you use the type input to launch the PID Tuner, the PID Tuner designs a controller in parallel form. To design a controller in standard form, use one of the following methods:

- Provide a standard-form baseline controller `Cbase` to the launch command `pidTuner`. The PID Tuner designs a controller of the same form as `Cbase`.
- Use the **Form** menu to change controller form after launching the PID Tuner.

For more information about parallel and standard controller forms, see the `pid` and `pidstd` reference pages.

Related Examples
- "PID Controller Design for Fast Reference Tracking" on page 4-11
- "Analyze Design in PID Tuner" on page 4-6

Analyze Design in PID Tuner

In this section...

Plot System Responses

To determine whether the compensator design meets your requirements, you can analyze the system response using the response plots. In the **PID Tuner** tab, select a response plot from the **Add Plot** menu. The **Add Plot** menu also lets you choose from several step plots (time-domain response) or Bode plots (frequency-domain response).
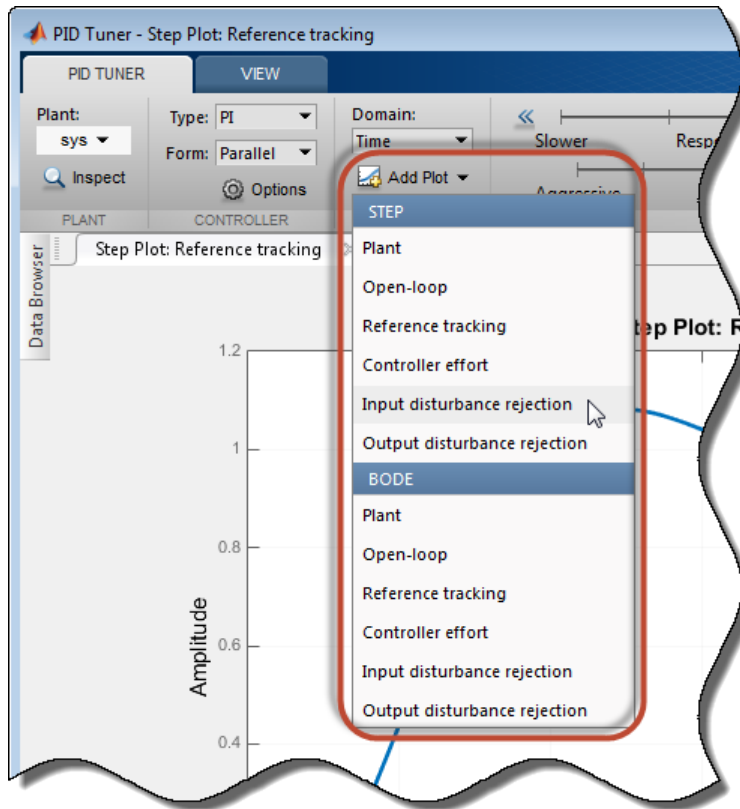
The PID Tuner computes the responses based upon the following single-loop control architecture:
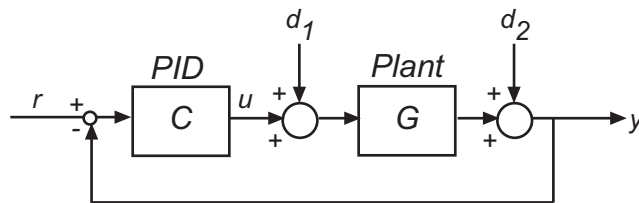
The following table summarizes the available responses.

| Response	Plotted System	Description
Plant	G	Shows the plant response. Use to examine plant dynamics.
Open-loop	CG	Shows response of the open-loop controller-plant system. Use for frequency-domain design. Use when your design specifications include robustness criteria such as open-loop gain margin and phase margin.
Reference tracking	$\dfrac{CG}{1+CG}$ (from r to y)	Shows the closed-loop system response to a step change in setpoint. Use when your design specifications include setpoint tracking.
Controller effort	$\dfrac{C}{1+CG}$ (from r to u)	Shows the closed-loop controller output response to a step change in setpoint. Use when your design is limited by practical constraints, such as controller saturation.
Input disturbance rejection	$\dfrac{G}{1+CG}$ (from d_1 to y)	Shows the closed-loop system response to load disturbance (a step disturbance at the plant input). Use when your design specifications include input disturbance rejection.
Output disturbance rejection	$\dfrac{1}{1+CG}$ (from d_2 to y)	Shows the closed-loop system response to a step disturbance at plant output. Use when you want to analyze sensitivity to measurement noise.

View Numeric Values of System Characteristics

You can view the values for system characteristics, such as peak response and gain margin, either:

- Directly on the response plot — Use the right-click menu to add characteristics, which appear as blue markers. Then, left-click the marker to display the corresponding data panel.

- In the **Performance and robustness** table — To display this table, click **Show Parameters**.
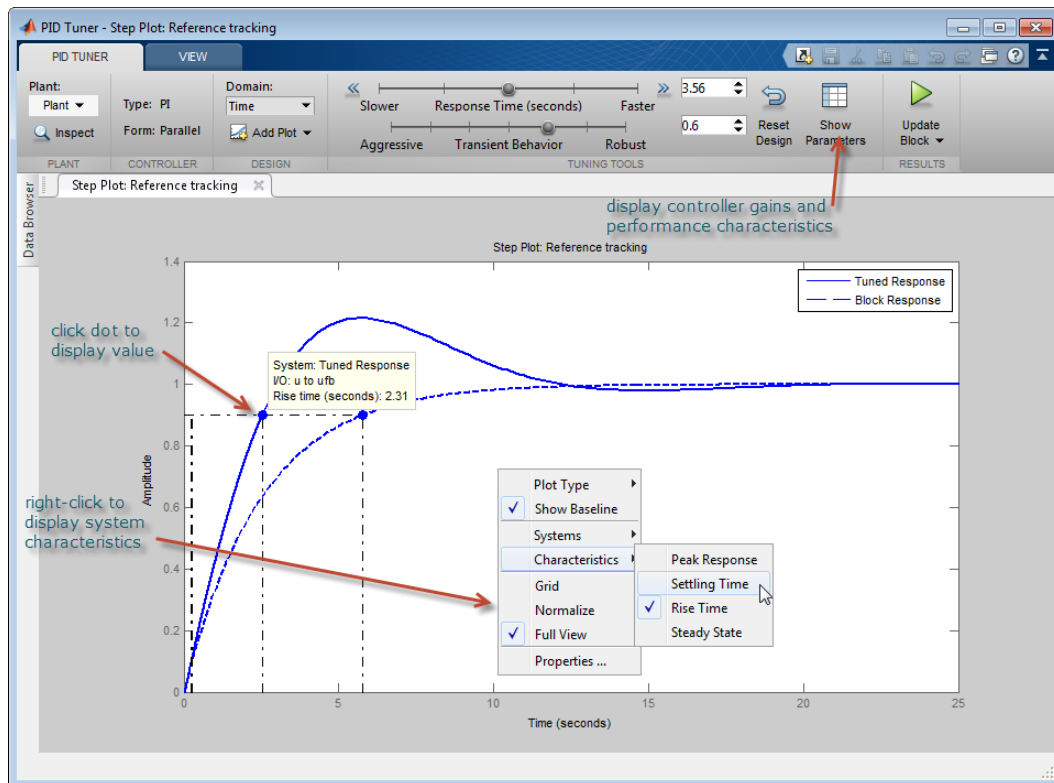
Refine the Design

If the response of the initial controller design does not meet your requirements, you can interactively adjust the design. The PID Tuner gives you two **Domain** options for refining the controller design:

- `Time` domain (default) — Use the **Response Time** slider to make the closed-loop response of the control system faster or slower. Use the **Transient Behavior** slider

to make the controller more aggressive at disturbance rejection or more robust against plant uncertainty.

- Frequency — Use the **Bandwidth** slider to make the closed-loop response of the control system faster or slower (the response time is $2/w_c$, where w_c is the bandwidth). Use the **Phase Margin** slider to make the controller more aggressive at disturbance rejection or more robust against plant uncertainty.

In both modes, there is a trade-off between reference tracking and disturbance rejection performance. For an example that shows how to use the sliders to adjust this trade-off, see "Tune PI Controller to Balance Tracking and Disturbance Rejection Performance" on page 4-33.

Tip To revert to the initial controller design after moving the sliders, click ↩ **Reset Design**.

Related Examples

- "PID Controller Design for Fast Reference Tracking" on page 4-11

PID Controller Design for Fast Reference Tracking

This example shows how to use the PID Tuner to design a controller for the plant:
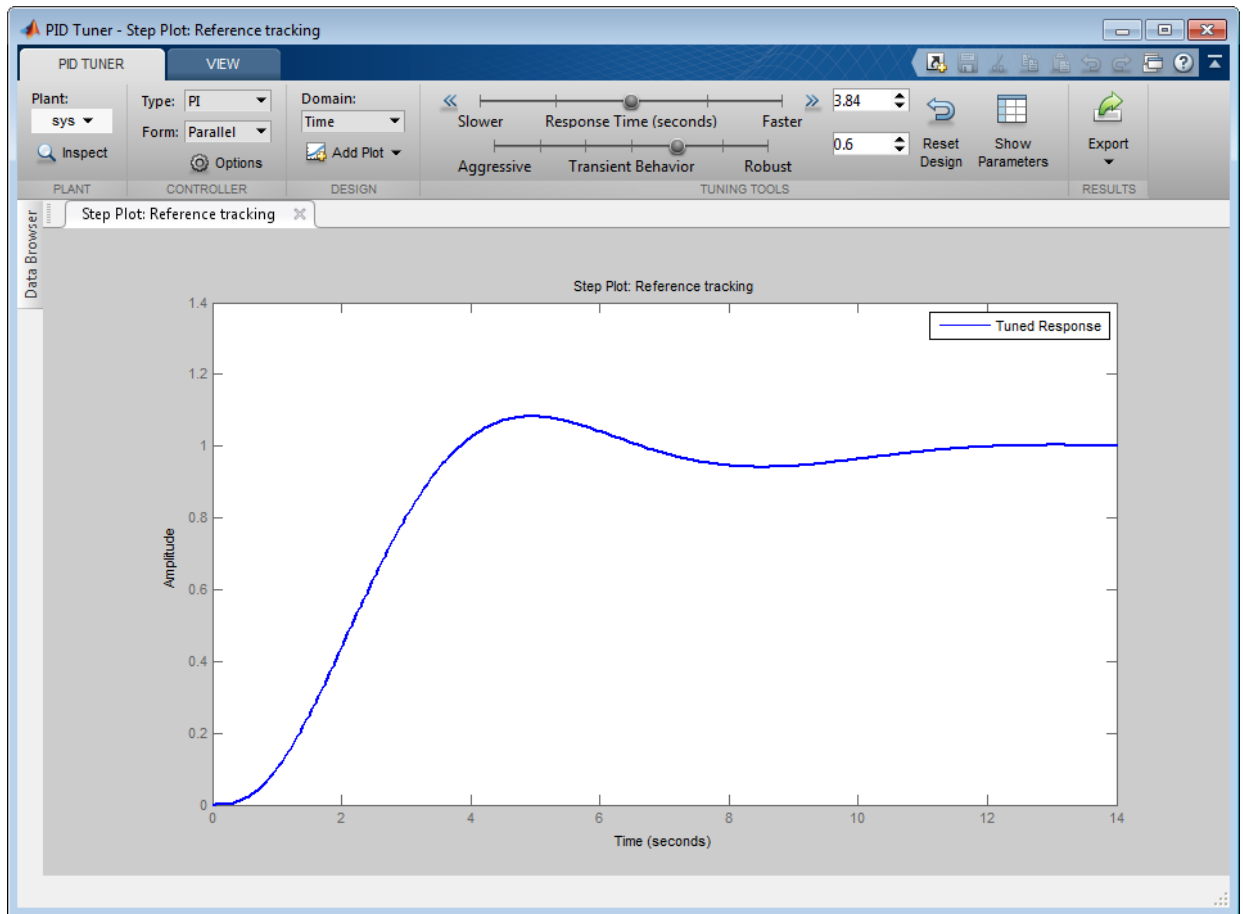
$$sys = \frac{1}{(s+1)^3}.$$

The design requirements are for the closed loop system to track a reference input with a rise time less than 1.5 s, and settling time less than 6 s.

1 Create the plant model and open the PID Tuner to design a PI controller for a first pass design.

```
sys = zpk([],[-1 -1 -1],1);
pidTuner(sys,'pi')
```

When you open the PID Tuner, it automatically designs a controller of the type you specify (here, PI). The controller is designed for a balance between performance (response time) and robustness (stability margins). The PID Tuner displays the closed-loop step response of the system with the designed controller.

Tip You can also open PID Tuner from the MATLAB desktop, in the **Apps** tab. When you do so, use the **Plant** menu in PID Tuner to specify your plant model.

2 Examine the reference tracking rise time and settling time.

Right-click on the plot and select `Characteristics > Rise Time` to mark the rise time as a blue dot on the plot. Select `Characteristics > Settling Time` to mark the settling time. To see tool-tips with numerical values, click each of the blue dots.
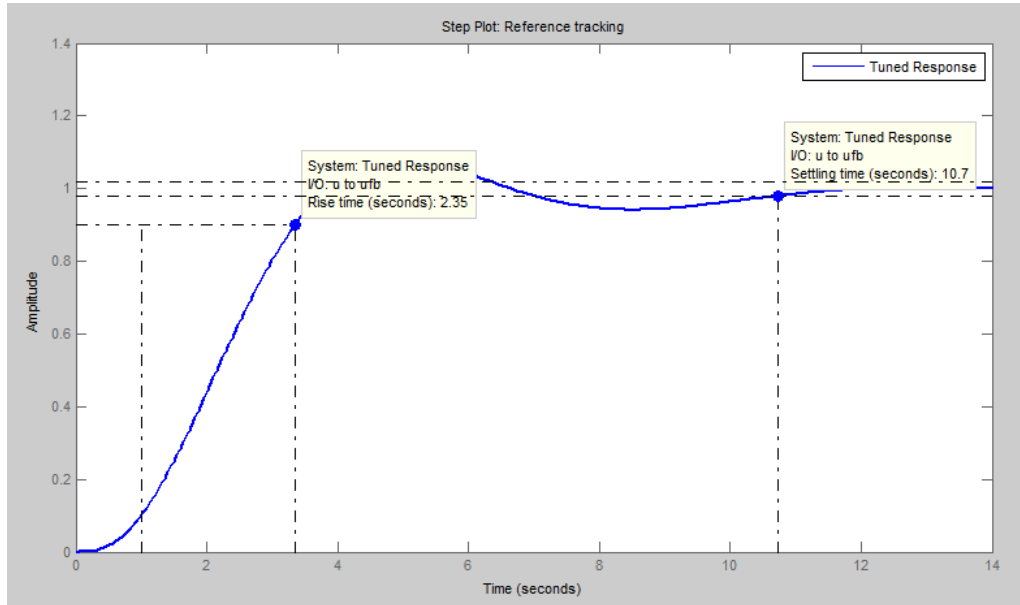
The PID Tuner's initial PI controller design provides a rise time of 2.35 s and settling time of 10.7 s. Both results are slower than the design requirements.
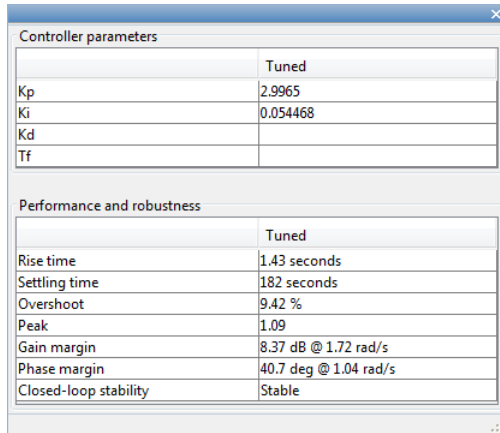
Note: To display the performance metrics in a table instead of in tool-tips on the plot, click **Show parameters**. This action opens a display containing performance and robustness metrics and the tuned controller gains.

---

3 Slide the **Response time** slider to the right to try to improve the loop performance. The response plot automatically updates with the new design.
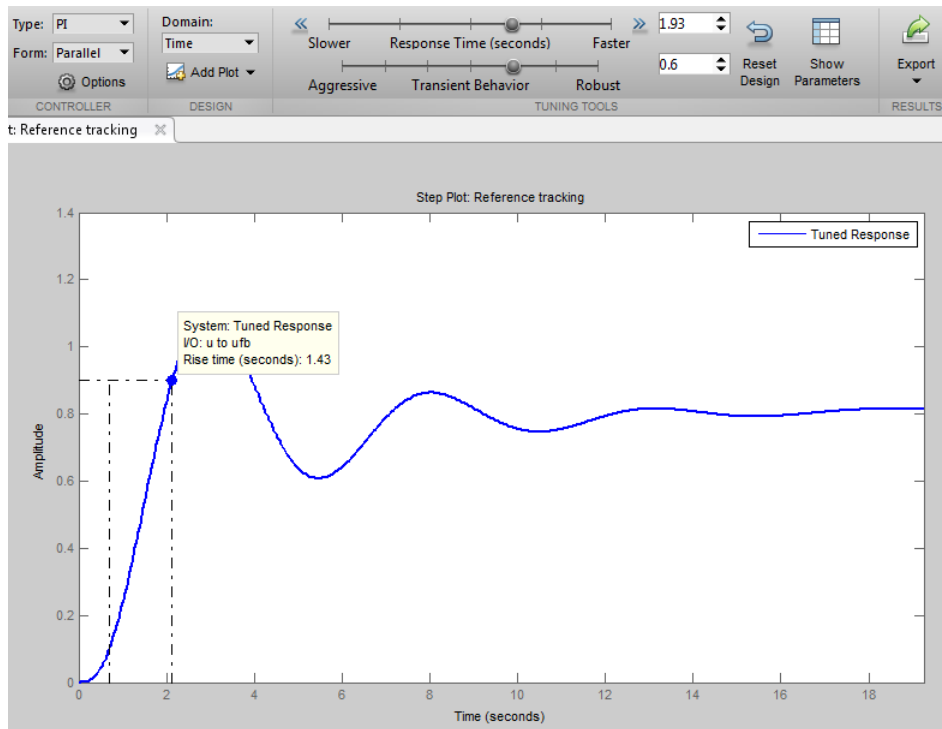
Moving the **Response time** slider far enough to meet the rise time requirement of less than 1.5 s results in more oscillation. Additionally, the parameters display shows that the new response has an unacceptably long settling time.

To achieve the faster response speed, the algorithm must sacrifice stability.

4 Change the controller type to improve the response.

Adding derivative action to the controller gives the PID Tuner more freedom to achieve adequate phase margin with the desired response speed.

In the **Type** menu, select `PIDF`. The PID Tuner designs a new PIDF controller. (See "PID Controller Type" on page 4-4 for more information about available controller types.)
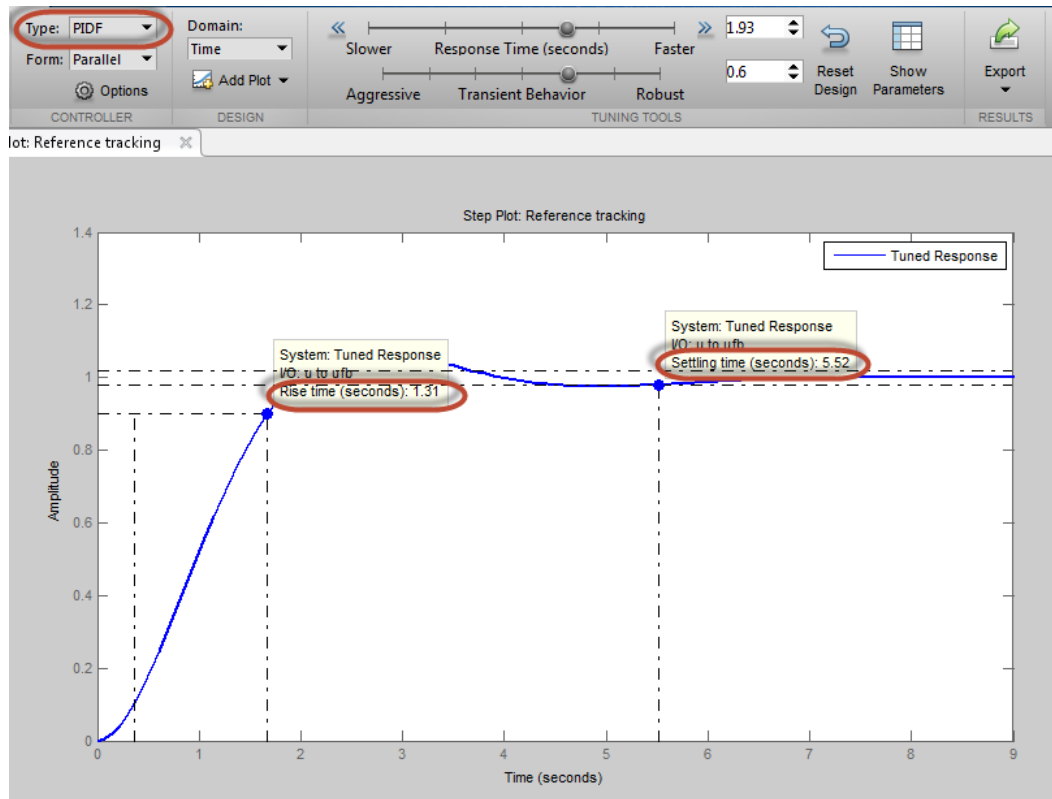
The rise time and settling time now meet the design requirements. You can use the **Response time** slider to make further adjustments to the response. To revert to the default automated tuning result, click **Reset Design**.

Note: To adjust the closed-loop bandwidth instead of the response time, select Frequency domain from the **Design mode** menu . The bandwidth is inversely proportional to the response time.

---

5 Analyze other system responses, if appropriate.

To analyze other system responses, click **Add Plot**. Select the system response you want to analyze.
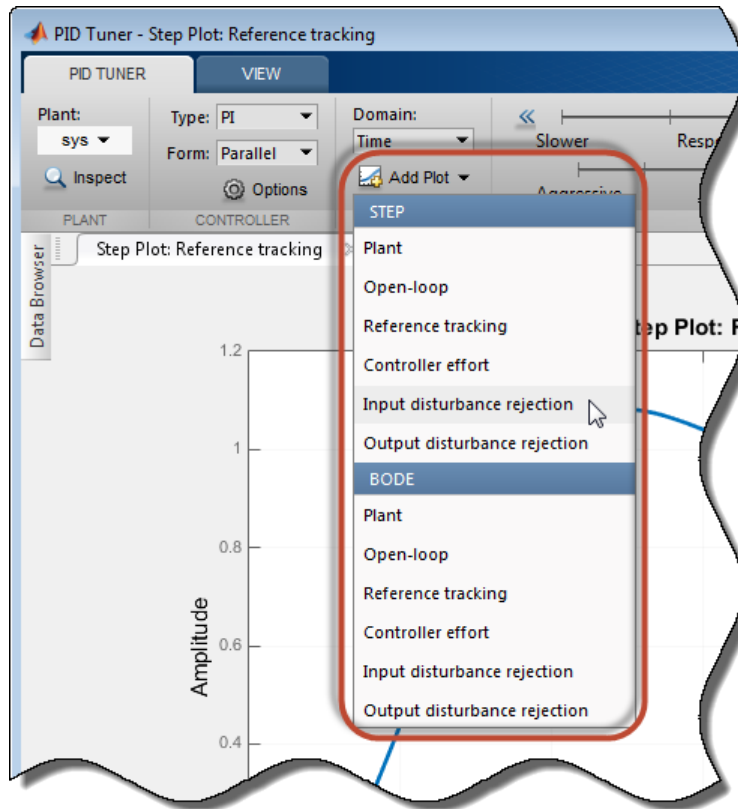
For example, to observe the closed-loop step response to disturbance at the plant input, in the **Step** section of the **Add Plot** menu, select `Input disturbance rejection`. The disturbance rejection response appears in a new figure.
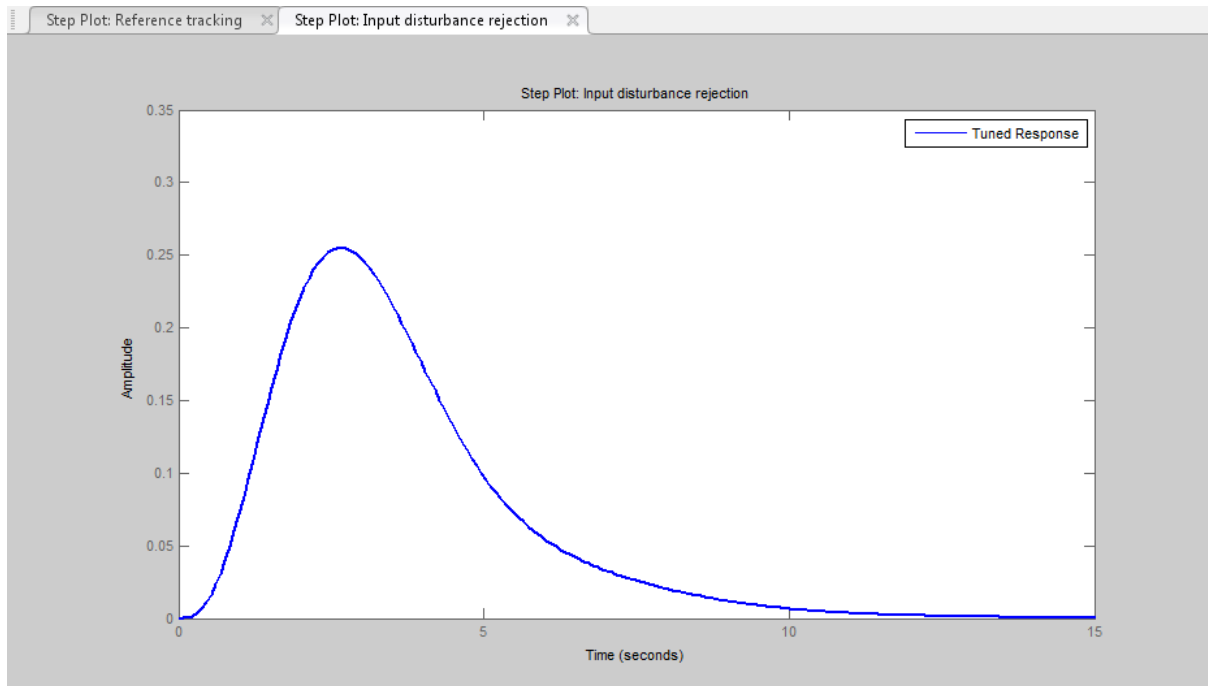
See "Analyze Design in PID Tuner" on page 4-6 for more information about available response plots.

Tip Use the options in the **View** tab to change how PID Tuner displays multiple plots.

---

6 Export your controller design to the MATLAB workspace.

To export your final controller design to the MATLAB workspace, click **Export**. The PID Tuner exports the controller as a

- `pid` controller object, if the **Form** is `Parallel`

- pidstd controller object, if the **Form** is Standard

Related Examples

- "Tune PI Controller to Balance Tracking and Disturbance Rejection Performance" on page 4-33
- "Designing PID for Disturbance Rejection with PID Tuner" on page 4-20

More About

- "Analyze Design in PID Tuner"

Designing PID for Disturbance Rejection with PID Tuner

This example shows how to design a PI controller with good disturbance rejection performance using the PID Tuner tool. The example also shows how to design an ISA-PID controller for both good disturbance rejection and good reference tracking.

Launching the PID Tuner with Initial PID Design

The plant model is

$$G(s) = \frac{6(s+5)e^{-s}}{(s+1)(s+2)(s+3)(s+4)}$$

```
G = zpk(-5,[-1 -2 -3 -4],6,'OutputDelay',1);
G.InputName = 'u';
G.OutputName = 'y';
```

Use the following command to launch the PID Tuner to design a PI controller in parallel form for plant G.

```
pidtool(G,'pi')
```

The PID Tuner automatically designs an initial PI controller. Click "Show parameters" button to display the controller gains and performance metrics.
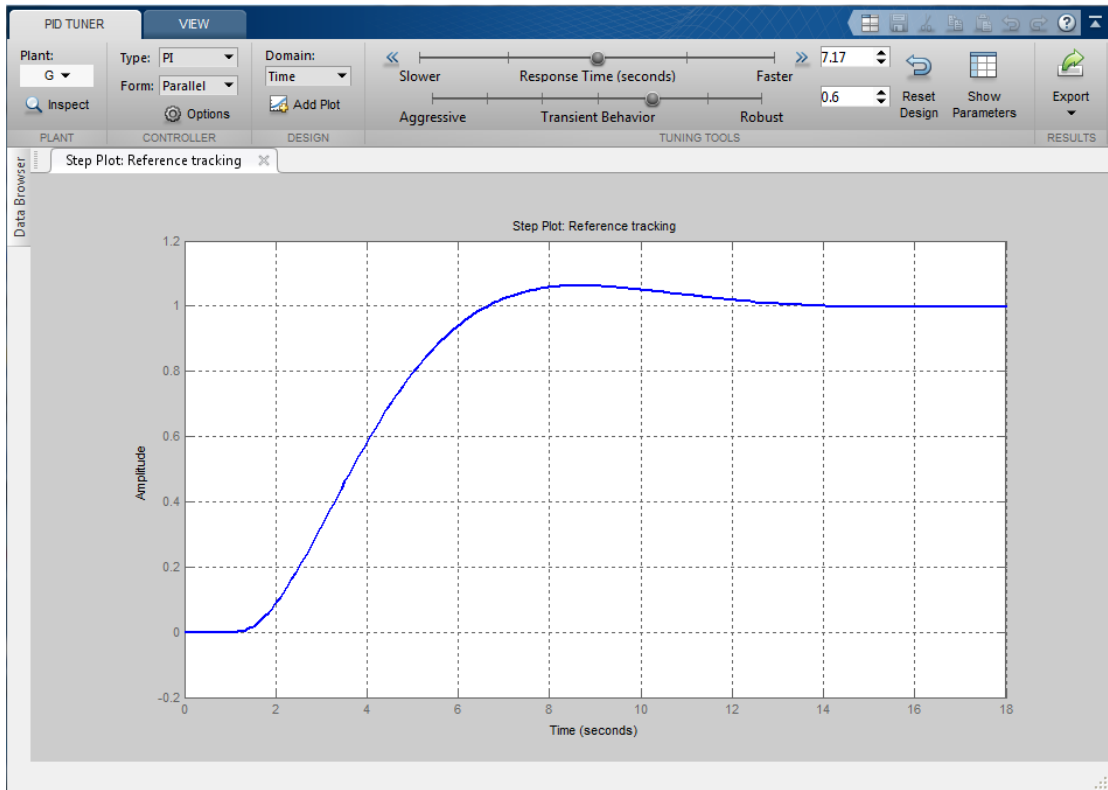
Controller parameters	
	Tuned
Kp	0.22729
Ki	0.22657
Kd	
Tf	

Performance and robustness	
	Tuned
Rise time	3.61 seconds
Settling time	11.9 seconds
Overshoot	6.27 %
Peak	1.06
Gain margin	10.7 dB @ 0.852 rad/s
Phase margin	60 deg @ 0.279 rad/s
Closed-loop stability	Stable

For step reference tracking, the settling time is about 12 seconds and the overshoot is about 6.3 percent, which is acceptable for this example.

Tuning PID for Disturbance Rejection

Assume that a step disturbance occurs at the plant input and the main purpose of the PI controller is to reject this disturbance quickly. In the rest of this section, we will show how to design the PI controller for better disturbance rejection in the PID Tuner. We also expect that the reference tracking performance is degraded as disturbance rejection performance improves.

Because the attenuation of low frequency disturbance is inversely proportional to integral gain Ki, maximizing the integral gain is a useful heuristic to obtain a PI controller with good disturbance rejection. For background, see Karl Astrom et al., "Advanced PID Control", Chapter 4 "Controller Design", 2006, The ISA Society.

Click **Add Plot**, select **Input disturbance rejection**, and click **Add** to plot the input disturbance step response. The peak deviation is about 1 and it settles to less than 0.1 in about 9 seconds.

Domain:

Time ▼

📊 Add Plot

Slower

Aggressive

Add a Response Plot ✕

Plot type

- ◉ Step
- ○ Bode

Response Type

- ○ Open-loop
- ○ Reference tracking
- ○ Controller effort
- ◉ Input disturbance rejection
- ○ Output disturbance rejection
- ○ Plant
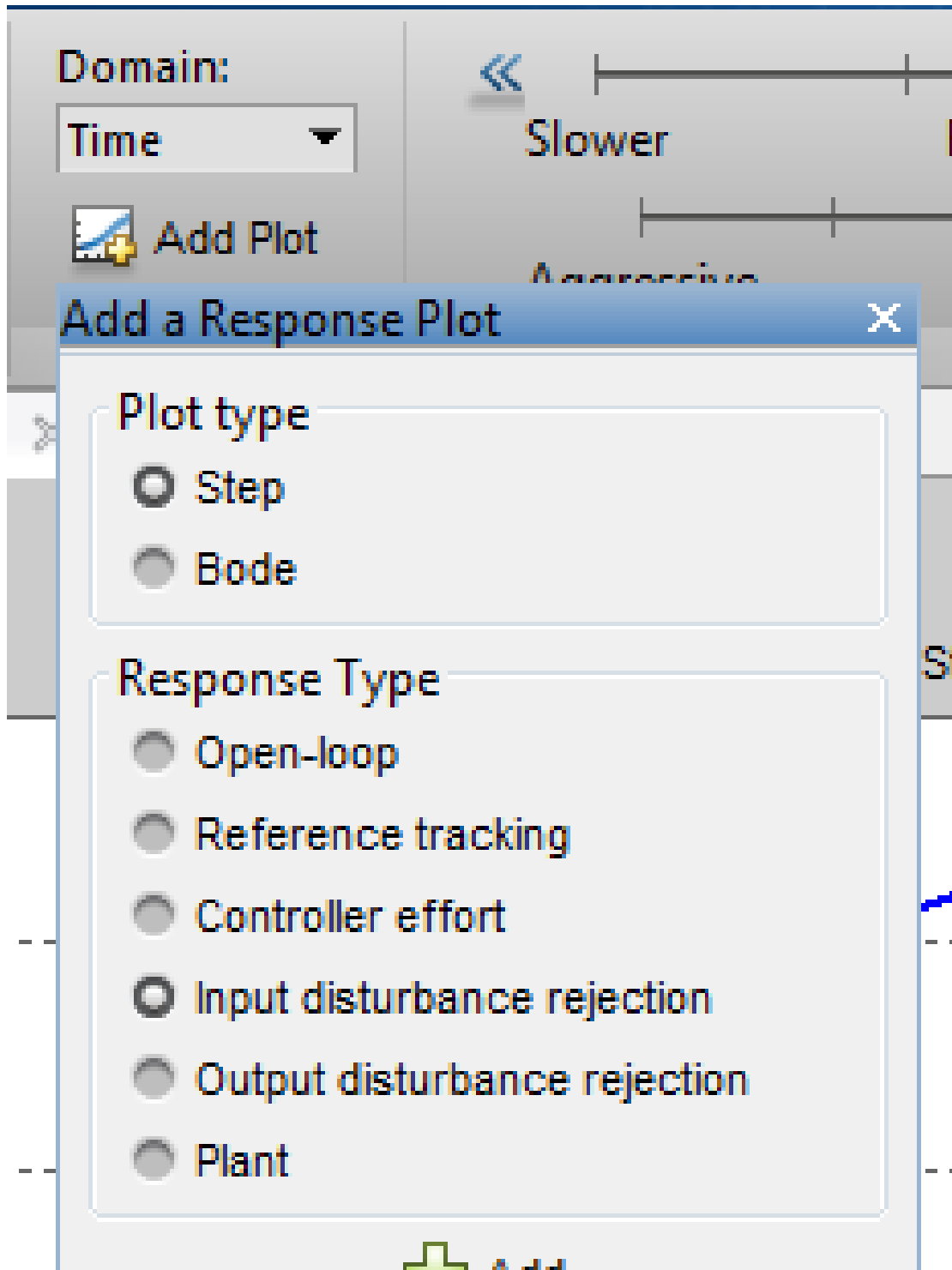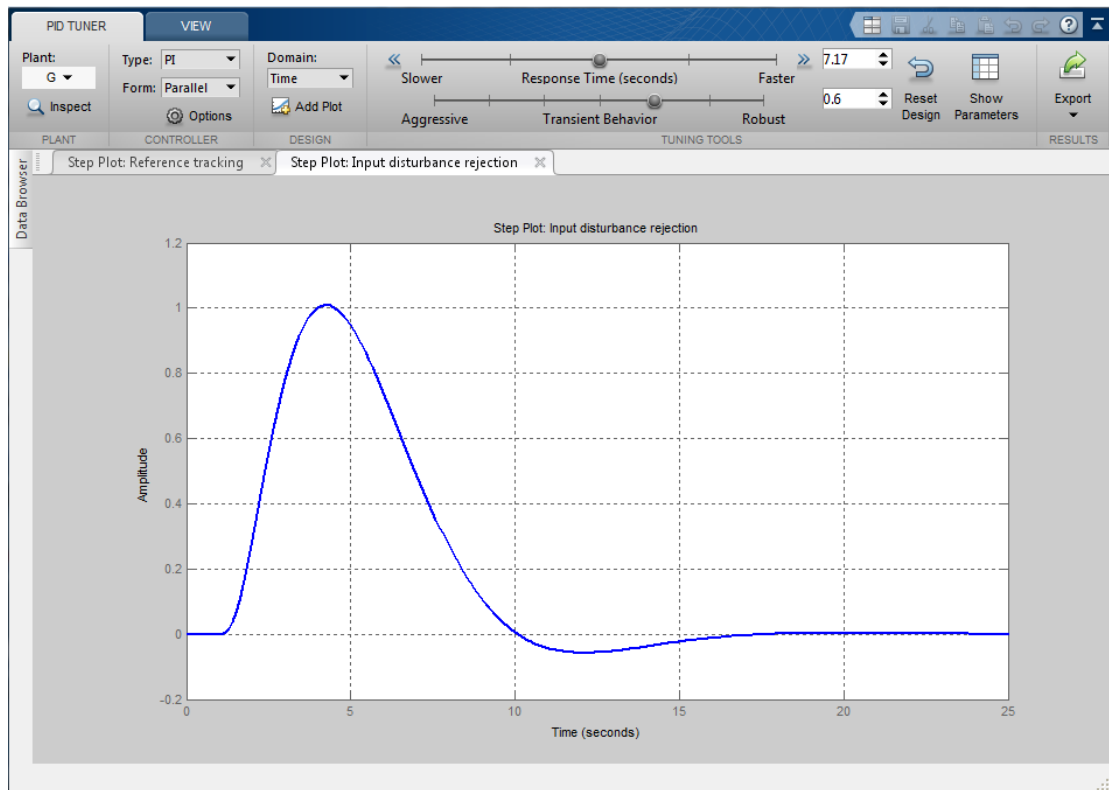
Tile the plots to show both the reference tracking and input disturbance responses. Move the response time slider to the right to increase the response speed (open loop bandwidth). The Ki gain in the **Controller parameters** table first increases and then decreases, with the maximum value occurring at 0.3. When Ki is 0.3, the peak deviation is reduced to 0.9 (about 10% improvement) and it settles to less than 0.1 in about 6.7 seconds (about 25% improvement).

Because we increased the bandwidth, the step reference tracking response becomes more oscillatory. Additionally the overshoot exceeds 15 percent, which is usually unacceptable. This type of performance trade off between reference tracking and disturbance rejection

often exists because a single PID controller is not able to satisfy both design goals at the same time.

Click **Export** to export the designed PI controller to the MATLAB Workspace. The controller is represented by a PID object and you need it to create an ISA-PID controller in the next section.

Export Linear System			×

☑ Export PID controller `C`

Export plant model

Selected	Plant Name	Type	Order
☐	G	zpk	4

✔ OK   ✖ Cancel   ? Help
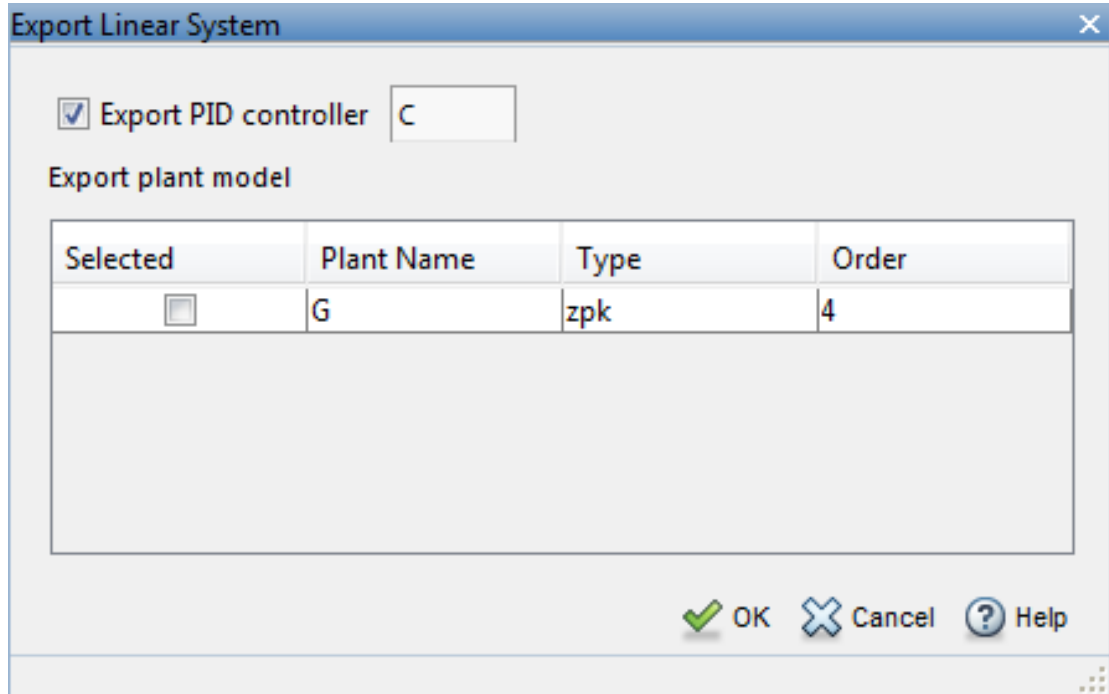
You can also manually create the same PI controller in MATLAB Workspace by using the **pid** command. In this command you can directly specify the Kp and Ki gains obtained from the parameter table of the PID Tuner.

```
C = pid(0.64362,0.30314);
C.InputName = 'e';
C.OutputName = 'u';
C
```

```
C =
```

```
          1
  Kp + Ki *  ---
             s

  with Kp = 0.644, Ki = 0.303

Continuous-time PI controller in parallel form.
```

Extending PID controller to ISA-PID Controller

A simple solution to make a PI controller perform well for both reference tracking
and disturbance rejection is to upgrade it to an ISA-PID controller. It improves
reference tracking response by providing an additional tuning parameters **b** that allows
independent control of the impact of the reference signal on the proportional action.
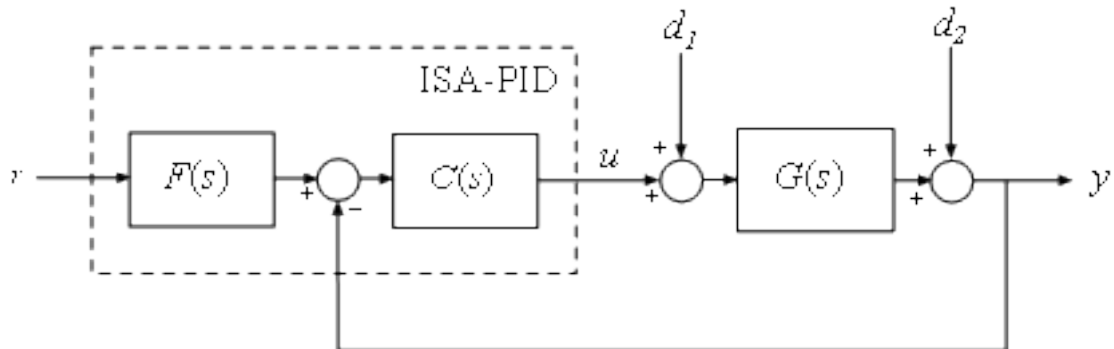
In the above ISA-PID structure, there is a feedback controller C and a feed-forward filter
F. In this example, C is a regular PI controller in parallel form that can be represented
by a PID object:

$$C(s) = pid(K_p, K_i) = K_p + \frac{K_i}{s}$$

F is a pre-filter that involves Kp and Ki gains from C plus the setpoint weight **b**:

$$F(s) = \frac{bK_ps + K_i}{K_ps + K_i}$$

Therefore the ISA-PID controller has two inputs (r and y) and one output (u).

Set-point weight **b** is a real number between 0 and 1. When it decreases, the overshoot in the reference tracking response is reduced. In this example, **b** is chosen to be 0.7.

```
b = 0.7;
% The following code constructs an ISA-PID from F and C
F = tf([b*C.Kp C.Ki],[C.Kp C.Ki]);
F.InputName = 'r';
F.OutputName = 'uf';
Sum = sumblk('e','uf','y','+-');
ISAPID = connect(C,F,Sum,{'r','y'},'u');
tf(ISAPID)
```

```
ans =

  From input "r" to output "u":
  0.4505 s^2 + 0.5153 s + 0.1428
  ------------------------------
          s^2 + 0.471 s

  From input "y" to output "u":
  -0.6436 s - 0.3031
  ------------------
          s

Continuous-time transfer function.
```

Compare Performance

The reference tracking response with ISA-PID controller has much less overshoot because setpoint weight **b** reduces overshoot.

```
% Closed-loop system with PI controller for reference tracking
sys1 = feedback(G*C,1);
% Closed-loop system with ISA-PID controller
sys2 = connect(ISAPID,G,{'r','u'},'y');
% Compare responses
step(sys1,'r-',sys2(1),'b.');
legend('show','location','southeast')
title('Reference Tracking')
```

Reference Tracking

From: In(1) To: y
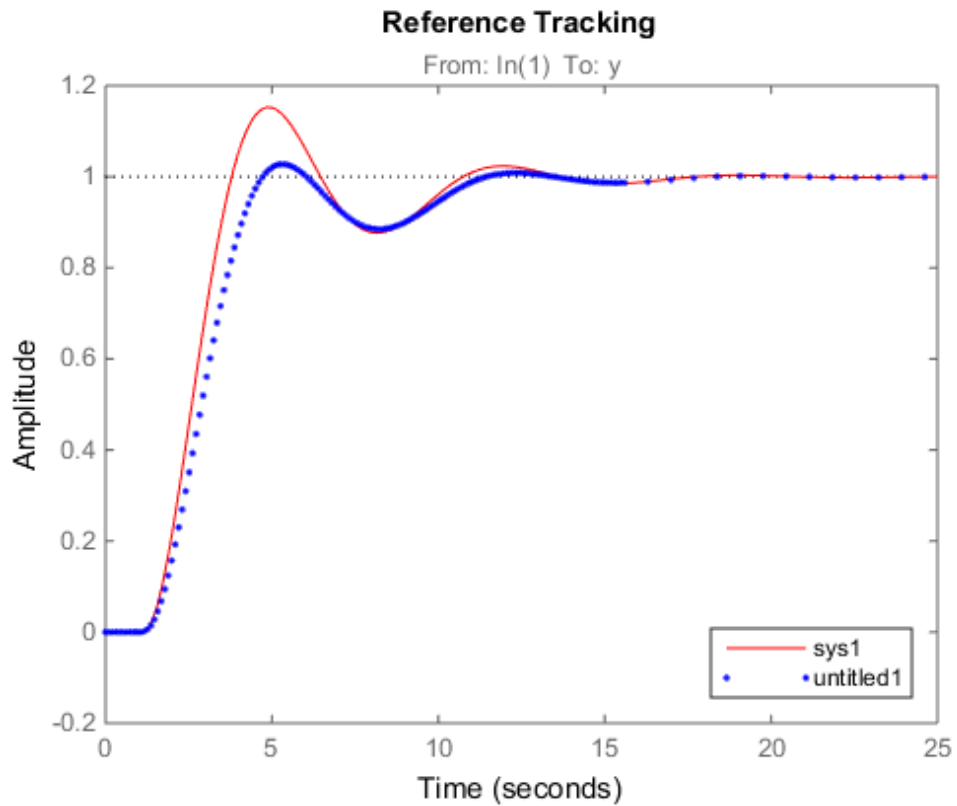
The disturbance rejection responses are the same because setpoint weight **b** only affects reference tracking.

```
% Closed-loop system with PI controller for disturbance rejection
sys1 = feedback(G,C);
% Compare responses
step(sys1,'r-',sys2(2),'b.');
legend('PID','ISA-PID');
title('Disturbance Rejection')
```
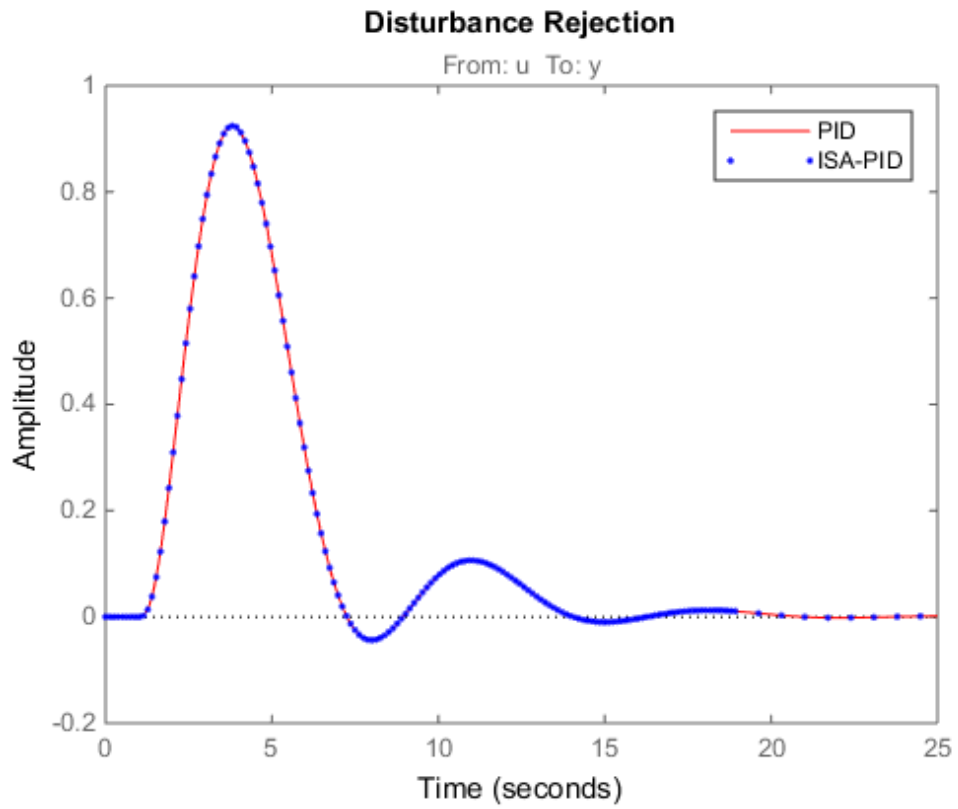
Tune PI Controller to Balance Tracking and Disturbance Rejection Performance

This example shows how to tune a PI controller to reduce overshoot in reference tracking or to improve rejection of a disturbance at the plant input. Using the PID Tuner, the example illustrates the trade-off between reference tracking and disturbance rejection performance.

Tune the control system of the following illustration. The plant G has poles at $s = 1$, $s = 0.2$, and $s = 0.05$.
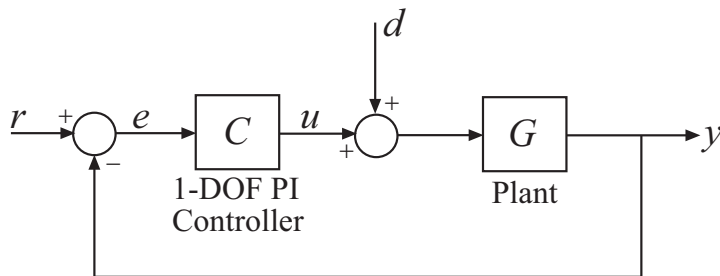
Setpoint tracking is the response at y to signals at r. Disturbance rejection is a measure of the suppression at y of signals at d.

1   Create a model of the plant, and open the PID Tuner to design an initial PI controller for the plant.

```
G = zpk([],[-1 -0.2 -0.05],1);
pidTuner(G,'pi')
```
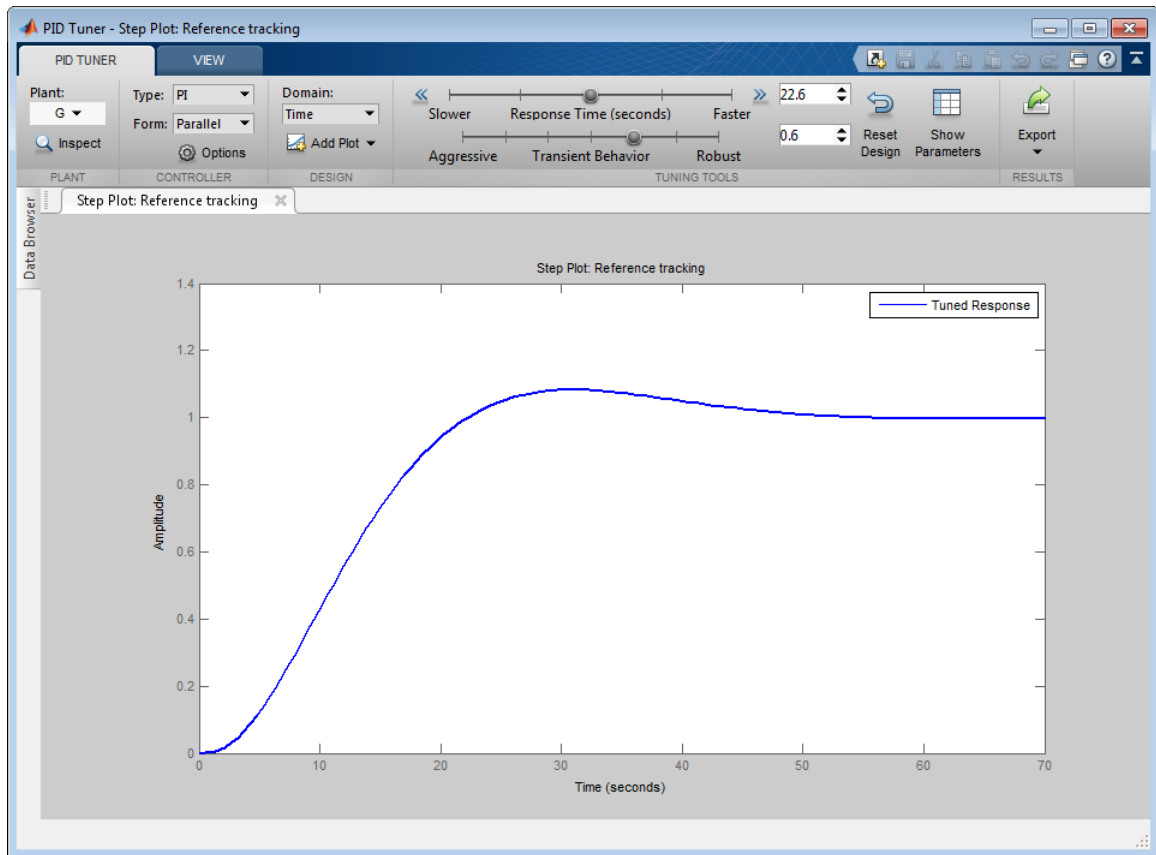
The initial controller design has some overshoot in the reference tracking response.

2 Right-click in the plot area, and select **Characteristics** > **Peak Response**. A blue dot appears at the point of maximum overshoot. Click the blue dot to display peak response data in a tool tip.
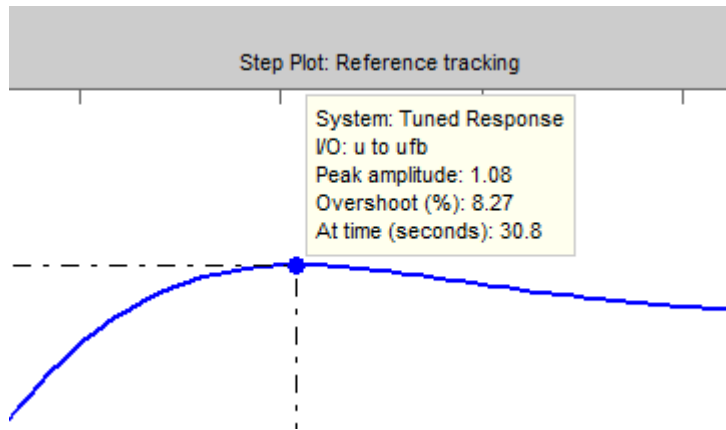
The tool tip shows that the response has more than 8% overshoot. If this overshoot is too much for your application, you can reduce it using the **Transient Behavior** slider.

3 Move the **Transient Behavior** slider to the right until the overshoot is less than 1%.
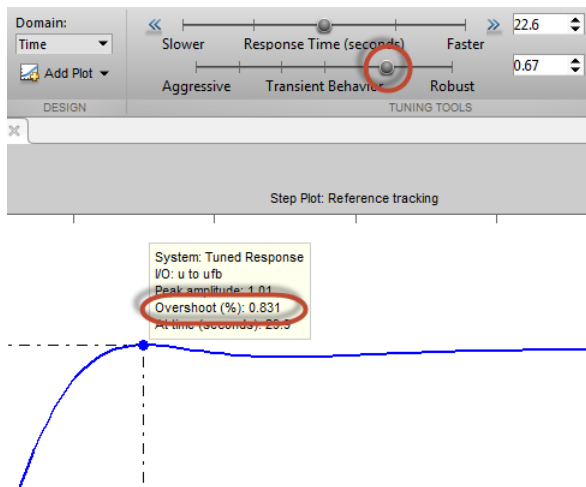
Increasing controller robustness also reduces overshoot, so moving the **Transient Behavior** slider to the right improves this measure of controller performance.

However, for a given controller bandwidth (or response time), there is a trade-off between reducing reference tracking overshoot and optimizing disturbance rejection.

4   Examine the effect of the overshoot reduction on the disturbance rejection performance. Click **Add Plot**. In the **Step** section of the **Add Plot** menu, select `Input disturbance rejection`. The disturbance rejection response appears in a new figure.

Tip Use the options in the **View** tab to change how PID Tuner displays multiple plots.

---

5   Right-click in the plot area, and select **Characteristics** > **Settling Time**. Click the blue dot to display the settling time tool tip.
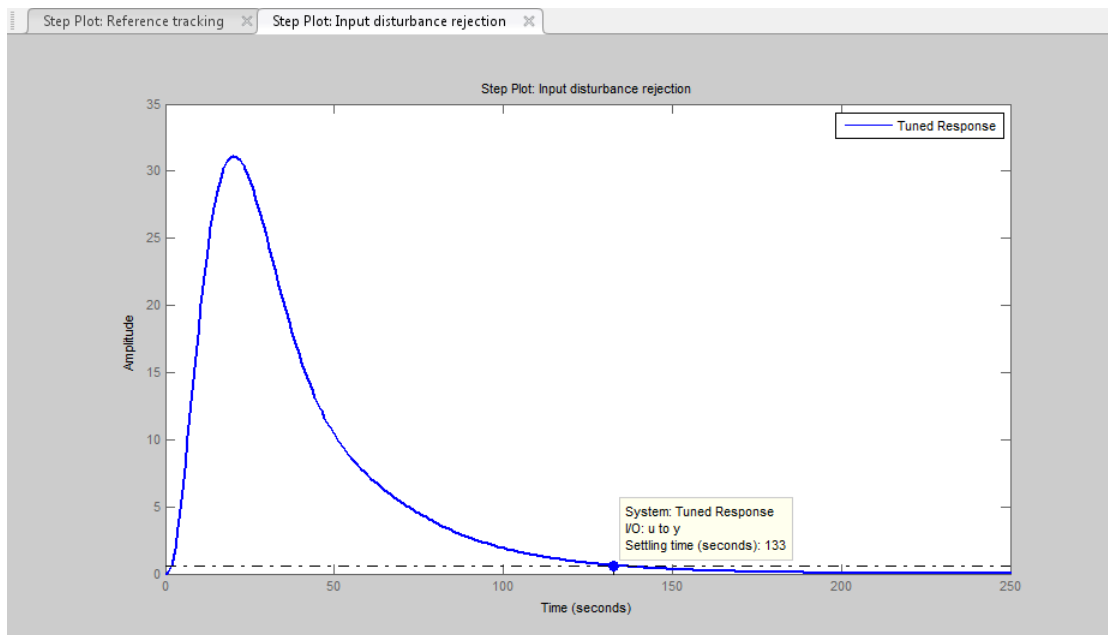
The current controller design minimizes reference tracking overshoot. However, it responds sluggishly to a disturbance at the plant input, taking over two minutes to settle. You can use the **Transient Behavior** slider to make the disturbance rejection more aggressive without changing the controller bandwidth.

6   Move the **Transient Behavior** slider to the left until the disturbance response settles in under 60 seconds.
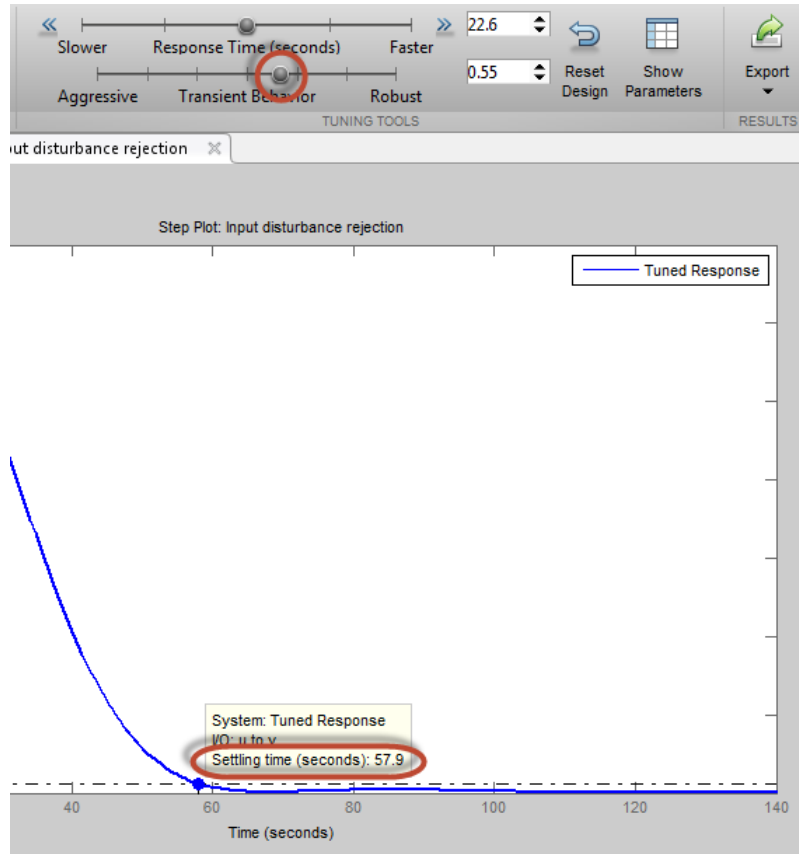
7   Examine the reference tracking response to see the effect of more aggressive transient behavior on the reference tracking overshoot. Select the **Step Plot: Reference tracking** figure to view this response plot, which updates automatically when you move the tuning sliders.
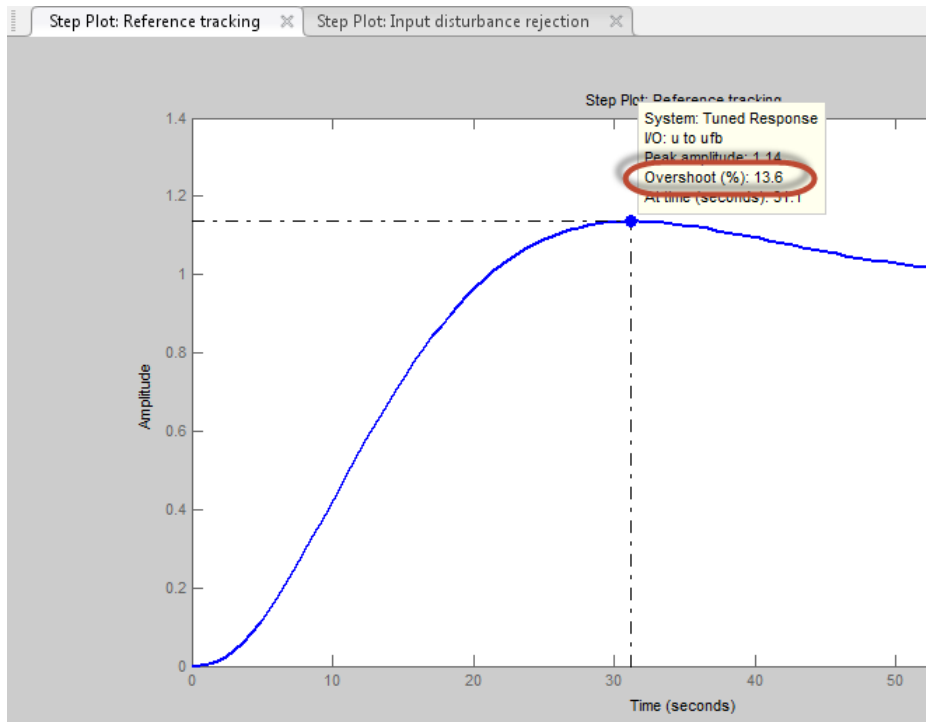
Optimizing the disturbance rejection response for the given bandwidth increases the reference tracking overshoot to over 13%.

The PID Tuner selects an initial controller design that balances this trade-off. The initial controller design has less reference tracking overshoot than the design optimized for disturbance rejection. Likewise, it has somewhat faster disturbance rejection than the design optimized to reduce overshoot. You can use the **Transient Behavior** slider to adjust this balance as needed to suit your application.

Tip To view multiple response plots side-by-side, use the options in the PID Tuner **View** tab.
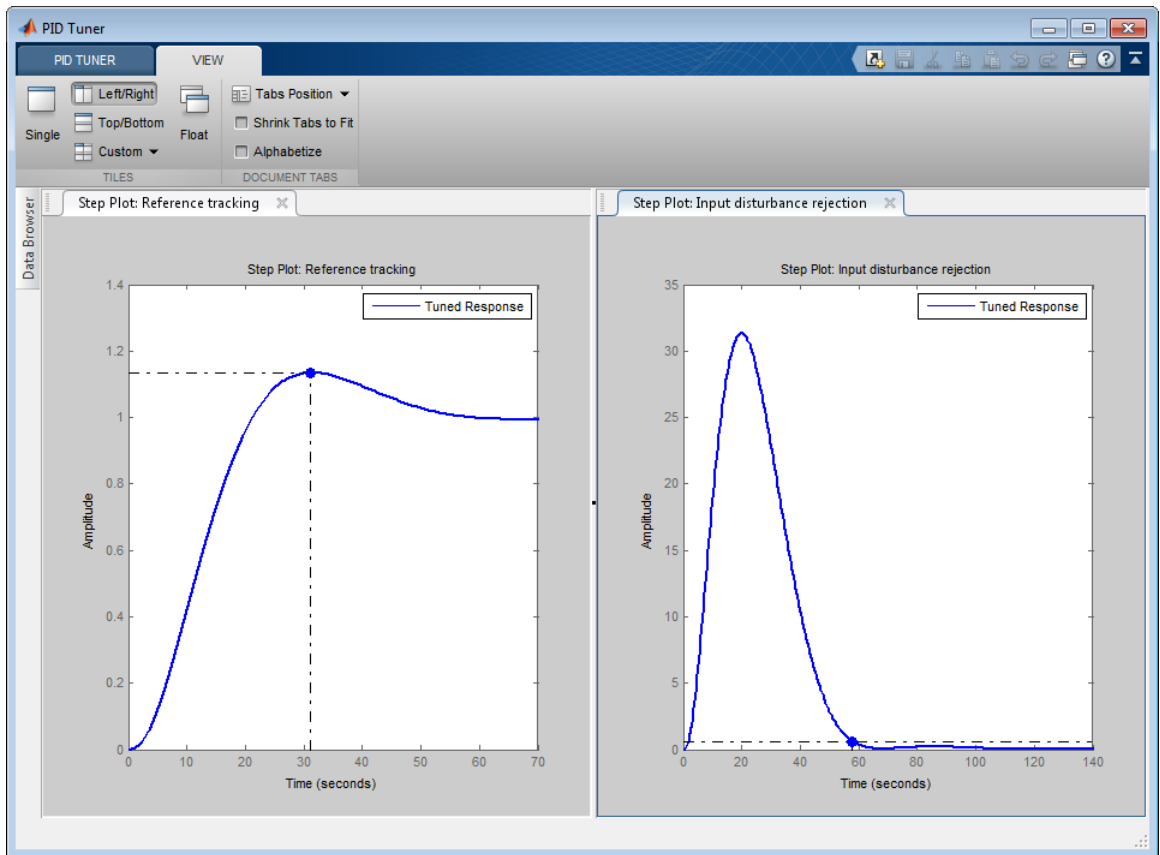
Related Examples

- "PID Controller Design for Fast Reference Tracking" on page 4-11
- "Designing PID for Disturbance Rejection with PID Tuner" on page 4-20

Interactively Estimate Plant Parameters from Response Data

This example shows how to use PID Tuner to fit a linear model to measured SISO response data.

If you have System Identification Toolbox software, you can use the PID Tuner to estimate the parameters of a linear plant model based on time-domain response data measured from your system. PID Tuner then tunes a PID controller for the resulting estimated model. PID Tuner gives you several techniques to graphically, manually, or automatically adjust the estimated model to match your response data. This example illustrates some of those techniques.

Import Response Data for Identification

1   Open the PID Tuner.

    pidTuner(tf(1),'PI')

    Load measured response data into the MATLAB workspace.

    load PIDPlantMeasuredIOData

    When you import response data, PID Tuner assumes that your measured data represents a plant connected to the PID controller in a negative-feedback loop. In other words, PID Tuner assumes the following structure for your system. PID Tuner assumes that you injected a step signal at u and measured the system response at y, as shown.
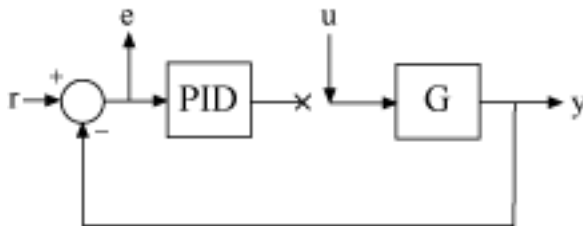
    The sample data file for this example, load PIDPlantMeasuredIOData.mat, contains three variables, each of which is a 501-by-1 array. inputu is the unit step function injected at u to obtain the response data. outputy, is the measured response of the system at y. The time vector t, runs from 0 to 50 s with a 0.1 s sample time. Comparing inputu to t shows that the step occurs at t = 5 s.

Tip You can import response data stored as a numeric array (as in this example), a `timeseries` object, or an `iddata` object.

2 In the PID Tuner, in the **Plant** menu, select `Identify New Plant`.
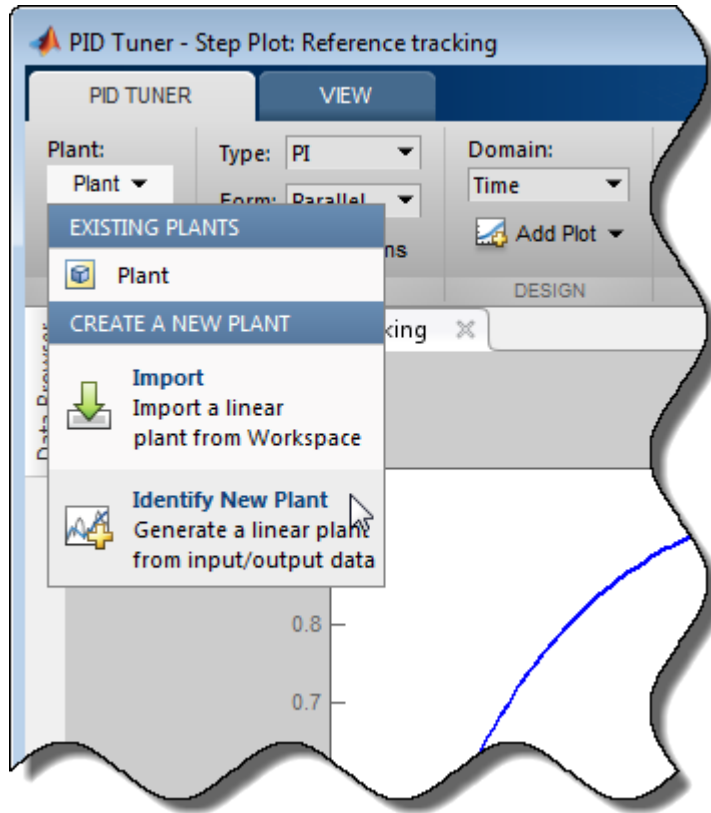
3 In the **Plant Identification** tab, click ✚ **Get I/O data** and select **Step Response**. This action opens the **Import Step Response** dialog box.

Enter information about the response data. The output signal is the measured system response, `outputy`. The input step signal is parametrized as shown in the

diagram in the dialog box. Here, enter 5 for the onset time, and 0.1 for sample time. Then, click ⬇ **Import**.
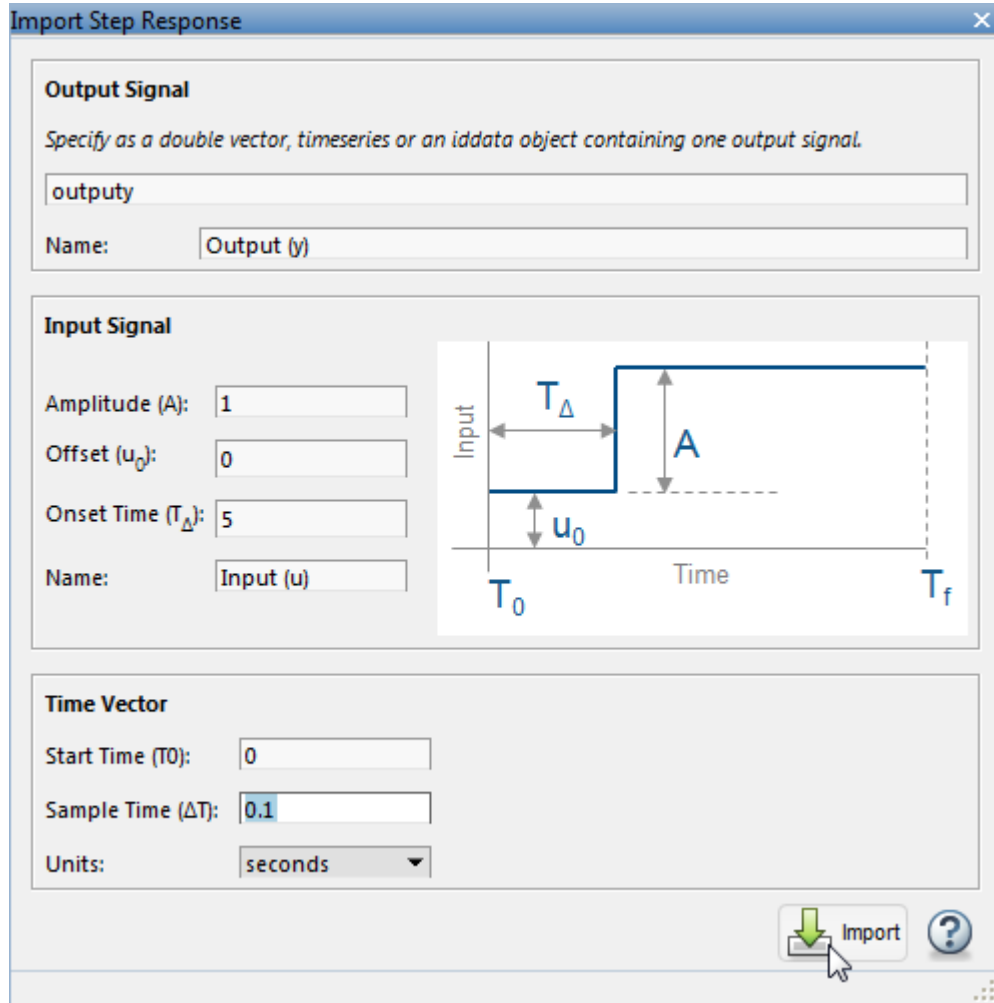
The **Plant Identification** tab opens, displaying the response data and the response of an initial estimated plant.
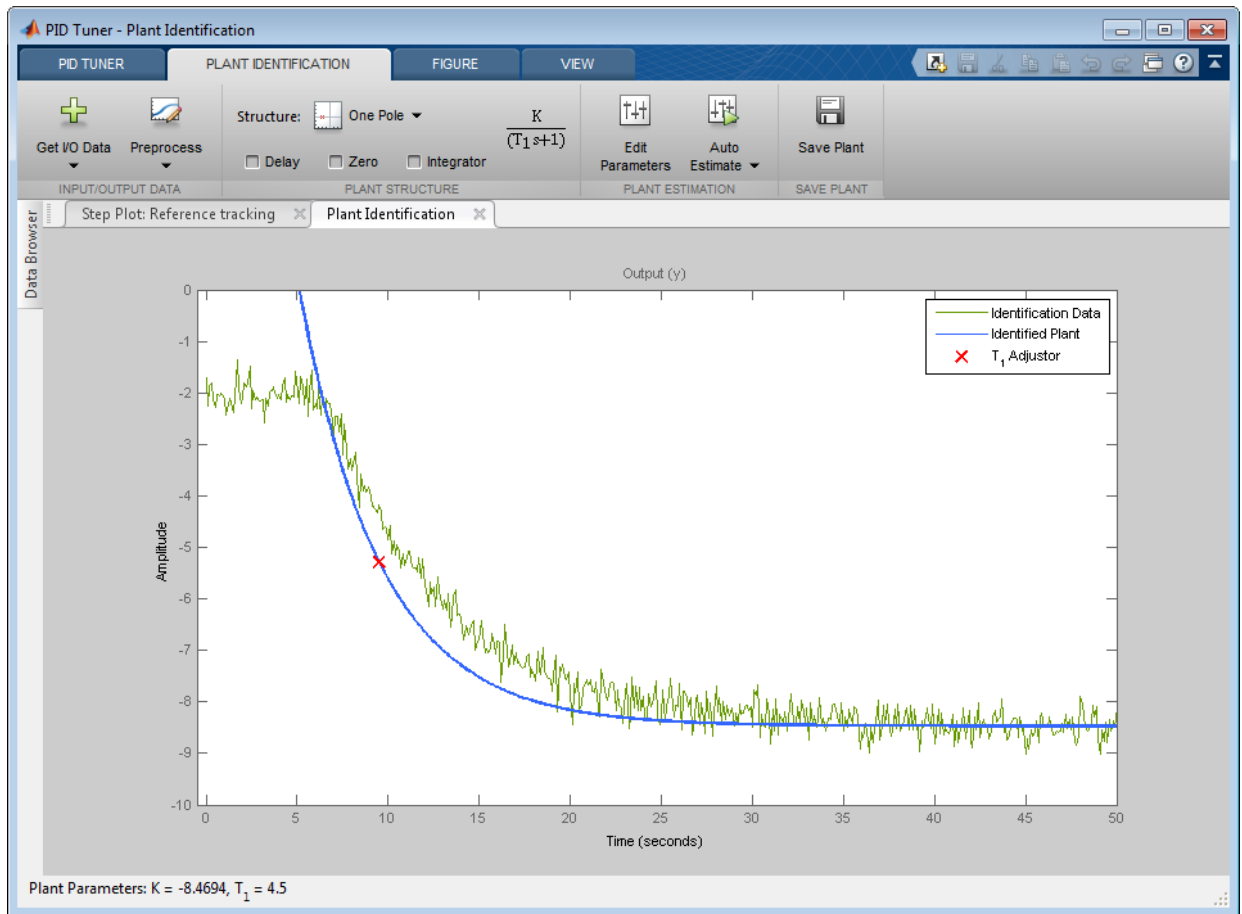
Preprocess Data

Depending on the quality and features of your response data, you might want to perform some preprocessing on the data to improve the estimated plant results. PID Tuner gives you several options for preprocessing response data, such as removing offsets, filtering, or extracting on a subset of the data. In this example, the response data has an offset. It is important for good identification results to remove data offsets. Use the **Preprocess** menu to do so.

1. In the **Plant Identification** tab, click ![icon] **Preprocess** and select `Remove Offset`. The **Remove Offset** tab opens, displaying time plots of the response data and corresponding input signal.

2. Select **Remove offset from signal** and choose the response, `Output (y)`. In the **Offset to remove** text box you can opt to remove the signal initial value or signal mean, or enter a numerical value. For this example, enter the value 2. The plot updates with an additional trace showing the signal with the offset applied.
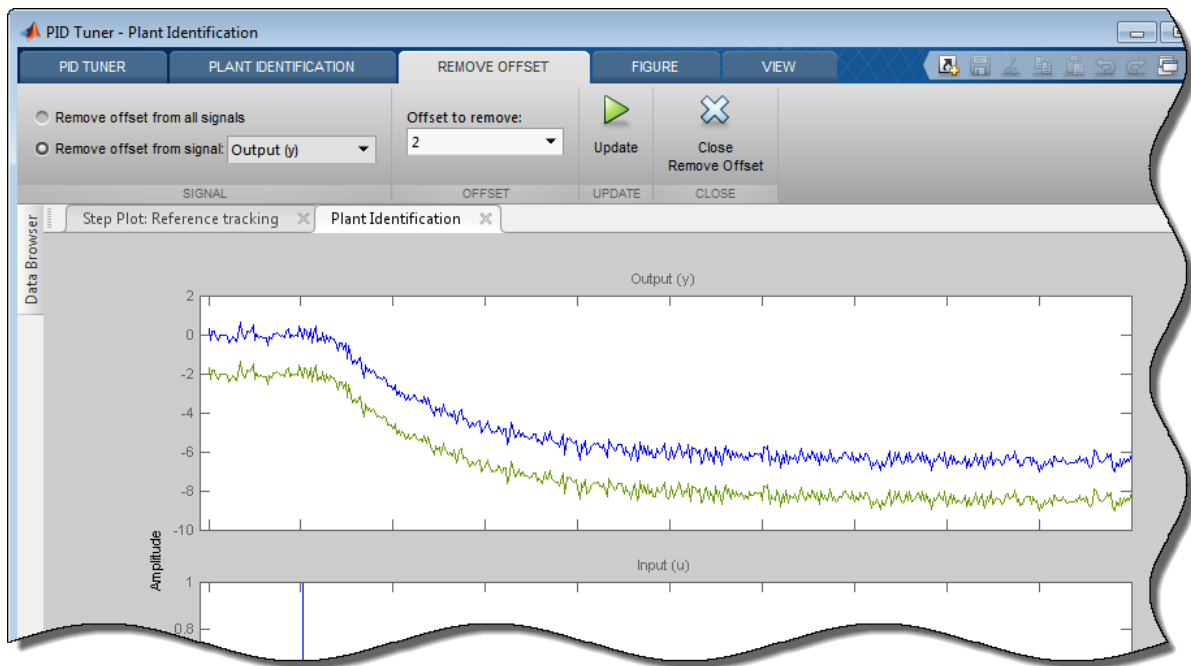
3. Click ![icon] **Update** to save the change to the signal. Click ![icon] **Close Remove Offset** to return to the **Plant Identification** tab.

    PID Tuner automatically adjusts the plant parameters to create a new initial guess for the plant based on the preprocessed response signal.

Adjust Plant Structure and Parameters

PID Tuner allows you to specify a plant structure, such as **One Pole**, **Underdamped Pair**, or **State-Space Model**. In the **Structure** menu, choose the plant structure that best matches your response. You can also add a transfer delay, a zero, or an integrator to your plant. For this example, the one-pole structure gives the qualitatively correct response. You can make further adjustments to the plant structure and parameter values to make the estimated system's response a better match to the measured response data.

PID Tuner gives you several ways to adjust the plant parameters:

• Graphically adjust the estimated system's response by dragging the adjustors on the plot. In this example, drag the red x to adjust the estimated plant time constant. PID Tuner recalculates system parameters as you do so. As you change the estimated system's response, it becomes apparent that there is some time delay between the application of the step input at $t = 5$ s, and the response of the system to that step input.
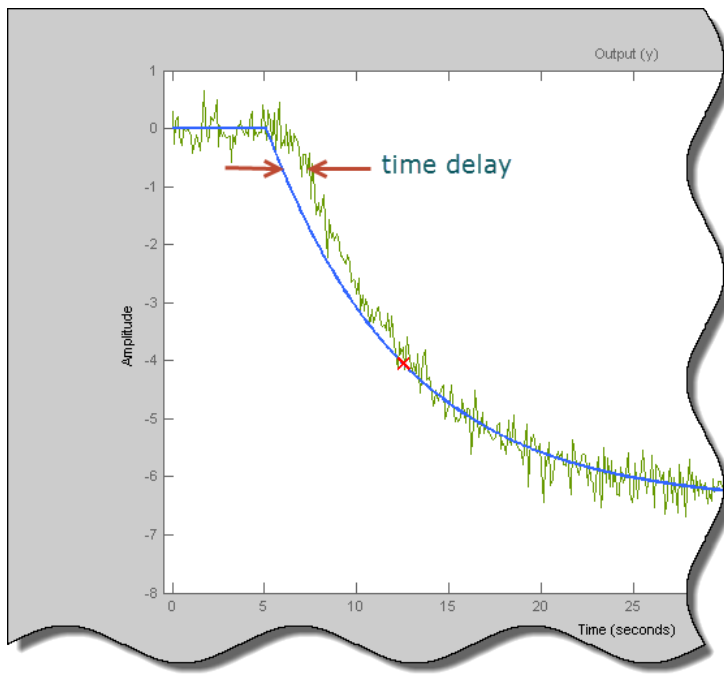
In the Plant Structure section of the tab, check **Delay** to add a transport delay to the estimated plant model. A vertical line appears on the plot, indicating the current value of the delay. Drag the line left or right to change the delay, and make further adjustments to the system response by dragging the red x.

- Adjust the numerical values of system parameters such as gains, time constants, and time delays. To numerically adjust the values of system parameters, click ⊞ **Edit Parameters**.

Suppose that in this example you know from an independent measurement that the transport delay in your system is 1.5 s. In the **Plant Parameters** dialog box, enter 1.5 for τ. Check **Fix** to fix the parameter value. When you check **Fix** for a parameter, neither graphical nor automatic adjustments to the estimated plant model affect that parameter value.

Plant Parameters ✕

$$\frac{K}{(T_1 s+1)} e^{-\tau s}$$

One Pole + Delay

K   -6.47   Fix

-77.2   12.9

T_1 6.43   Fix

τ 1.5   ☑ Fix

Reset
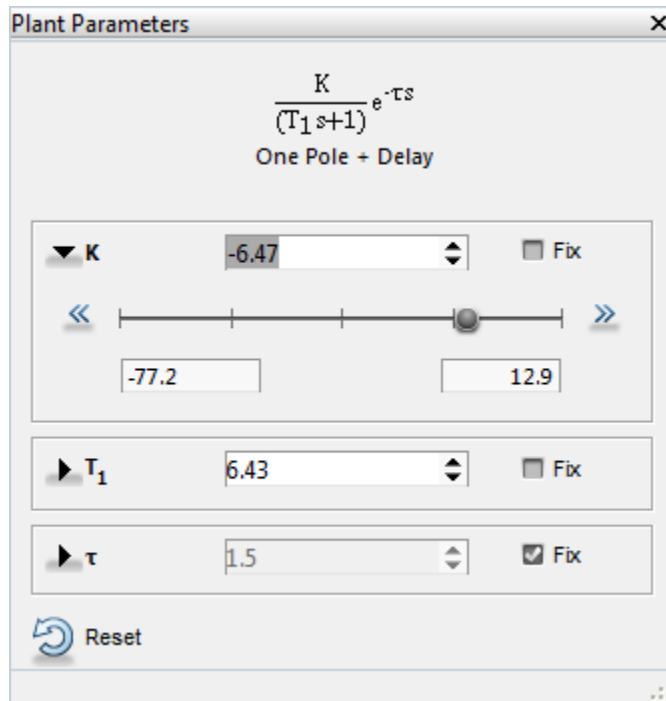
- Automatically optimize the system parameters to match the measured response data.

  Click **Auto Estimate** to update the estimated system parameters using the current values as an initial guess.

You can continue to iterate using any of these methods to adjust plant structure and parameter values until the estimated system's response adequately matches the measured response.

Save Plant and Tune PID Controller

When you are satisfied with the fit, click **Save Plant**. Doing so saves the estimated plant, `Plant1`, to the PID Tuner workspace. Doing so also selects the **Step Plot: Reference Tracking** figure and returns you to the **PID Tuner** tab. The PID Tuner automatically designs a PI controller for `Plant1`, and displays a response plot for the new closed-loop system. The **Plant** menu reflects that `Plant1` is selected for the current controller design.
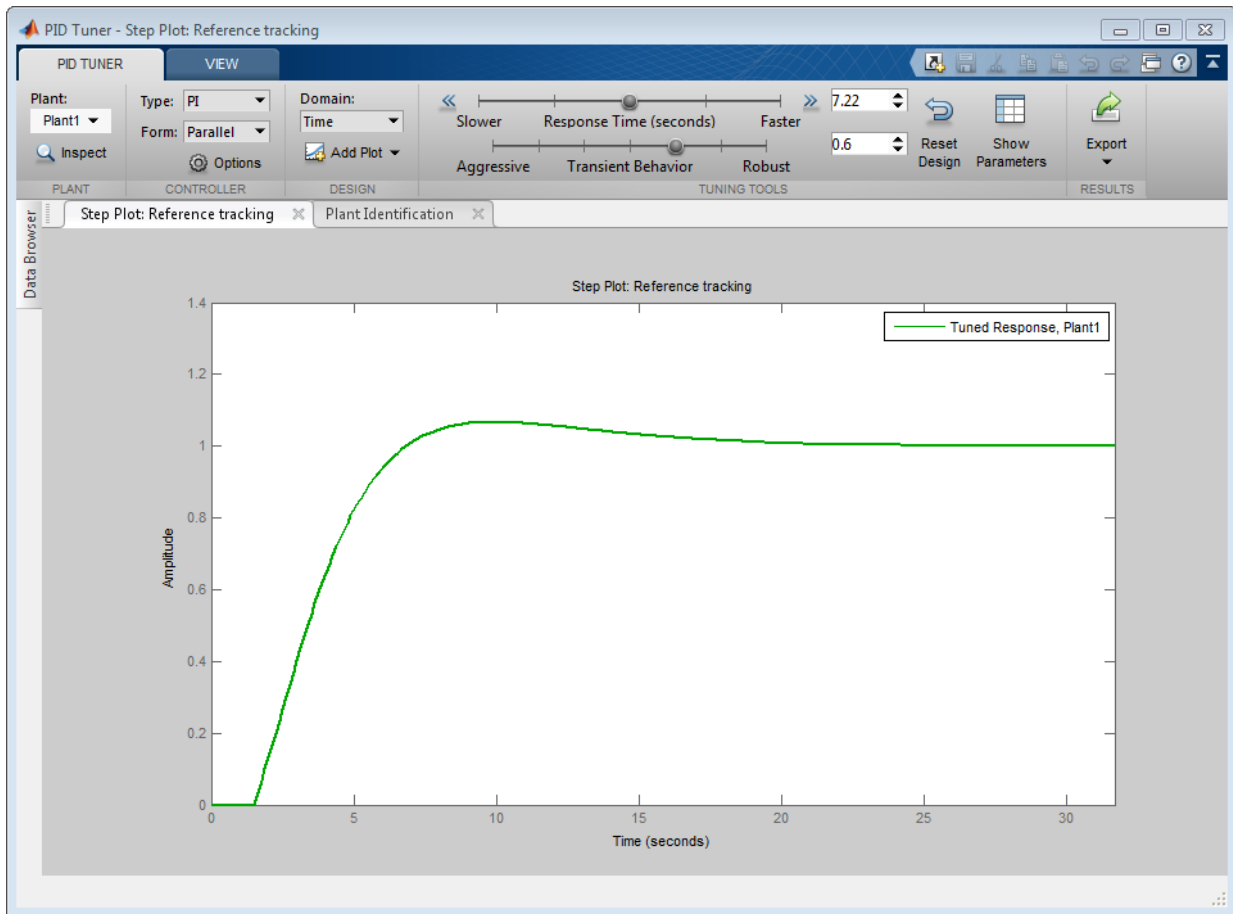
Tip To examine variables stored in the PID Tuner workspace, open the **Data Browser**.
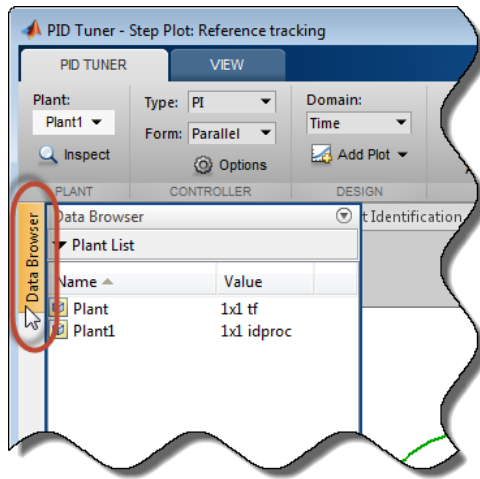
You can now use the PID Tuner tools to refine the controller design for the estimated plant and examine tuned system responses.

You can also export the identified plant from the PID Tuner workspace to the MATLAB workspace for further analysis. In the **PID Tuner** tab, click ![export icon] **Export**. Check the plant model you want to export to the MATLAB workspace. For this example, export `Plant1`, the plant you identified from response data. You can also export the tuned PID controller. Click ![checkmark icon] **OK**. The models you selected are saved to the MATLAB workspace.

Identified plant models are saved as identified LTI models, such as `idproc` or `idss`.

More About

- "System Identification for PID Control" on page 4-51

PID Tuning Algorithm

Typical PID tuning objectives include:

- Closed-loop stability — The closed-loop system output remains bounded for bounded input.
- Adequate performance — The closed-loop system tracks reference changes and suppresses disturbances as rapidly as possible. The larger the loop bandwidth (the frequency of unity open-loop gain), the faster the controller responds to changes in the reference or disturbances in the loop.
- Adequate robustness — The loop design has enough gain margin and phase margin to allow for modeling errors or variations in system dynamics.

MathWorks® algorithm for tuning PID controllers meets these objectives by tuning the PID gains to achieve a good balance between performance and robustness. The algorithm designs an initial controller by choosing a bandwidth to achieve that balance, based upon the open-loop frequency response of your linearized model. When you interactively change the response time, bandwidth, transient response, or phase margin using the PID Tuner interface, the algorithm computes new PID gains.

System Identification for PID Control

In this section...

Plant Identification

In many situations, a dynamic representation of the system you want to control is not readily available. One solution to this problem is to obtain a dynamical model using identification techniques. The system is excited by a measurable signal and the corresponding response of the system is collected at some sample rate. The resulting input-output data is then used to obtain a model of the system such as a transfer function or a state-space model. This process is called *system identification* or *estimation*. The goal of system identification is to choose a model that yields the best possible fit between the measured system response to a particular input and the model's response to the same input.

If you have a Simulink® model of your control system, you can simulate input/output data instead of measuring it. The process of estimation is the same. The system response to some known excitation is simulated, and a dynamical model is estimated based upon the resulting simulated input/output data.

Whether you use measured or simulated date for estimation, once a suitable plant model is identified, you impose control objectives on the plant based on your knowledge of the desired behavior of the system that the plant model represents. You then design a feedback controller to meet those objectives.

If you have System Identification Toolbox software, you can use PID Tuner for both plant identification and controller design in a single interface. You can import input/output data and use it to identify one or more plant models. Or, you can obtain simulated input/output data from a Simulink model and use that to identify one or more plant models. You can then design and verify PID controllers using these plants. The PID Tuner also allows you to directly import plant models, such as one you have obtained from an independent identification task.

For an overview of system identification, see "About System Identification" in the System Identification Toolbox documentation.

Linear Approximation of Nonlinear Systems for PID Control

The dynamical behavior of many systems can be described adequately by a linear relationship between the system's input and output. Even when behavior becomes nonlinear in some operating regimes, there are often regimes in which the system dynamics are linear. For example, the behavior of an operational amplifier or the lift-vs-force dynamics of aerodynamic bodies can be described by linear models, within a certain limited operating range of inputs. For such a system, you can perform an experiment (or a simulation) that excites the system only in its linear range of behavior and collect the input/output data. You can then use the data to estimate a linear plant model, and design a PID controller for the linear model.

In other cases, the effects of nonlinearities are small. In such a case, a linear model can provide a good approximation, such that the nonlinear deviations are treated as disturbances. Such approximations depend heavily on the input profile, the amplitude and frequency content of the excitation signal.

Linear models often describe the deviation of the response of a system from some equilibrium point, due to small perturbing inputs. Consider a nonlinear system whose output, $y(t)$, follows a prescribed trajectory in response to a known input, $u(t)$. The dynamics are described by $dx(t)/dt = f(x, u)$, $y = g(x,u)$. Here, x is a vector of internal states of the system, and y is the vector of output variables. The functions f and g, which can be nonlinear, are the mathematical descriptions of the system and measurement dynamics. Suppose that when the system is at an equilibrium condition, a small perturbation to the input, Δu, leads to a small perturbation in the output, Δy:

$$\Delta \dot{x} = \frac{\partial f}{\partial x} \Delta x + \frac{\partial f}{\partial u} \Delta u,$$
$$\Delta y = \frac{\partial g}{\partial x} \Delta x + \frac{\partial g}{\partial u} \Delta u.$$

For example, consider the system of the following Simulink block diagram:
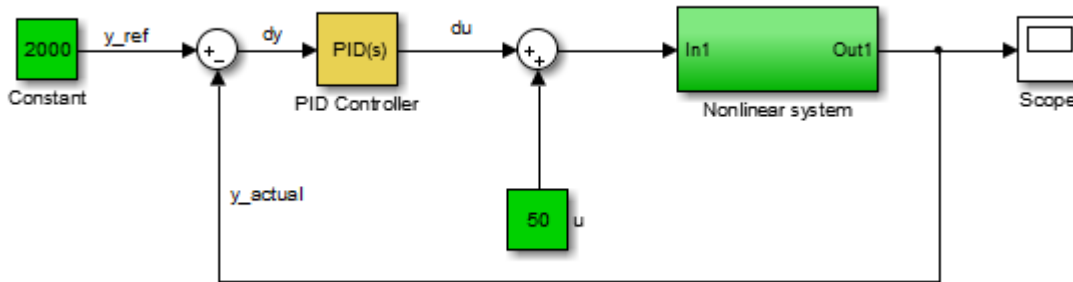
When operating in a disturbance-free environment, the nominal input of value 50 keeps the plant along its constant trajectory of value 2000. Any disturbances would cause the plant to deviate from this value. The PID Controller's task is to add a small correction to the input signal that brings the system back to its nominal value in a reasonable amount of time. The PID Controller thus needs to work only on the linear deviation dynamics even though the actual plant itself might be nonlinear. Thus, you might be able to achieve effective control over a nonlinear system in some regimes by designing a PID controller for a linear approximation of the system at equilibrium conditions.

Linear Process Models

A common use case is designing PID controllers for the steady-state operation of manufacturing plants. In these plants, a model relating the effect of a measurable input variable on an output quantity is often required in the form of a SISO plant. The overall system may be MIMO in nature, but the experimentation or simulation is carried out in a way that makes it possible to measure the incremental effect of one input variable on a selected output. The data can be quite noisy, but since the expectation is to control only the dominant dynamics, a low-order plant model often suffices. Such a proxy is obtained by collecting or simulating input-output data and deriving a process model (low order transfer function with unknown delay) from it. The excitation signal for deriving the data can often be a simple bump in the value of the selected input variable.

Advanced System Identification Tasks

In the PID Tuner, you can only identify single-input, single output, continuous-time plant models. Additionally, the PID Tuner cannot perform the following system identification tasks:

- Identify transfer functions of arbitrary number of poles and zeros. (PID Tuner can identify transfer functions up to three poles and one zero, plus an integrator and a time delay. PID Tuner can identify state-space models of arbitrary order.)
- Estimate the disturbance component of a model, which can be useful for separating measured dynamics from noise dynamics.
- Validate estimation by comparing the plant response against an independent dataset.
- Perform residual analysis.

If you need these enhanced identification features, import your data into the System Identification Tool (`systemIdentification`). Use the System Identification Tool to perform model identification and export the identified model to the MATLAB workspace. Then import the identified model into PID Tuner for PID controller design.

For more information about the System Identification Tool, see "Identify Linear Models Using System Identification App".

Input/Output Data for Identification

In this section...
"Data Preparation" on page 4-55
"Data Preprocessing" on page 4-55

Data Preparation

Identification of a plant model for PID tuning requires a single-input, single-output dataset.

If you have measured data, use the data import dialogs to bring in identification data. Some common sources of identification data are transient tests such as bump test and impact test. For such data, PID Tuner provides dedicated dialogs that require you to specify data for only the output signal while characterizing the input by its shape. For an example, see "Interactively Estimate Plant Parameters from Response Data".

If you want to obtain input/output data by simulating a Simulink model, the PID Tuner interface lets you specify the shape of the input stimulus used to generate the response. For an example, see the Simulink Control Design™ example "Design a PID Controller Using Simulated I/O Data."

Data Preprocessing

PID Tuner lets you preprocess your imported or simulated data. PID Tuner provides various options for detrending, scaling, and filtering the data.

It is strongly recommended to remove any equilibrium-related signal offsets from the input and output signals before proceeding with estimation. You can also filter the data to focus the signal contents to the frequency band of interest.

Some data processing actions can alter the nature of the data, which can result in transient data (step, impulse or wide pulse responses) to be treated as arbitrary input/output data. When that happens the identification plot does not show markers for adjusting the model time constants and damping coefficient.

For an example that includes a data-preprocessing step, see:

- "Interactively Estimate Plant Parameters from Response Data"

.

Choosing Identified Plant Structure

PID Tuner provides two types of model structures for representing the plant dynamics: process models and state-space models.

Use your knowledge of system characteristics and the level of accuracy required by your application to pick a model structure. In absence of any prior information, you can gain some insight into the order of dynamics and delays by analyzing the experimentally obtained step response and frequency response of the system. For more information see the following topics in the System Identification Toolbox documentation:

· "Identifying Impulse-Response Models"

· "Identifying Frequency-Response Models"

Each model structure you choose has associated dynamic elements, or *model parameters*. You adjust the values of these parameters manually or automatically to find an identified model that yields a satisfactory match to your measured or simulated response data. In many cases, when you are unsure of the best structure to use, it helps to start with the simplest model structure, transfer function with one pole. You can progressively try identification with higher-order structures until a satisfactory match between the plant response and measured output is achieved. The state-space model structure allows an automatic search for optimal model order based on an analysis of the input-output data.

When you begin the plant identification task, a transfer function model structure with one real pole is selected by default. This default set up is not sensitive to the nature of the data and may not be a good fit for your application. It is therefore strongly recommended that you choose a suitable model structure before performing parameter identification.

In this section...
"Process Models" on page 4-58
"State-Space Models" on page 4-61
"Existing Plant Models" on page 4-63
"Switching Between Model Structures" on page 4-64
"Estimating Parameter Values" on page 4-64
"Handling Initial Conditions" on page 4-65

Process Models

Process models are transfer functions with 3 or fewer poles, and can be augmented by addition of zero, delay and integrator elements. Process models are parameterized by model parameters representing time constants, gain, and time delay. In PID Tuner, choose a process model in the **Plant Identification** tab using the **Structure** menu.
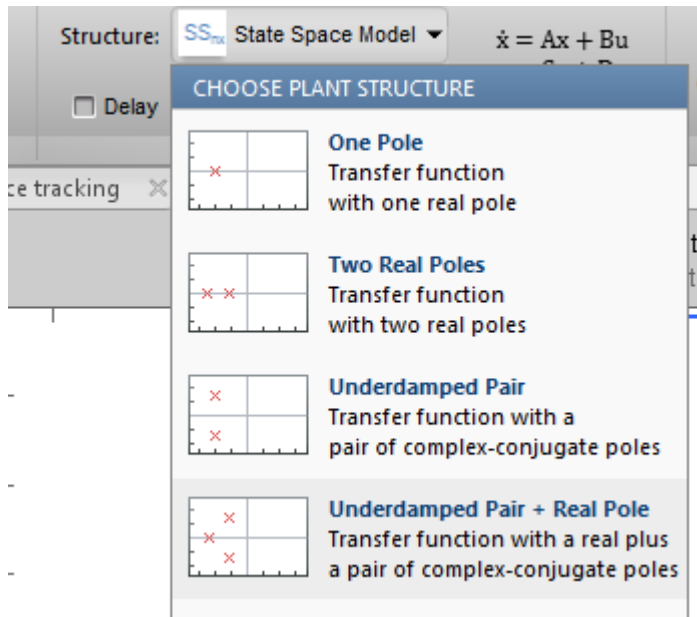
For any chosen structure you can optionally add a delay, a zero and/or an integrator element using the corresponding checkboxes. The model transfer function configured by these choices is displayed next to the **Structure** menu.
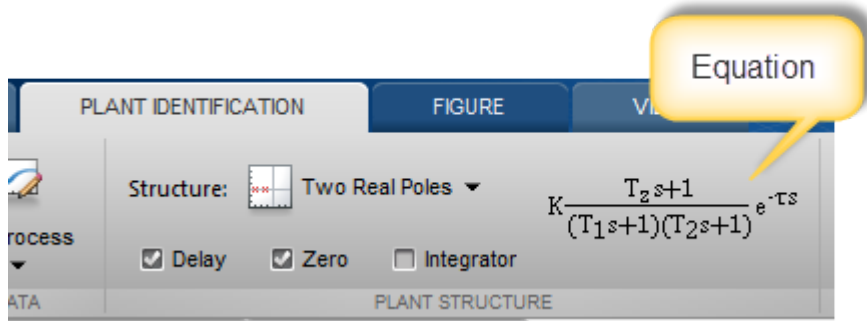
The simplest available process model is a transfer function with one real pole and no zero or delay elements:

$$H(s) = \frac{K}{T_1 s + 1}.$$

This model is defined by the parameters K, the gain, and T_1, the first time constant. The most complex process-model structure choose has three poles, an additional integrator, a zero, and a time delay, such as the following model, which has one real pole and one complex conjugate pair of poles:

$$H(s) = K \frac{T_z s + 1}{s(T_1 s + 1)(T_\omega^2 s^2 + 2\zeta T_\omega s + 1)} e^{-\tau s}.$$

In this model, the configurable parameters include the time constants associated with the poles and the zero, T_1, T_ω, and T_z. The other parameters are the damping coefficient ζ, the gain K, and the time delay τ.

When you select a process model type, the PID Tuner automatically computes initial values for the plant parameters and displays a plot showing both the estimated model response and your measured or simulated data. You can edit the parameter values graphically using indicators on the plot, or numerically using the Plant Parameters editor. For an example illustrating this process, see "Interactively Estimate Plant Parameters from Response Data".

The following table summarizes the various parameters that define the available types of process models.

Parameter	Used By	Description
K — Gain	All transfer functions	Can take any real value. In the plot, drag the plant response curve (blue) up or down to adjust K.
T_1 — First time constant	Transfer function with one or more real poles	Can take any value between 0 and T, the time span of measured or simulated data. In the plot, drag the red x left (towards zero) or right (towards T) to adjust T_1.
T_2— Second time constant	Transfer function with two real poles	Can take any value between 0 and T, the time span of measured or simulated data. In the plot, drag the magenta x left (towards zero) or right (towards T) to adjust T_2.
T_ω — Time constant associated with the natural frequency ω_n, where $T_\omega = 1/\omega_n$	Transfer function with underdamped pair (complex conjugate pair) of poles	Can take any value between 0 and T, the time span of measured or simulated data. In the plot, drag one of the orange response envelope curves left (towards zero) or right (towards T) to adjust T_ω.
ζ — Damping coefficient	Transfer function with underdamped pair (complex conjugate pair) of poles	Can take any value between 0 and 1. In the plot, drag one of the orange response envelope curves left (towards zero) or right (towards T) to adjust ζ.

Parameter	Used By	Description
τ — Transport delay	Any transfer function	Can take any value between 0 and T, the time span of measured or simulated data. In the plot, drag the orange vertical bar left (towards zero) or right (towards T) to adjust τ.
T_z — Model zero	Any transfer function	Can take any value between $-T$ and T, the time span of measured or simulated data. In the plot, drag the red circle left (towards $-T$) or right (towards T) to adjust T_z.
Integrator	Any transfer function	Adds a factor of $1/s$ to the transfer function. There is no associated parameter to adjust.

State-Space Models

The state-space model structure for identification is primarily defined by the choice of number of states, the *model order*. Use the state-space model structure when higher order models than those supported by process model structures are required to achieve a satisfactory match to your measured or simulated I/O data. In the state-space model structure, the system dynamics are represented by the state and output equations:

$$\dot{x} = Ax + Bu,$$
$$y = Cx + Du.$$

x is a vector of state variables, automatically chosen by the software based on the selected model order. u represents the input signal, and y the output signals.

To use a state-space model structure, in the **Plant Identification** tab, in the **Structure** menu, select `State-Space Model`. Then click **Configure Structure** to open the **State-Space Model Structure** dialog box.

State Space Model Structure   ✕

$$\dot{x} = Ax + Bu$$
$$y = Cx + Du$$

Plant Order:

◉ Specify value: 1

◯ Pick best value in the range: 1:10

Input delay: 0

☐ Allow feedthrough (nonzero D matrix)

⑦
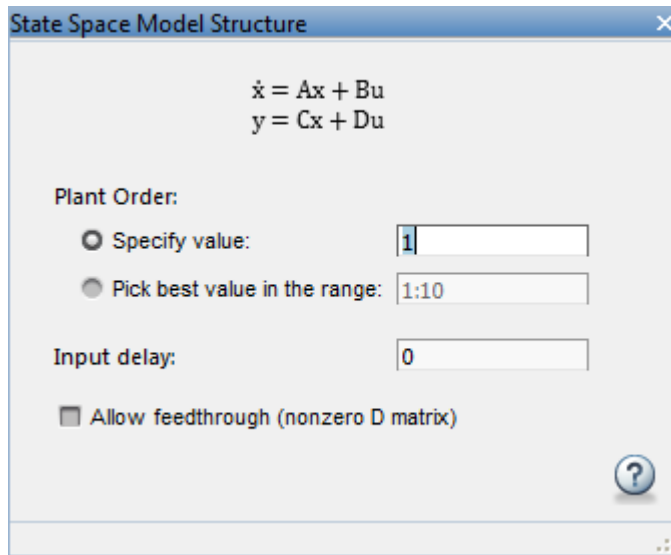
Use the dialog box to specify model order, delay and feedthrough characteristics. If you are unsure about the order, select **Pick best value in the range**, and enter a range of orders. In this case, when you click **Estimate** in the **Plant Estimation** tab, the software displays a bar chart of Hankel singular values. Choose a model order equal to the number of Hankel singular values that make significant contributions to the system dynamics.

When you choose a state-space model structure, the identification plot shows a plant response (blue) curve only if a valid estimated model exists. For example, if you change structure after estimating a process model, the state-space equivalent of the estimated model is displayed. If you change the model order, the plant response curve disappears until a new estimation is performed.

When using the state-space model structure, you cannot directly interact with the model parameters. The identified model should thus be considered unstructured with no physical meaning attached to the state variables of the model.

However, you can graphically adjust the input delay and the overall gain of the model. When you select a state-space model with a time delay, the delay is represented on the plot by a vertical orange bar is shown on the plot. Drag this bar horizontally to change the delay value. Drag the plant response (blue) curve up and down to adjust the model gain.

Existing Plant Models

Any previously imported or identified plant models are listed the **Plant List** section of the Data Browser. You can define the model structure and initialize the model parameter values using one of these plants. A process model (`idproc` object) is retained as is, while other model types are converted into the state-space model structure.
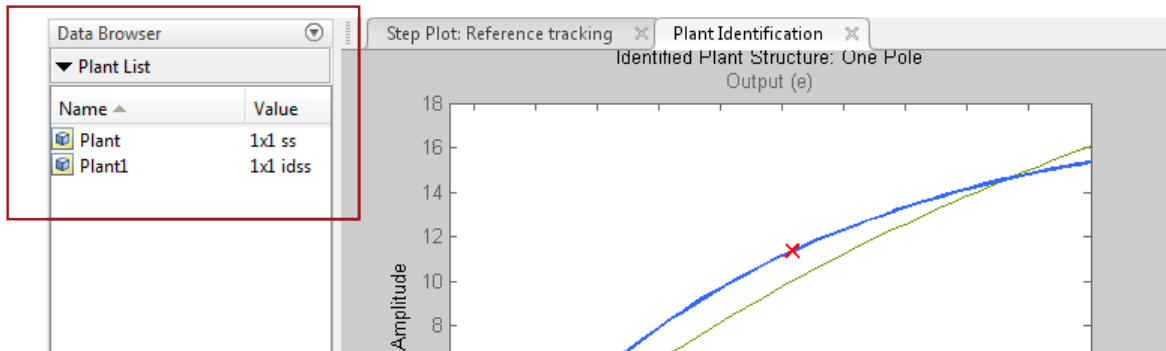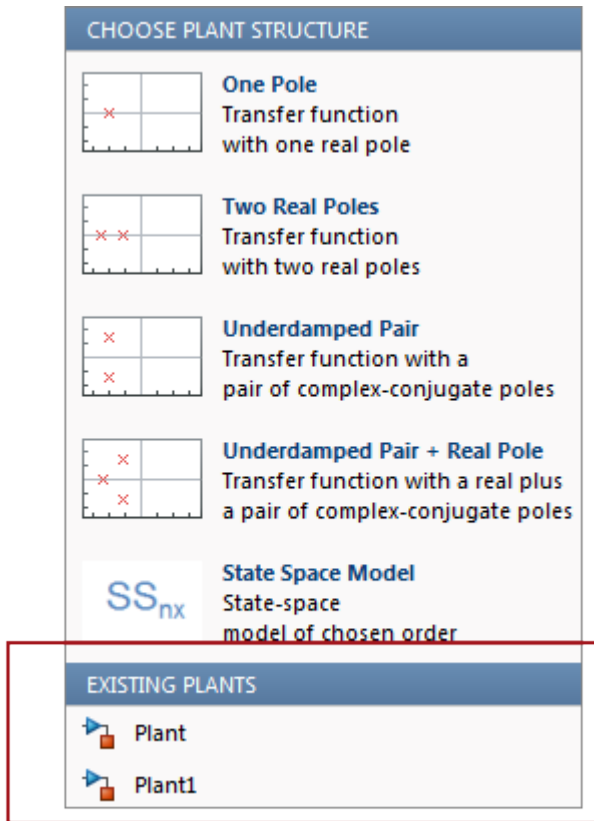
Switching Between Model Structures

When you switch from one model structure to another, the software preserves the model characteristics (pole/zero locations, gain, delay) as much as possible. For example, when you switch from a one-pole model to a two-pole model, the existing values of T_1, T_z, τ and K are retained, T_2 is initialized to a default (or previously assigned, if any) value.

Estimating Parameter Values

Once you have selected a model structure, you have several options for manually or automatically adjusting parameter values to achieve a good match between the estimated

model response and your measured or simulated input/output data. For an example that illustrates all these options, see:

- "Interactively Estimate Plant Parameters from Response Data" (Control System Toolbox)
- "Interactively Estimate Plant from Measured or Simulated Response Data" Simulink Control Design)

The PID Tuner does not perform a smart initialization of model parameters when a model structure is selected. Rather, the initial values of the model parameters, reflected in the plot, are arbitrarily-chosen middle of the range values. If you need a good starting point before manually adjusting the parameter values, use the **Initialize and Estimate** option from the **Plant Identification** tab.

Handling Initial Conditions

In some cases, the system response is strongly influenced by the initial conditions. Thus a description of the input to output relationship in the form of a transfer function is insufficient to fit the observed data. This is especially true of systems containing weakly damped modes. PID Tuner allows you to estimate initial conditions in addition to the model parameters such that the sum of the initial condition response and the input response matches the observed output well. Use the **Estimation Options** dialog box to specify how the initial conditions should be handled during automatic estimation. By default, the initial condition handling (whether to fix to zero values or to estimate) is automatically performed by the estimation algorithm. However, you can enforce a certain choice by using the Initial Conditions menu.

Initial conditions can only be estimated with automatic estimation. Unlike the model parameters, they cannot be modified manually. However, once estimated they remain fixed to their estimated values, unless the model structure is changed or new identification data is imported.

If you modify the model parameters after having performed an automatic estimation, the model response will show a fixed contribution (i.e., independent of model parameters) from initial conditions. In the following plot, the effects of initial conditions were identified to be particularly significant. When the delay is adjusted afterwards, the portion of the response to the left of the input delay marker (the τ Adjustor) comes purely from initial conditions. The portion to the right of the τ Adjustor contains the effects of both the input signal as well as the initial conditions.
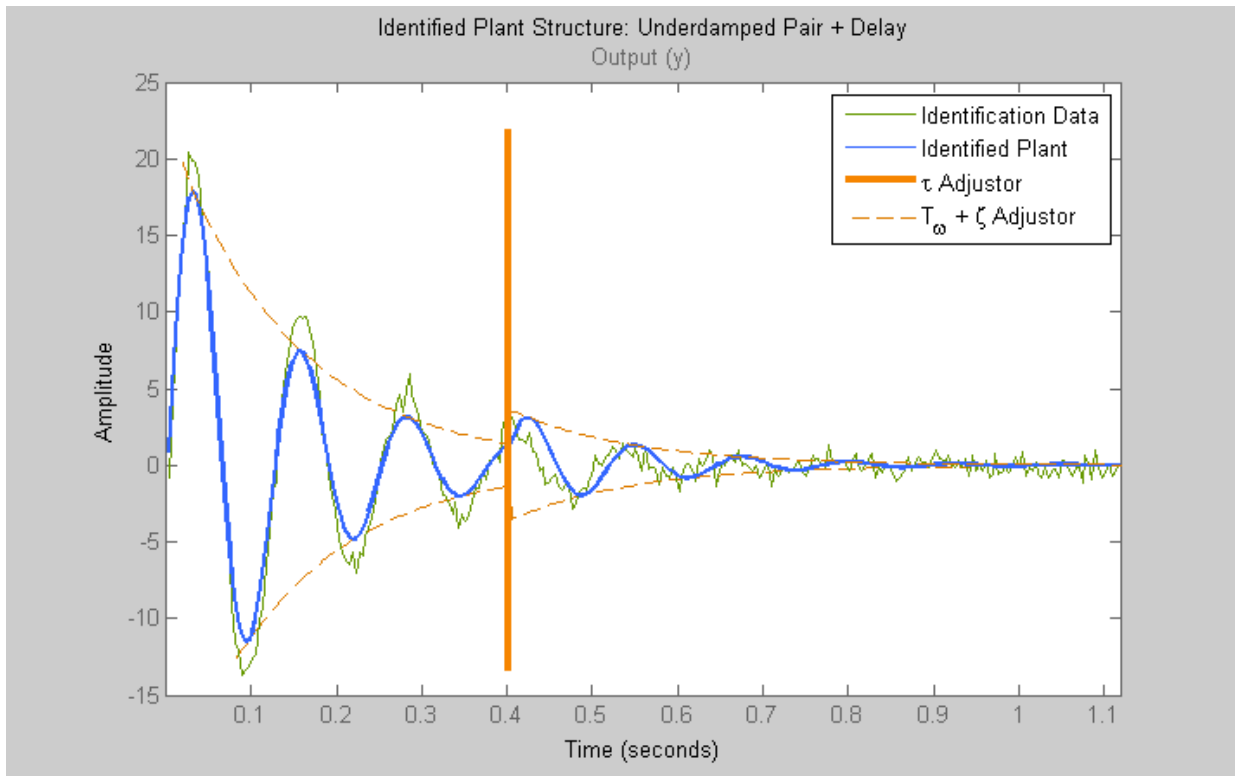
SISO Design Tool

In this section...

Tune Compensator Parameters Using the SISO Design Tool

In this example, you get an overview of the steps for tuning compensator parameters. Before you begin, you need:

- A model in the MATLAB workspace representing your plant
- Design requirements for your system

To tune parameters:

1 Open the control design GUIs by typing the following in the MATLAB Command Window:

    ```
    sisotool
    ```
    This command opens the Control and Estimation Tools Manager and the SISO Design Tool, as shown in the following figure.
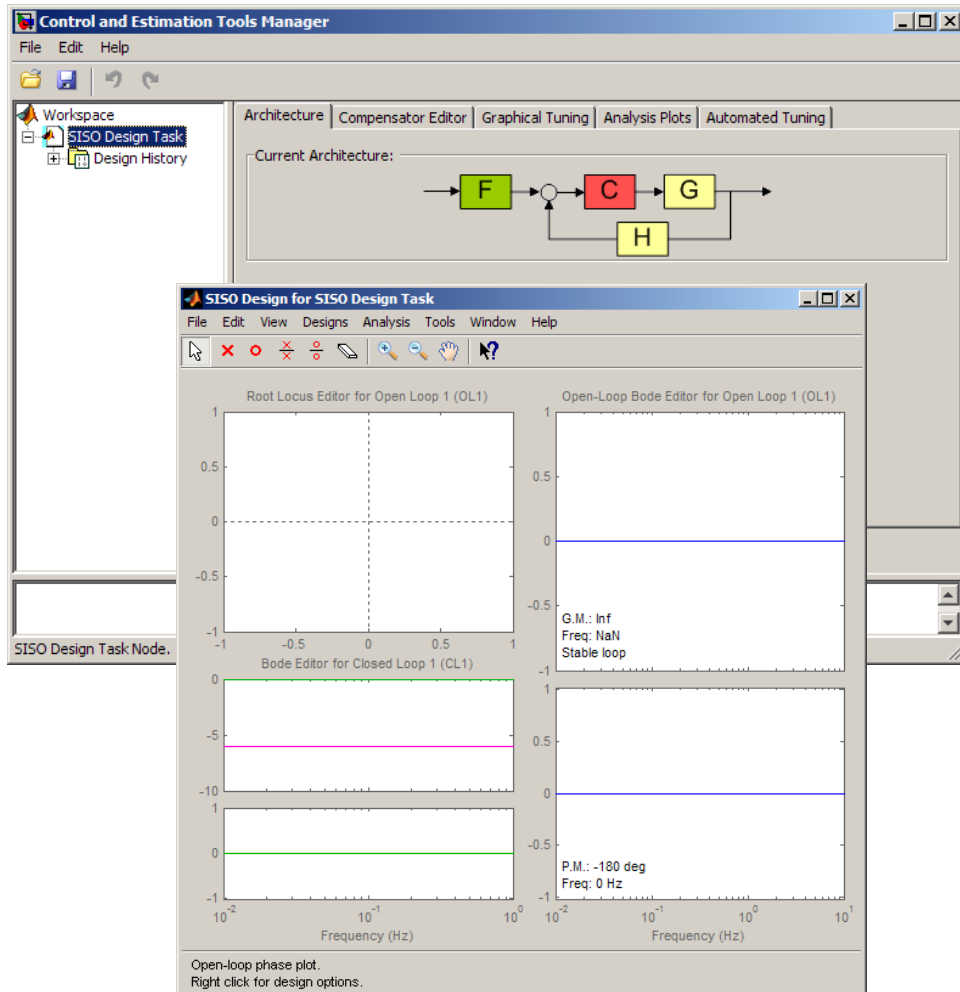
2   Define the control architecture for your system.

a   In the Architecture tab of the Control and Estimation Tools Manager, click
   Control Architecture.

b   In the Control Architecture dialog box, select the control architecture.

c   Specify the sign of summing junctions as + or -.

Select control architecture
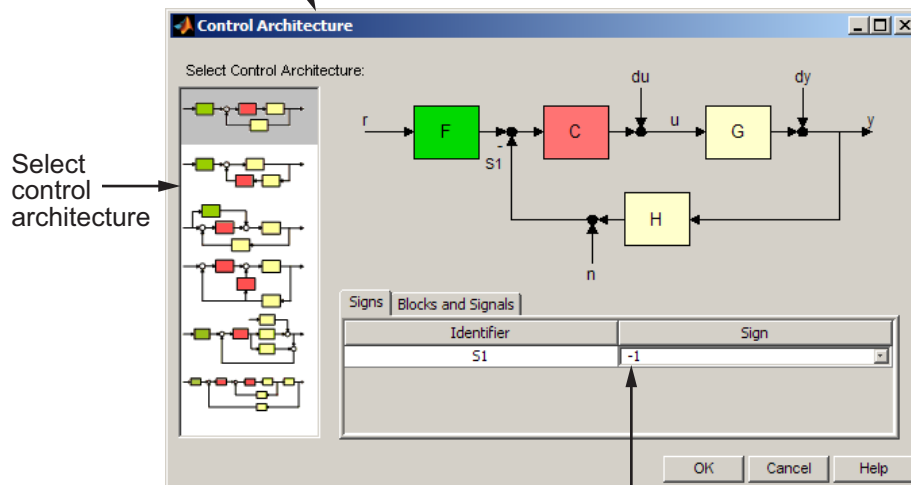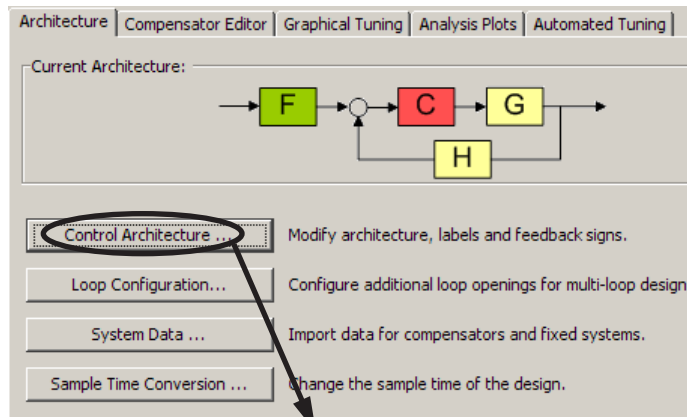
Specify sign of summing junctions as + or -

3　Specify the plant model and initial guesses for the compensator elements in the control architecture.

　　a　In the Architecture tab of the Control and Estimation Tools Manager, click **System Data**.

　　b　In the System Data dialog box, browse for a model in the MATLAB workspace.

Browse for
model in MATLAB
workspace
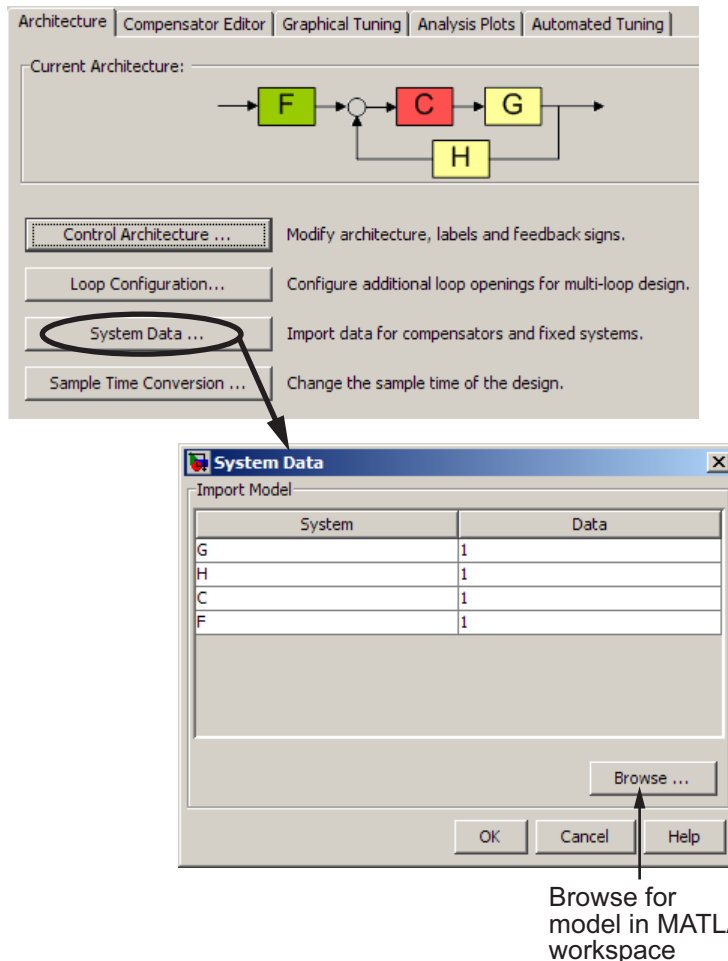
4 Design a compensator using automated tuning, for example PID Tuning.

  a In the Automated Tuning tab of the Control and Estimation Tools Manager,
      select an automated tuning method.

  b Specify tuning options.

  c Click **Update Compensator** to tune parameters.

Select
automated
tuning method

Specify
tuning options

Tune parameters
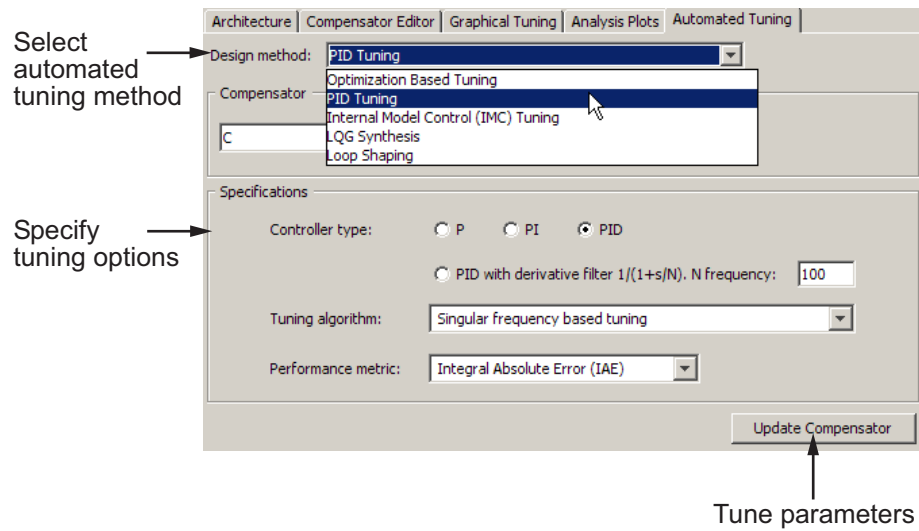
5 Evaluate the system's closed-loop performance.

a Plot the system response.

i In the Analysis Plots tab, select a plot type.

ii Select the type of response for each plot.
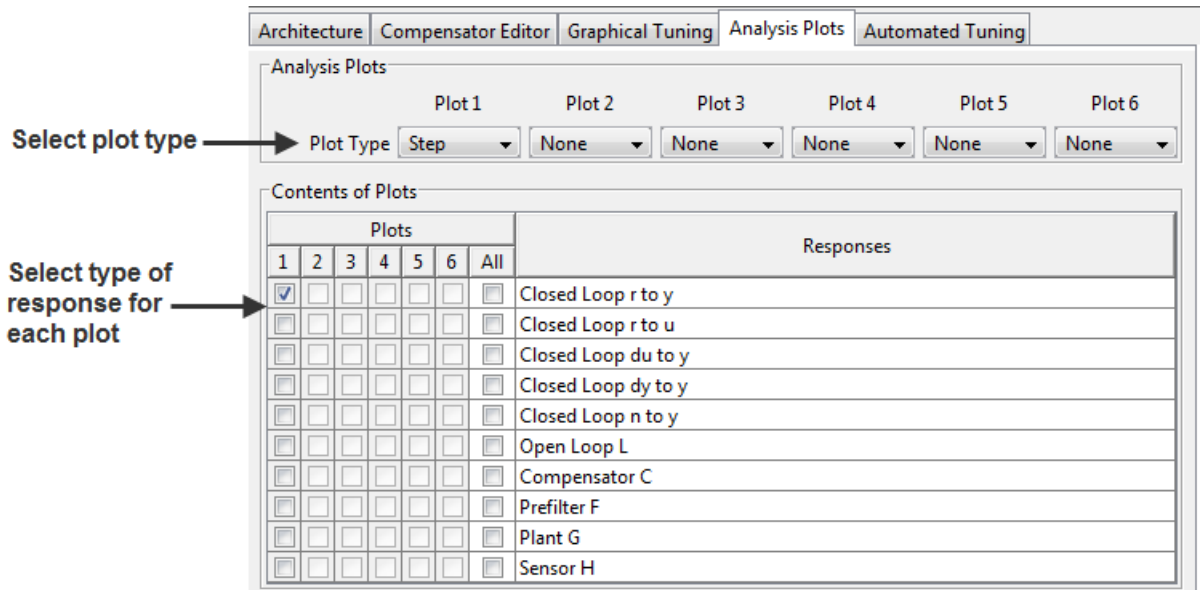
b Display specific performance characteristic for your system. Compare values to design requirements.

  i Right-click to select performance characteristics.

  ii Click on the dot that appears to view the characteristic value.

Click dot
to view value

Right-click
to select
performance
characteristics
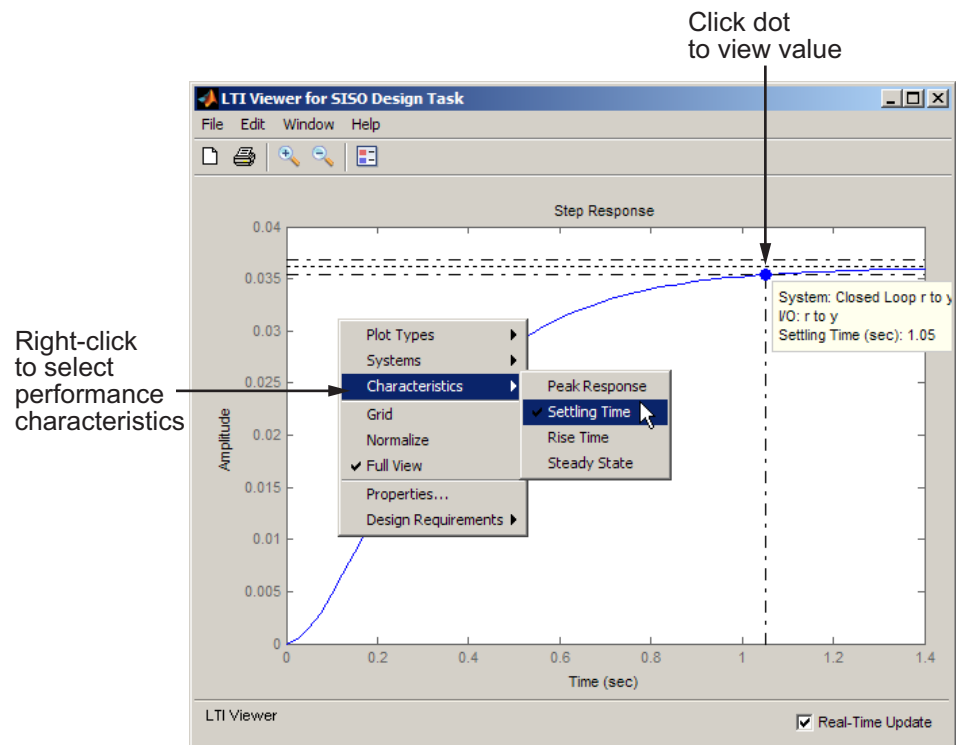
6 Refine compensator design using graphical tuning.

    a Create plots for graphical tuning.

        i In the Graphical Tuning tab, select the loop to tune in each plot.

        ii Select plot types.

Select the loop to
tune in each plot

Select plot types
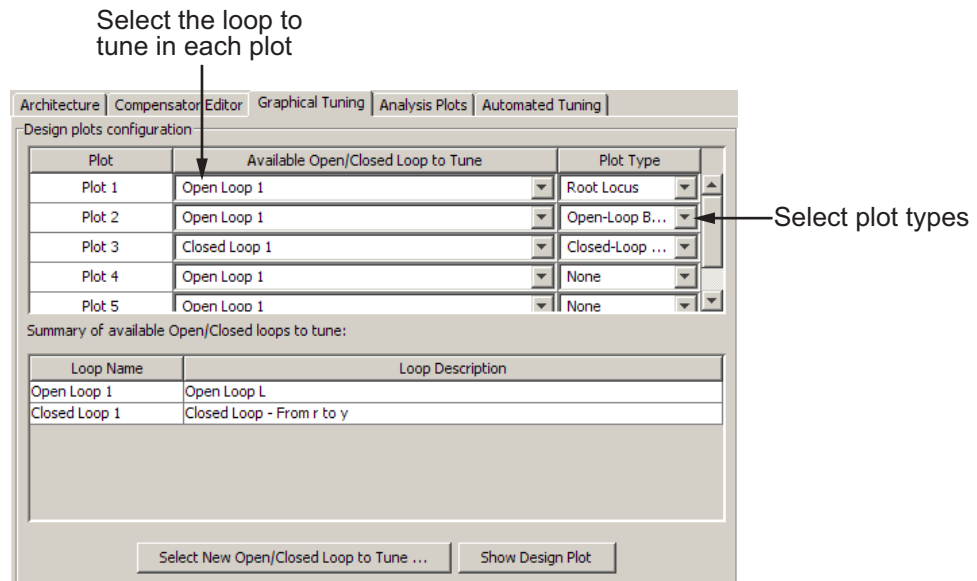
b In a design plot, modify the compensator by adding poles, zeros, lead, and lag, for example. To do this, right-click to add dynamic elements in the controller structure.

Right-click to add dynamic elements in controller structure
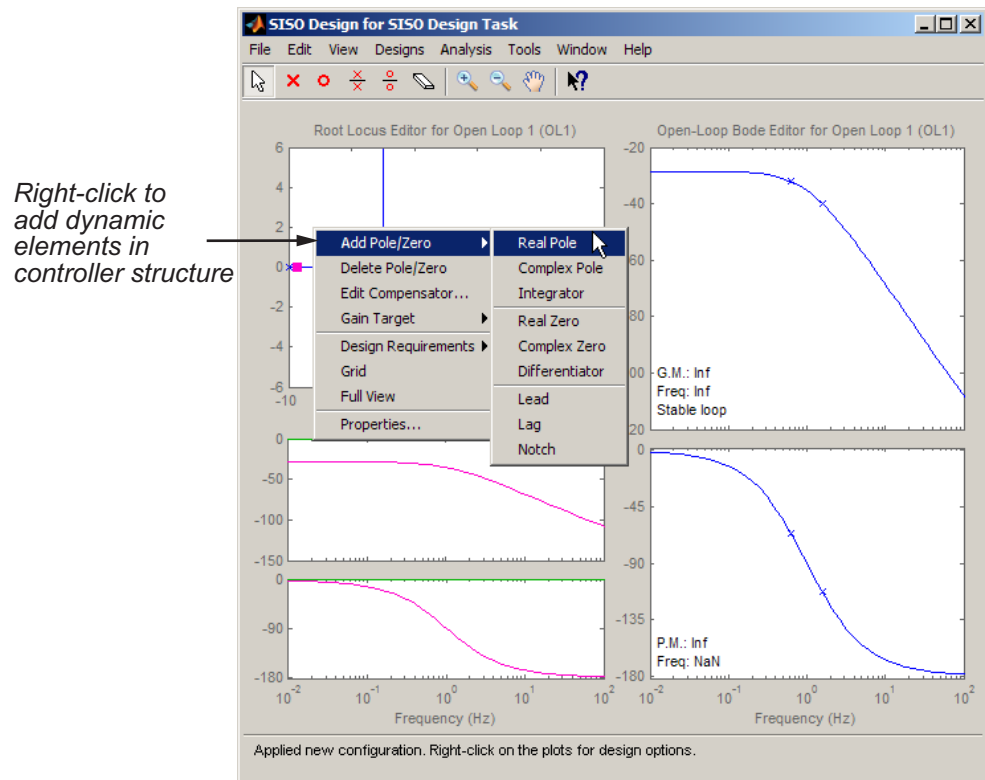
c Add a new pole, for example, by clicking the pole location on the plot.

d Modify the compensator structure by moving the poles, zeros, and other dynamic elements directly on the plot.

Click and drag plot
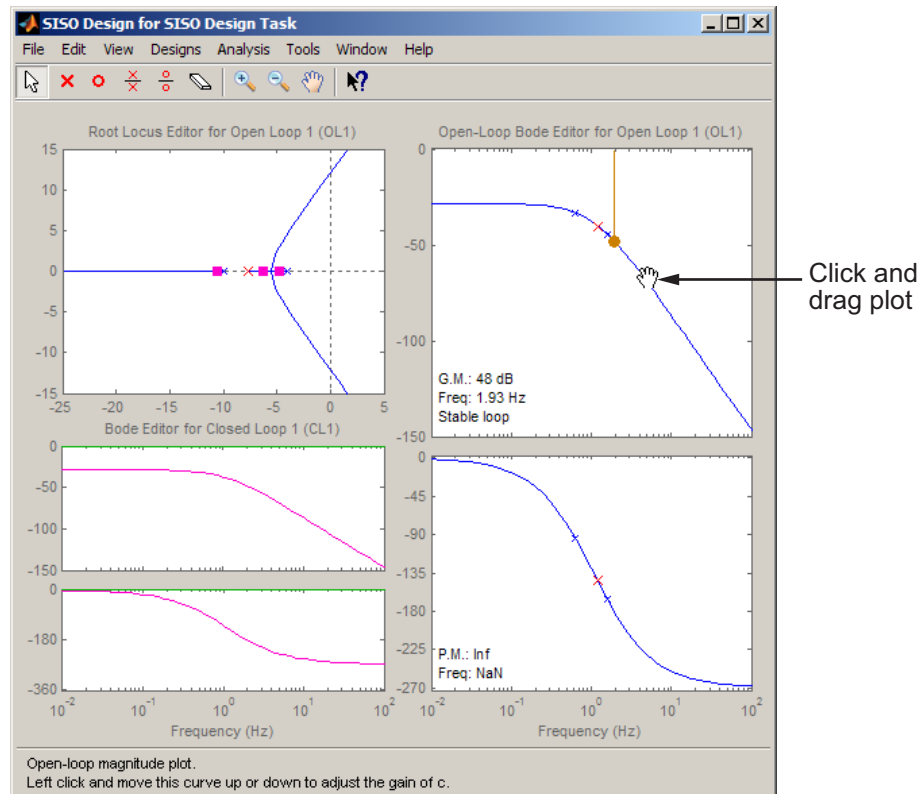
7 Export the compensator to the MATLAB workspace for further analysis.

a In the Control and Estimation Tools Manager, select **File** > **Export**.

b In the SISO Tool Export dialog box, select the compensator.

c Click **Export to Workspace**.

Components of the SISO Tool

The SISO Design Tool has the following components:

- **SISO Design Task** node in the Control and Estimation Tools Manager is a user interface (UI) that facilitates the design of compensators for single-input, single-output feedback loops through a series of interactive tabs.

- The Graphical Tuning window is a graphical user interface (GUI) for displaying and manipulating the Bode, root locus, and Nichols plots for the controller currently being designed. This window is titled SISO Design for *Design Name*.
- The LTI Viewer associated with the **SISO Design Task**. For instructions on how to operate the LTI Viewer, see "LTI Viewer for SISO Design Task Design Requirements".

Design Options in the SISO Tool

The SISO Design Tool facilitates the design of compensators for single-input, single-output feedback loops, lets you iterate on your designs and perform the following tasks:

- Manipulate closed-loop dynamics using root locus techniques.
- Shape open-loop Bode responses.
- "Add compensator poles and zeros, lead/lag networks and notch filters".
- "Add and tune lead/lag networks and notch filters".
- Inspect closed-loop responses using the LTI Viewer.
- "Adjust phase and gain margins".
- Perform multi-loop control design.
- Perform control design analysis for multiple models.
- "Convert models between discrete and continuous time".
- Automate compensator design.

Opening the SISO Design Tool

You can open the SISO Design Tool using an LTI model or an "array of LTI models". This topic shows how to open the SISO Design Tool with the DC motor example.

If you have not built the DC motor model, type

```
load ltiexamples
```

at the MATLAB prompt. This loads a collection of linear models, including the DC motor. To open the SISO Design Tool and import the DC motor, type

```
sisotool(sys_dc)
```

This command opens:

- The **SISO Design Task** node in the Control and Estimation Tools Manager
- The Graphical Tuning window with the root locus and open-loop Bode diagrams for the DC motor plotted by default
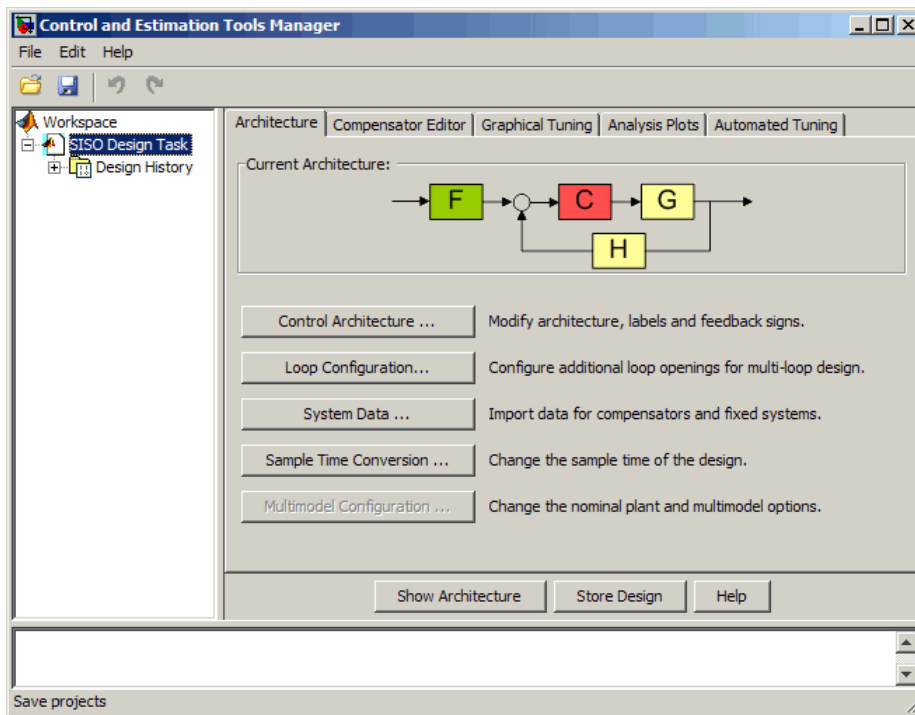
SISO Design Task Node (Architecture Tab)

SISO Design for SISO Design Task

File   Edit   View   Designs   Analysis   Tools   Window   Help

Root Locus Editor for Open-Loop 1 (OL1)

Open-Loop Bode Editor for Open-Loop 1 (OL1)

Right-click in any of these regions to see design options.

G.M.: Inf
Freq: Inf
Stable loop

This status bar shows useful tips about how to use this window and information about the status of your design.

P.M.: Inf
Freq: NaN

Real Axis

Frequency (rad/sec)

Right-click on the plots for more design options.

Graphical Tuning Window with the DC Motor Example

Using the SISO Design Task Node in the Control and Estimation Tools Manager

The **SISO Design Task** node in the Control and Estimation Tools Manager contains the following tabs for specifying controller design and behavior:

- "**Architecture**":

  - Change the feedback structure and label signals and blocks.

  - Configure loops for multi-loop design by opening signals to remove the effects of other feedback loops.

  - Import LTI models or row or column arrays of LTI models into your system.

- Convert sample time of the system or switch between different sample times to design different compensators.
  - Specify nominal model and frequency grid for multimodel computations.
- "**Compensator Editor**":
  - Directly edit compensator poles, zeros, and gains.
  - Add or remove compensator poles and zeros.
- "**Graphical Tuning**":
  - Configure design plots in the Graphical Tuning window.
  - Use design plots to graphically manipulate system response.
- "**Analysis Plots**":
  - Configure analysis plots in the LTI Viewer.
  - Use analysis plots to view the response of open- or closed-loop systems.
- "**Automated Tuning**":
  - Automatically generate compensators using optimization-based, PID, internal model control (IMC), linear-quadratic-Gaussian (LQG), or loop shaping methods.
  - Use optimization-based methods that automatically tune the system to satisfy design requirements (available when you have the Simulink Design Optimization™ product).

Importing Models into the SISO Design Tool

If you type `sisotool` at the MATLAB prompt, the Control and Estimation Tools Manager opens with the **SISO Design Task** node and an empty Graphical Tuning window.

You can import an LTI model or a row or column "array of LTI models" by clicking **System Data** in the **Architecture** page, as shown in the following figure.
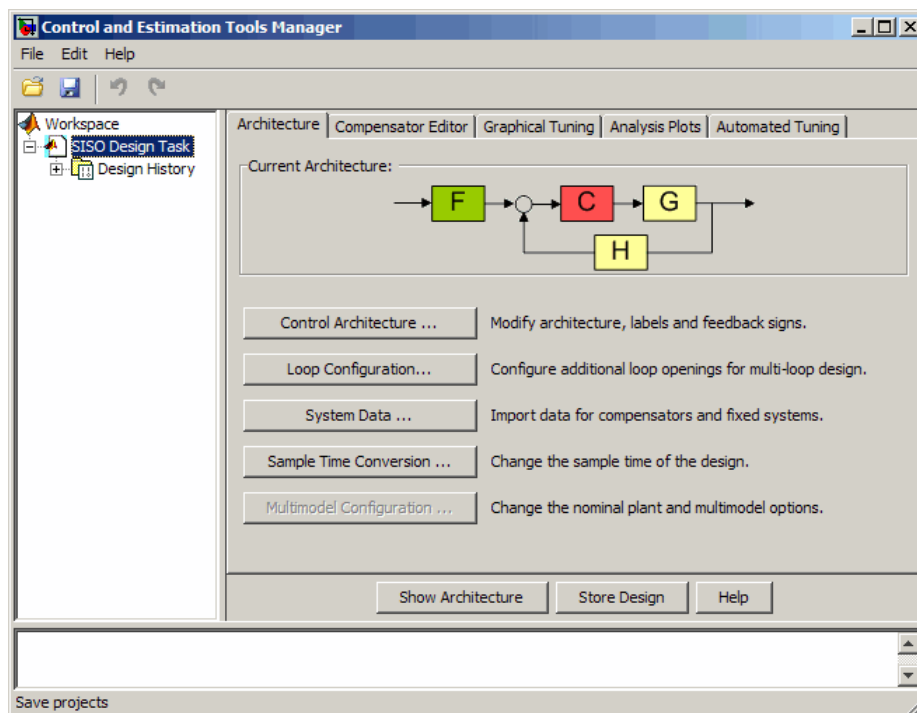
The System Data dialog box opens, as shown in the following figure.
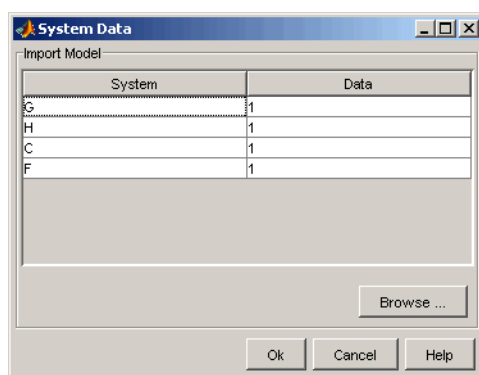
- **G** is a plant modeled as an LTI model or multiple plants modeled as a row or column array of LTI models. If **H** is also an array, their sizes must match.

- **H** is the sensor modeled as an LTI model or multiple sensors modeled as a row or column array of LTI models. If **G** is also an array, their sizes must match.

- **C** is a controller and is an LTI model.

- **F** is a prefilter and is an LTI model.

The default values of **G**, **H**, **C**, and **F** are 1. See "Feedback Structure" on page 4-84 for more information.

Note: If you import an array of LTI models for either **G** or **H** or both for control design, you can use the default multimodel options or configure the options by clicking **Multimodel Configuration** in the **Architecture tab**. For more information, see "Control Design Analysis of Multiple Models" on page 4-177.

To import the DC motor model:

1 Select **G** and click **Browse**. The Model Import dialog box opens, as shown in the following figure.
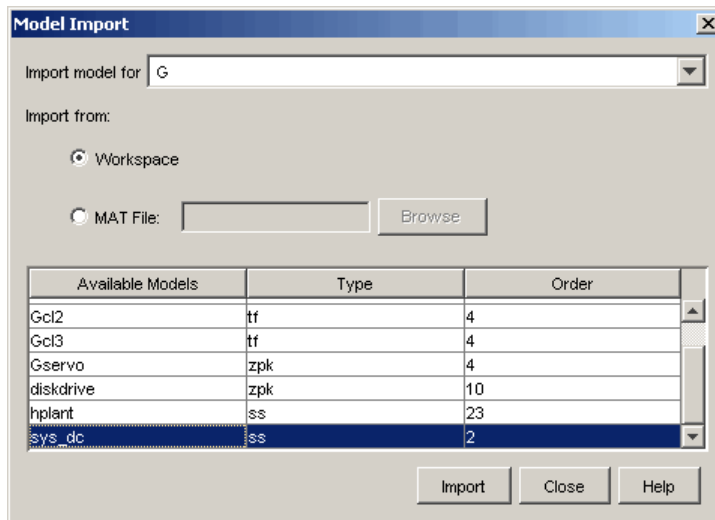
2 Select `sys_dc` from the **Available Models** list. Click **Import**, and then click **Close**.

You can now see `sys_dc` in the System Data dialog box.
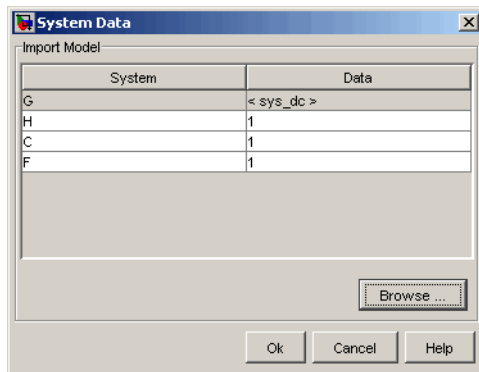
3 Click **OK**.

The Graphical Tuning window updates with the DC motor model, as shown in the following figure.
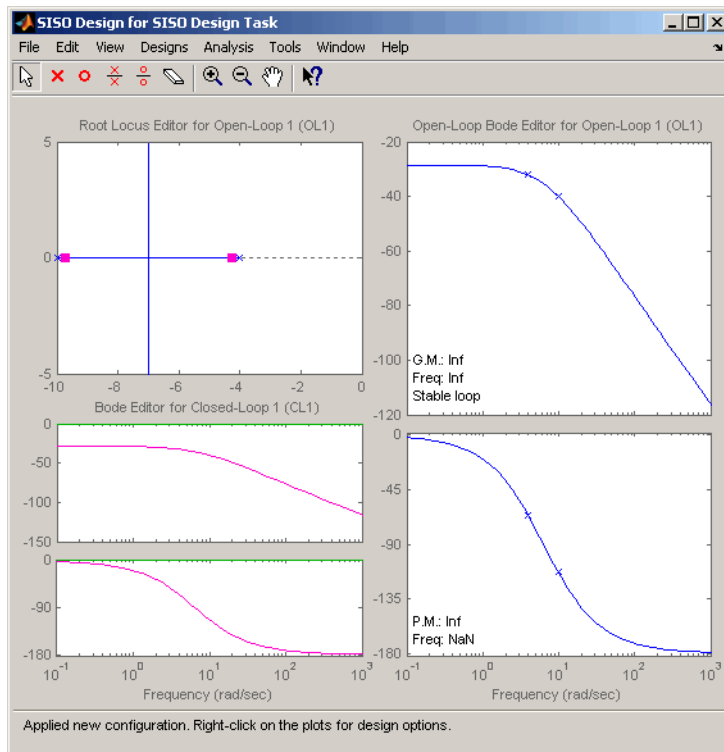
Feedback Structure

The SISO Design Tool, by default, assumes that the compensator is in the forward path, i.e., that the feedback structure looks like this figure.

Default Feedback Structure — Compensator in the Forward Path

In this figure:

- **G** represents the plant
- **H** represents the sensor dynamics
- **F** represents the prefilter
- **C** represents the compensator

The default values for **F**, **H**, and **C** are all 1. You can see the default values in the System Data dialog box. This means that, by default, the compensator has unity gain.

For the DC motor, **G** contains the DC motor model, `sys_dc`.

Alternative Feedback Structures

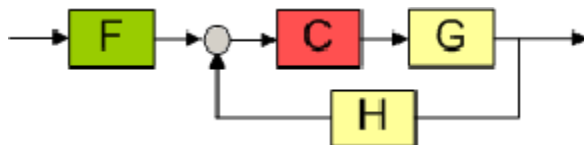In the **Architecture** page, click **Control Architecture** to open the Control Architecture dialog box.
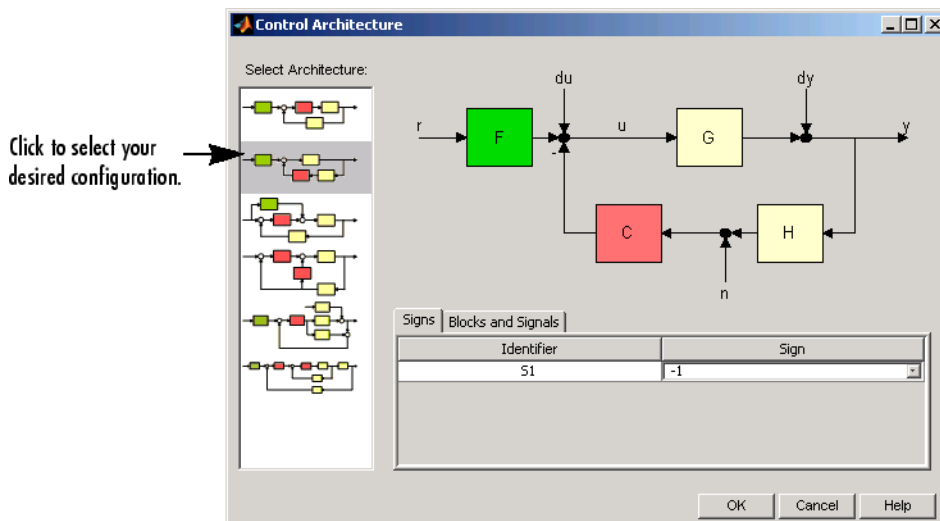
You can use the **Signs** and **Blocks and Signals** panes to change the sign of the feedback signal into a summing junction and rename blocks and signals in the diagram respectively. See "Block Diagram Structure Modifications" for more details.

On any tab in the **SISO Design Task** node, click **Show Architecture** to see the current architecture and a list of the identifiers and names associated with the components.
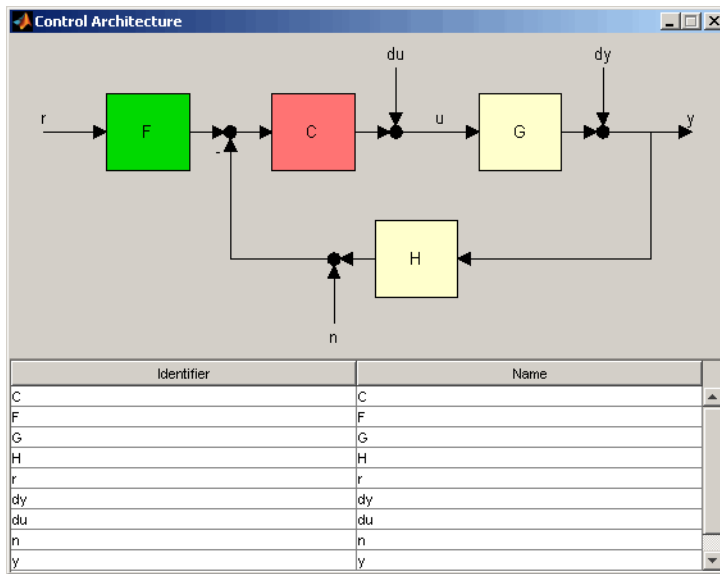
Analysis Plots for Loop Responses

As you design different controllers, you examine the various loop responses for a particular design. For example, to view the closed-loop step response, click the **Analysis Plots** tab. This tab lists the available responses, as shown in the following figure.
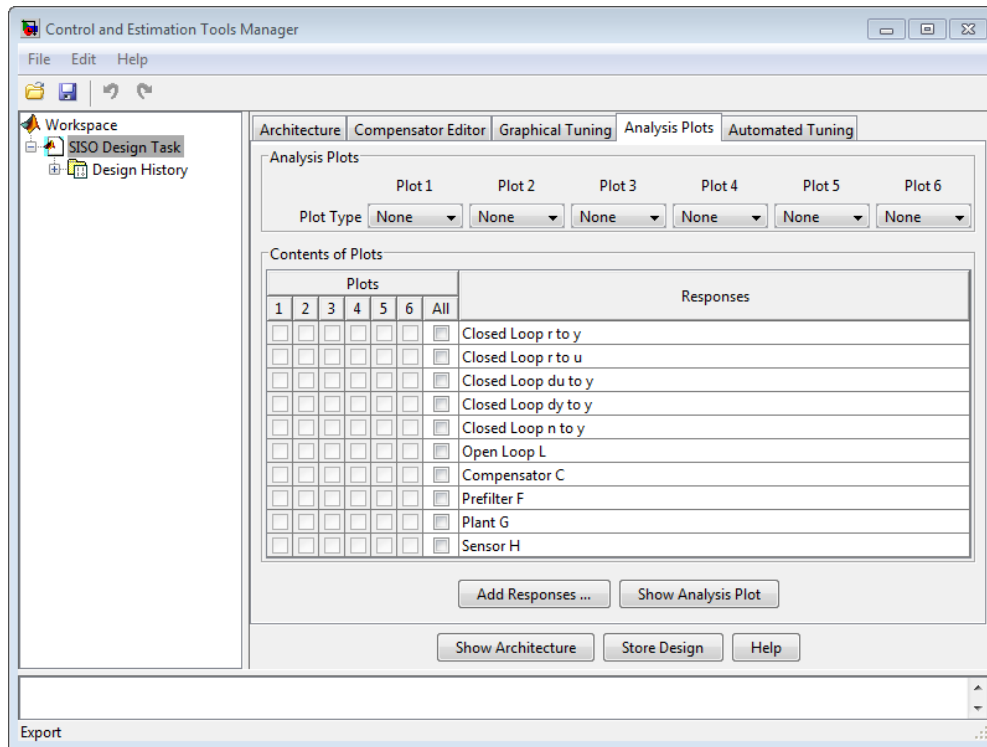
Select the plot types for each plot in the **Analysis Plots** area. Then select the plots to appear in the **Plots** list in the **Contents of Plots** table, as shown in the following figure.
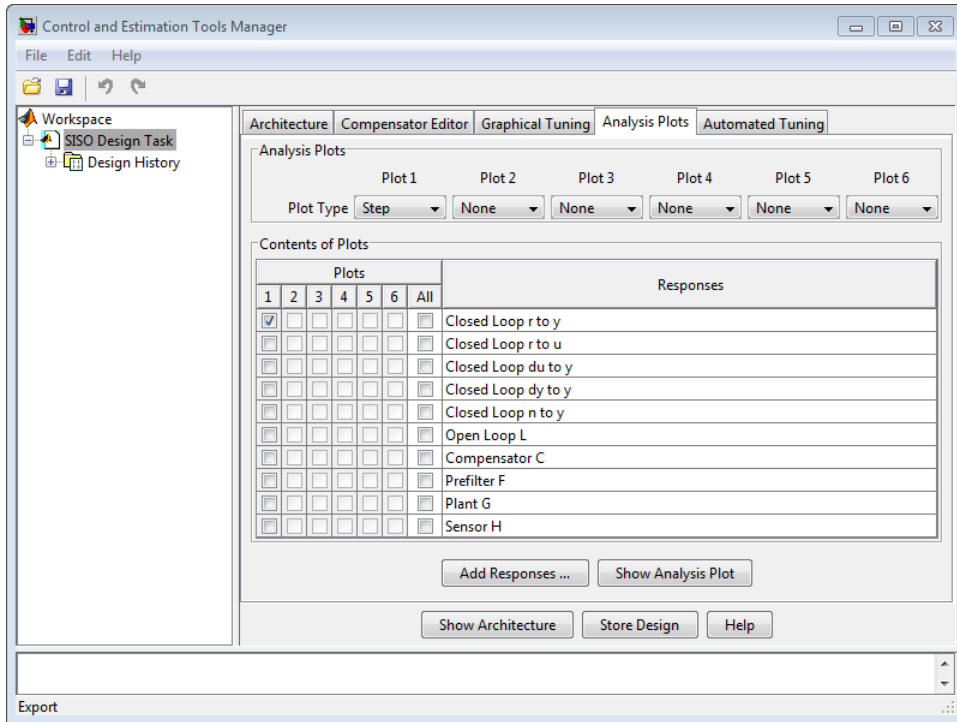
Analysis Plots Loop Response Selection

After you select a plot, the LTI Viewer opens and displays the appropriate response(s) opens. You can also click **Show Analysis Plot** to open the LTI Viewer.

The following figure shows the resulting plot for the closed-loop step response of the DC motor.
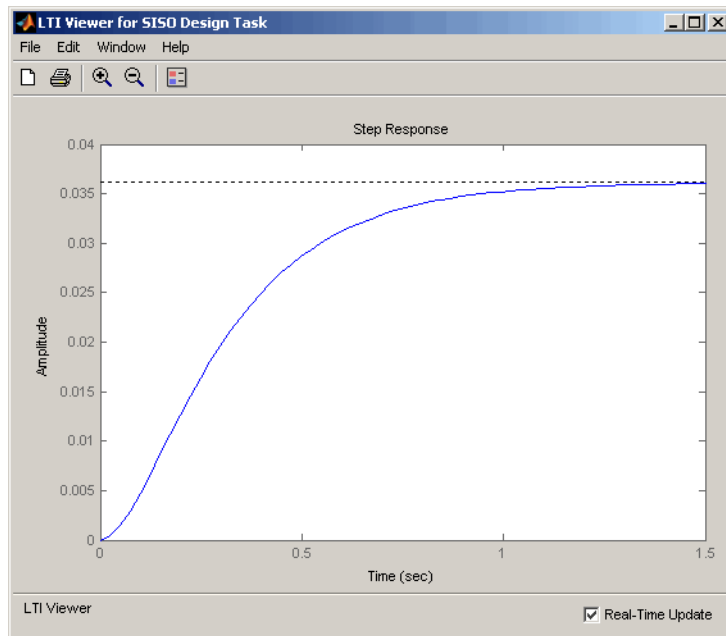
LTI Viewer Showing the Step Response for the DC Motor

The settling time of the DC motor is about 1.5 seconds, which is too slow for many applications. Also, the plot shows a large steady-state error. You can use Bode diagram techniques to improve the response time and steady-state error of the DC motor step response, as described in "Bode Diagram Design" on page 4-103.

For a row or column array of LTI models, the analysis plots show the response of the nominal model only. To view the responses of the remaining models in the array, right-click the plot, and select one of the following options:

- **Multimodel Display** > **Bounds** — Displays an envelope encompassing all model responses.
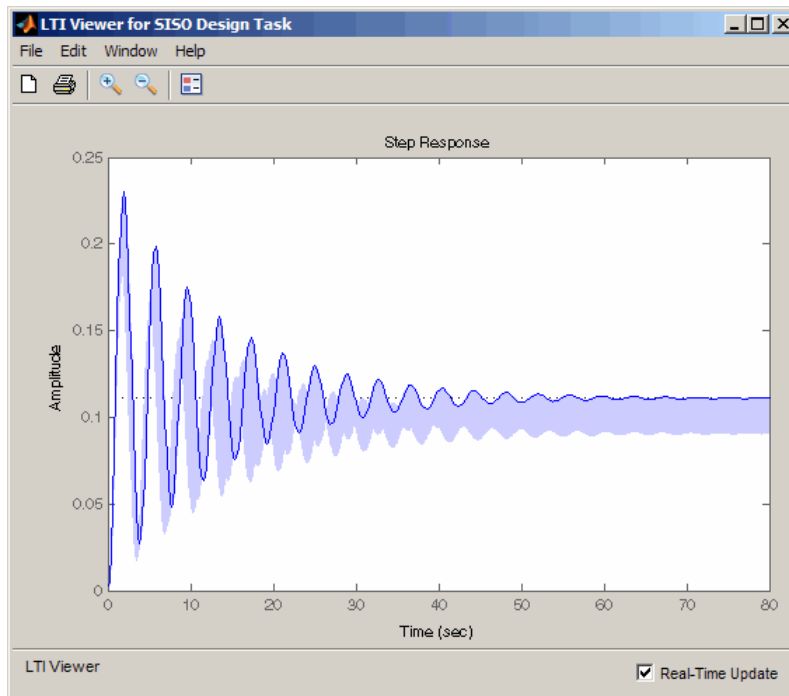
- **Multimodel Display > Individual Responses** — Displays individual model responses.
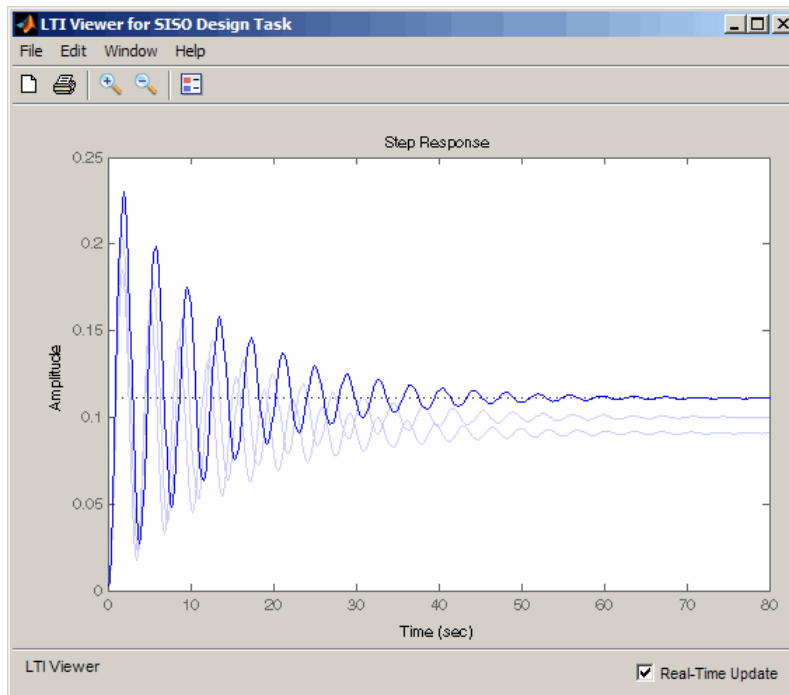
The plot line in darker shade of blue is the response of the nominal model. When you design a controller for the nominal plant, you can simultaneously analyze the controller's effect on the remaining model responses in the array. For more information on analyzing control design for multiple models, see "Control Design Analysis of Multiple Models" on page 4-177.

As you select different compensator designs, the software computes the responses and updates the response plots in real-time in the LTI Viewer associated with your **SISO Design Task**. For an array, this computation is expensive. Therefore, real-time updates may cause delay in refreshing the plots for:

- A large number of responses
- Responses of a large number of models

To deactivate real-time updates, unselect the **Real-Time Update** option.

Using the Graphical Tuning Window

The SISO Design for SISO Design Task graphical tuning window is a graphical user interface (GUI) that you use to display and manipulate the Bode, root locus, and Nichols plots for the controller currently being designed. You can accomplish most control design tasks using the tabs in the **SISO Design Task** node in the Control and Estimation Tools Manager.

The graphical tuning window shows the plots configured in the **Graphical Tuning tab**.
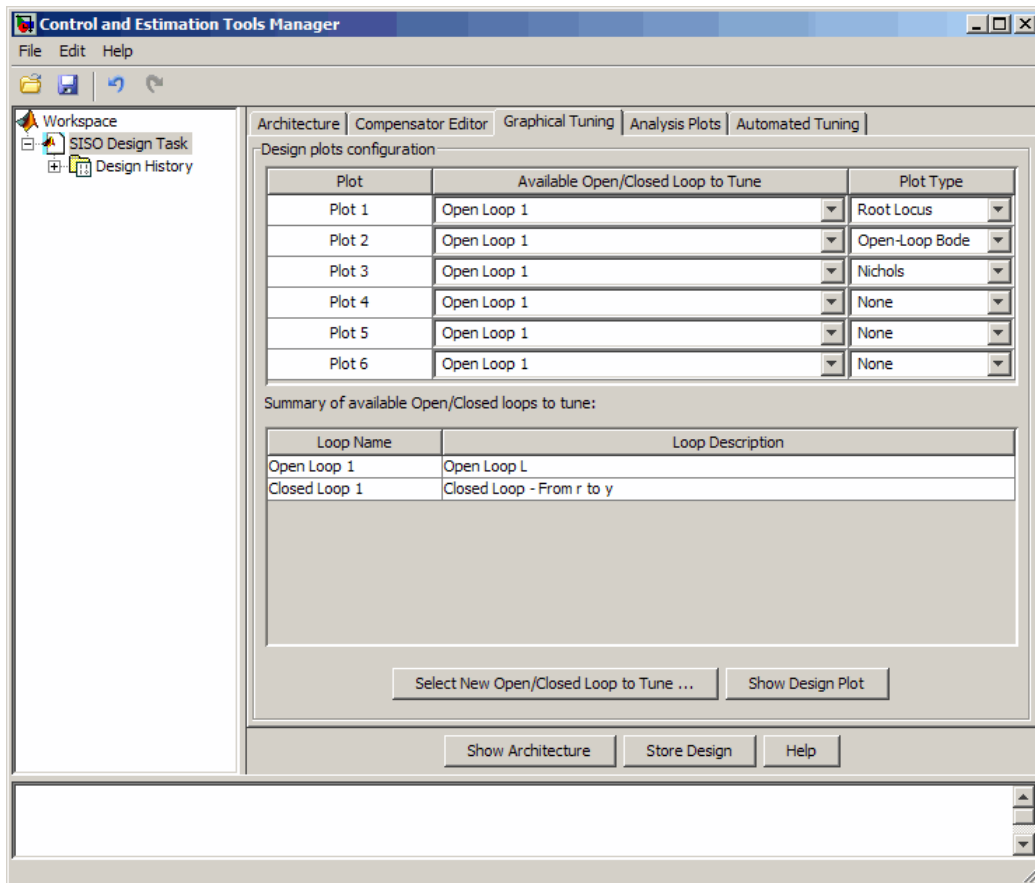
This topic describes some of the methods you can use to navigate in the Graphical Tuning window and manipulate the window's appearance.

Graphical Tuning Window Display

The Graphical Tuning window shows:

- Poles as x's
- Zeros as o's
- Gain and phase margins (by default) in the lower-left corners of the Bode magnitude and phase plots

For the DC motor, the graphical tuning window resembles the following figure.
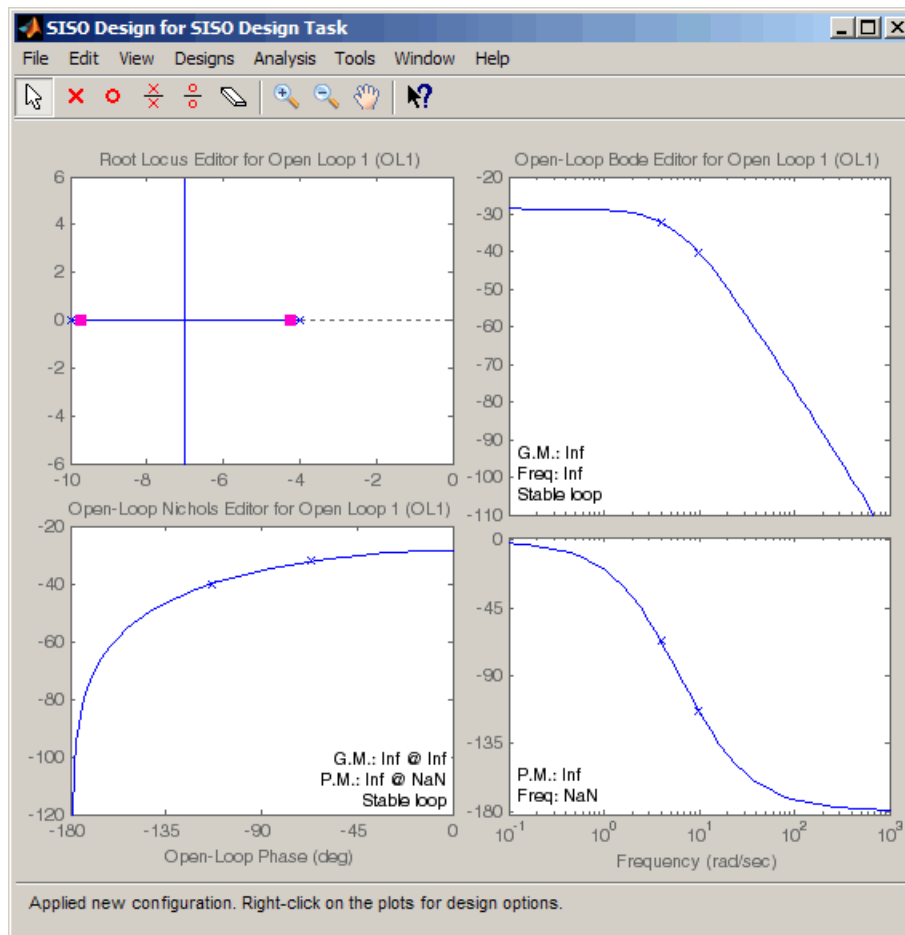
For a row or column arrays of LTI models, the plots show the individual responses and poles and zeros of all models in the array, as shown in the following figure.
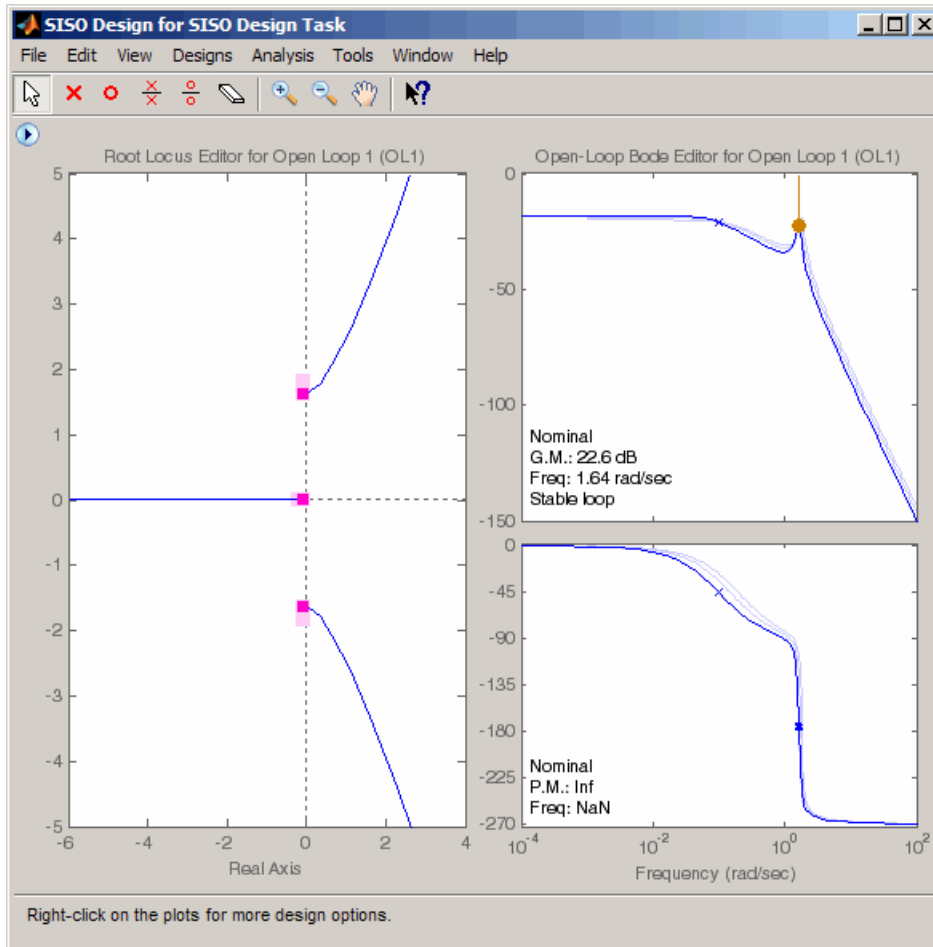
The plot line in darker shade of blue represents the response of the nominal model. This plot also displays the response characteristics of the nominal model.

You can change the display to view an envelope that encompasses all individual model responses. To do so, right-click the plot, and select **Multimodel Display** > **Bounds**. The bounds resemble the following figure.
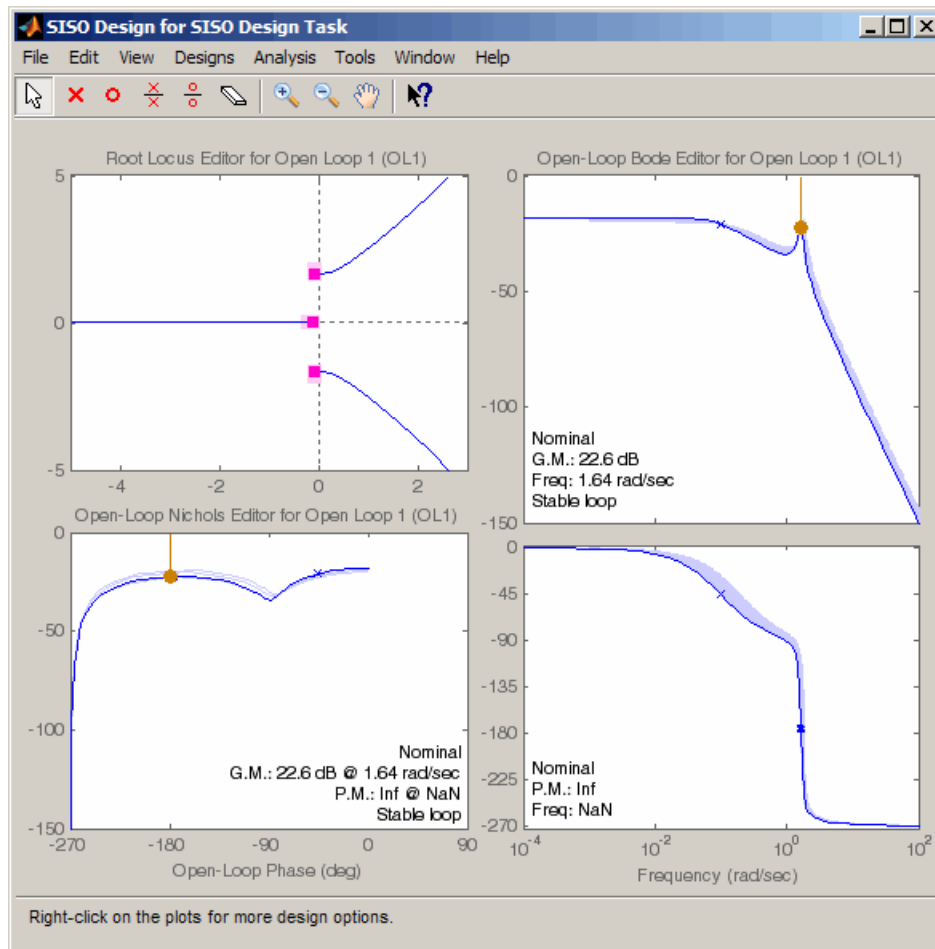
As you are designing a controller for the nominal plant, you can simultaneously analyze the controller's effect on the remaining models in the array. The software computes the response for each model and plots them. This computation is expensive and results in a slower refresh of the plots for:

• Large number of responses

• Responses for a large number of models

• Dense frequency grid

To speed up the plot updates, change the display to plot the envelope instead of individual responses. For more information, see "Control Design Analysis of Multiple Models" on page 4-177.

Changing Units on a Plot

The SISO Design Tool provides editors for setting plot options in the Graphical Tuning window. For example, if you want to change the frequency units on all the Bode plots created in the SISO Design Tool from rad/s to Hertz, select **SISO Tool Preferences** from the **Edit** menu in the **SISO Design Task** node on the Control and Estimation Tools Manager, as shown in the next figure.
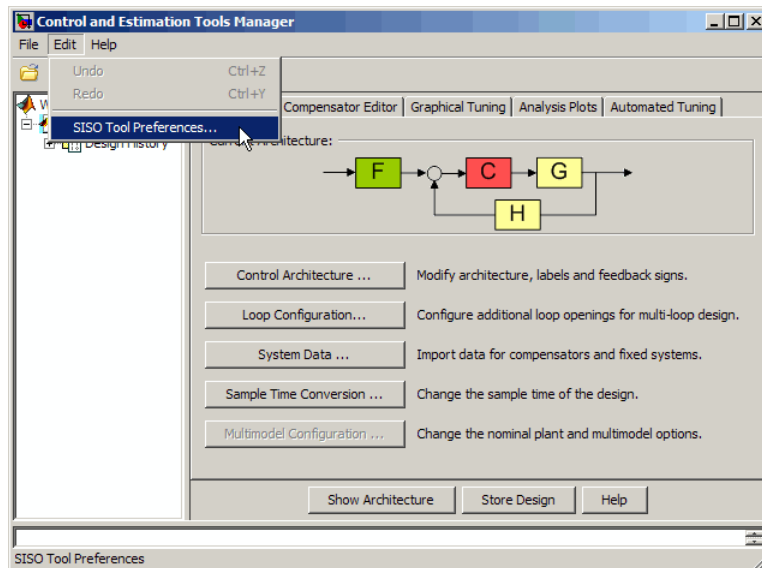
This opens the SISO Tool Preferences dialog box.
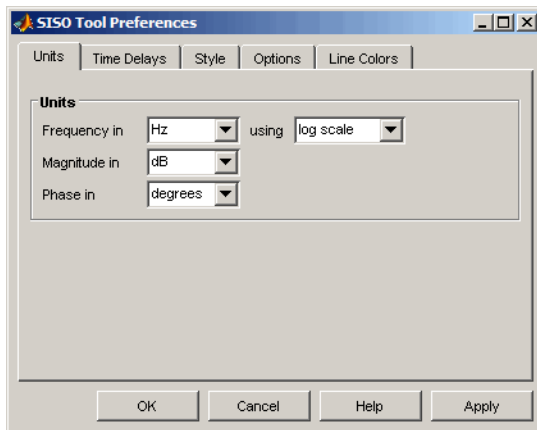
Use the options on the **Units** page to make the change. This unit change persists for the entire session.

Right-Click Menus

The SISO Design Tool has right-click menus available in any of the plot regions. Open the Bode magnitude menu by right-clicking in the Bode magnitude plot. The following menu appears.
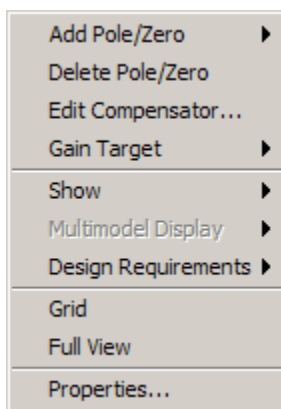
Right-Click Menu for the Bode Magnitude Plot

Although the menus for each plot generally contain the same options, there are some options specific to each plot type:

- The Open-Loop Bode Editor has a **Gain Target** option, the Closed-Loop Bode Editor has a **Select Compensator** option instead.
- For arrays of LTI models, the **Multimodel Display** option for the Root Locus Editor has an option to show the poles and zeros of the nominal or all models, instead of **Bounds** and **Individual Responses** options.

The right-click menus contain numerous features. The DC motor example uses many of the available features; for a complete discussion of the right-click menus, see the help for the "SISO Design Tool"."

Exporting the Compensator and Models

After you design the controller, you may want to save your design parameters for future implementation. You can do this by selecting:

- **File > Export** in the Control and Estimation Tools Manager
- **File > Export** in the Graphical Tuning window.

The SISO Tool Export dialog box opens:
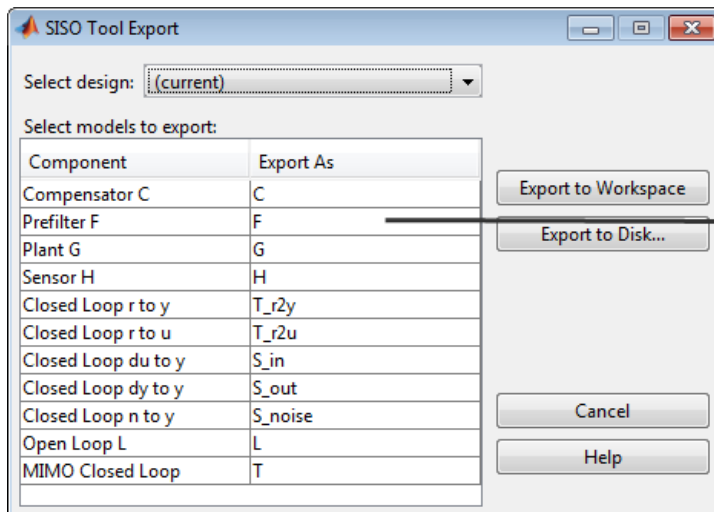
- **Select models to export** area shows a list of models for the components of your designs. For a plant or sensor modeled as row or column array of LTI models, the components are also arrays of LTI models.

- **Export As** column displays variables with default names or previously named in the System Data dialog box.
- **Select design** drop-down list lets you view the components of other designs.

To export your controller to the workspace:

1 Select `Compensator C` in the **Component** column. If you want to change the name, double-click in the cell for `Compensator C`.

   To select multiple components, use the **Shift** key if they are all adjacent and the **Ctrl** key if they are not.

2 Click **Export to Workspace** to export the compensator to the MATLAB workspace.

3 At the MATLAB prompt, type

   `who`
   to view the compensator variable `C`.

4 To see the format in which this variable is stored, type

   `C`

Clicking **Export to Disk** opens the Export to Disk dialog box.
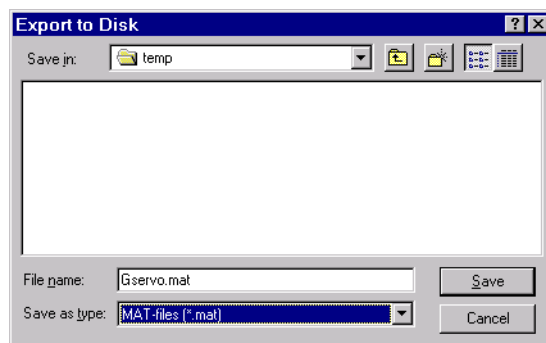
You can save your models as MAT-files in any folder. The default name for the MAT-file is the name of your original model. You can also specify a name for the MAT-file. If you save multiple components, they are stored in a single MAT-file.

Storing and Retrieving Intermediate Designs

You can store and retrieve intermediate compensators while you iterate on your compensator design. To store intermediate designs, click the **Design History** node or **Store Design**, both located on the **SISO Design Task** node in the Control and Estimation Tools Manager.

Alternatively, you can select **Store/Retrieve** from the **Designs** menu in the Graphical Tuning window. Using either method, the following **Design History** page opens.
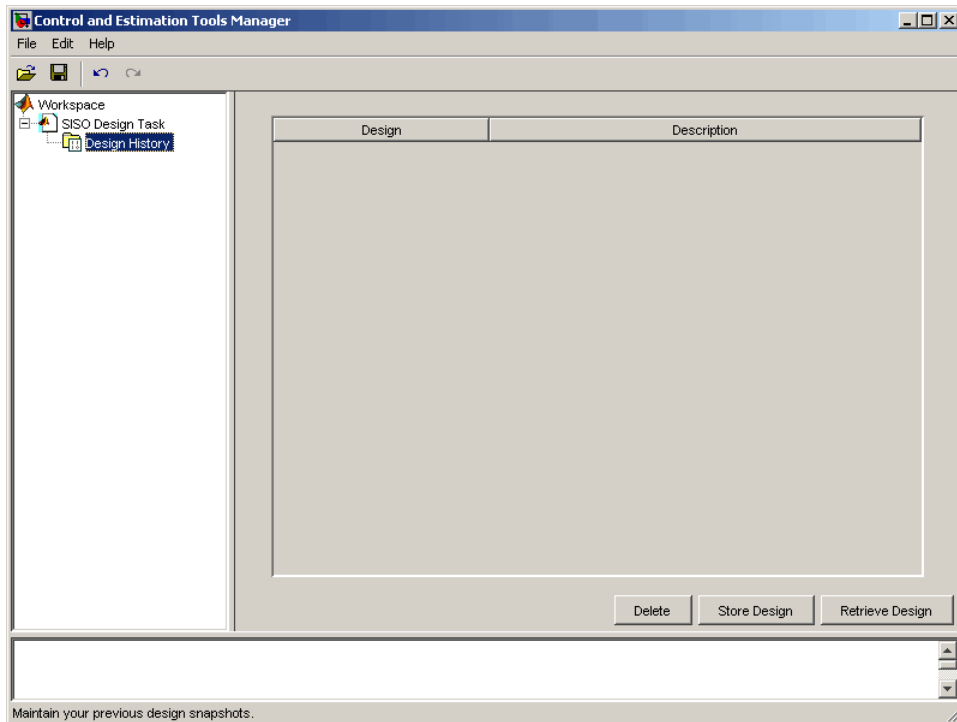
If you have any intermediate designs already stored, they will appear on the **Design History** page.

Click **Store Design** to save the current design with the default name `Design;` the suffix increments when you store additional compensators without renaming them. You can rename the design by right-clicking the name under the node and selecting **Rename**.

To retrieve intermediate designs, again click the **Design History** node or select **Store/Retrieve** from the **Designs** menu. From the **Design History** page, select the design to retrieve, and then click **Retrieve Design**, as shown next.
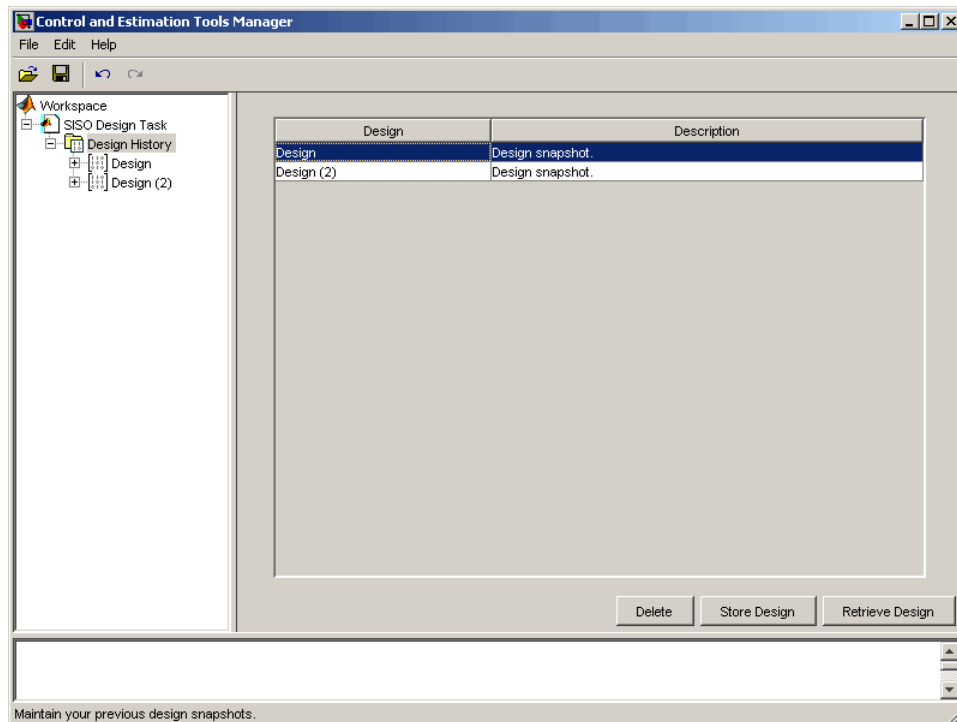
Design History Page Listing Current Designs

The Graphical Tuning window automatically reverts to the selected compensator design.

Click any design name in the **Design History** to view a snapshot summary of the design, as shown in the following figure.

Design Snapshot Summary

Return to the compensator list by clicking the **Design History** node.

You can delete an intermediate design by selecting it from the **Design History** page and clicking **Delete**.

Bode Diagram Design

What Is Bode Diagram Design?

One technique for compensator design is to work with Bode diagrams of the open-loop response (loop shaping).

Using Bode diagrams, you can

- Design to gain and phase margin specifications
- Adjust the bandwidth
- Add notch filters for disturbance rejection

Bode Diagram Design for DC Motor

The following topics use the DC motor example to show how to create a compensator using Bode diagram design techniques. From "SISO Example: The DC Motor" on page 2-3, the transfer function of the DC motor is

```
Transfer function:
        1.5
-----------------
s^2 + 14 s + 40.02
```

For this example, the design criteria are as follows:

- Rise time of less than 0.5 second
- Steady-state error of less than 5%
- Overshoot of less than 10%
- Gain margin greater than 20 dB
- Phase margin greater than 40 degrees

Adjusting the Compensator Gain

The LTI Viewer Showing the Step Response for the DC Motor, shows that the closed-loop step response is too slow. The simplest approach to speeding up the response is to increase the gain of the compensator.

To increase the gain:

1 Click the **Compensator Editor** tab to open the **Compensator Editor** page.

2 Select **C** from the compensator selection list.

3 In the text box to the right of the equal sign in the **Compensator** area, enter 38 and press **Enter**.
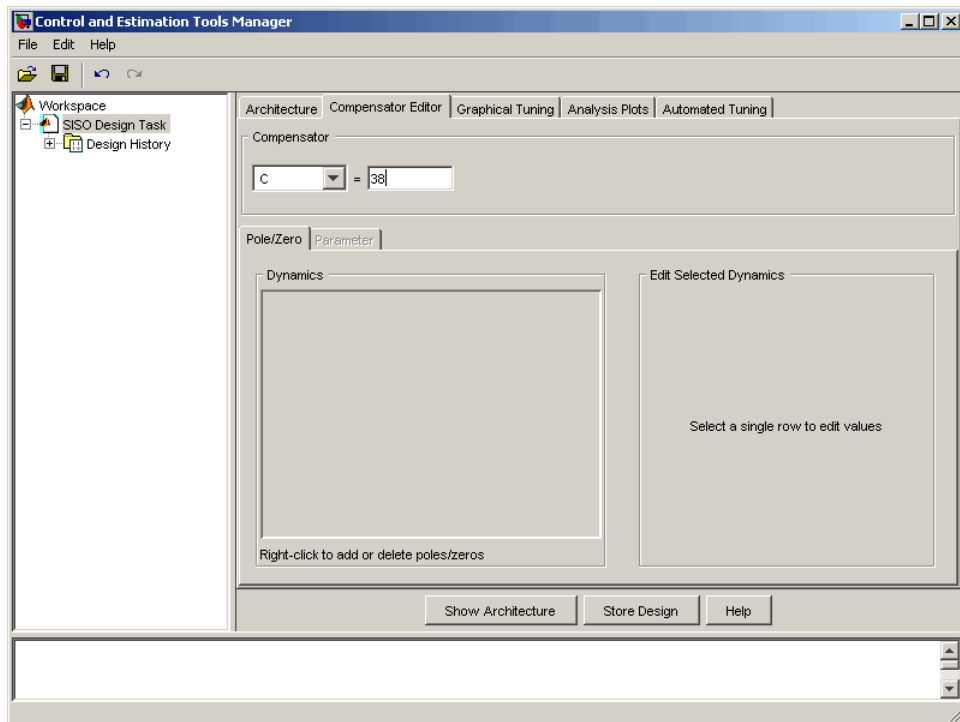
Adjusting Compensator Gain on the Compensator Editor Page

The SISO Design Tool calculates the compensator gain, and Bode and root locus graphs in the Graphical Tuning window are updated.

Alternatively, you can set the gain in the Graphical Tuning window by grabbing the Bode magnitude line and dragging it upward. The gain and poles change as the closed-loop set point is recomputed, and the new compensator value is updated in the **Compensator Editor** page.

Adjusting the Bandwidth

Because the design requirements include a 0.5-second rise time, try setting the gain so that the DC crossover frequency is about 3 rad/s. The rationale for setting the bandwidth to 3 rad/s is that, to a first-order approximation, this should correspond to about a 0.33-second time constant.

To make the crossover easier to see, select **Grid** from the right-click menu. This creates a grid for the Bode magnitude plot. Left-click the Bode magnitude plot and drag the curve until you see the curve crossing over the 0 dB line (on the *y* axis) at 3 rad/s. This changes both the SISO Design Tool display and the LTI Viewer step response.

For a crossover at 3 rad/s, the compensator gain should be about 38. By default, the Graphical Tuning window shows gain and phase margin information in the lower-left corners of the Bode diagrams. In the Bode magnitude plot, it also tells you if your closed-loop system is stable or unstable.

This figure shows the Graphical Tuning window.
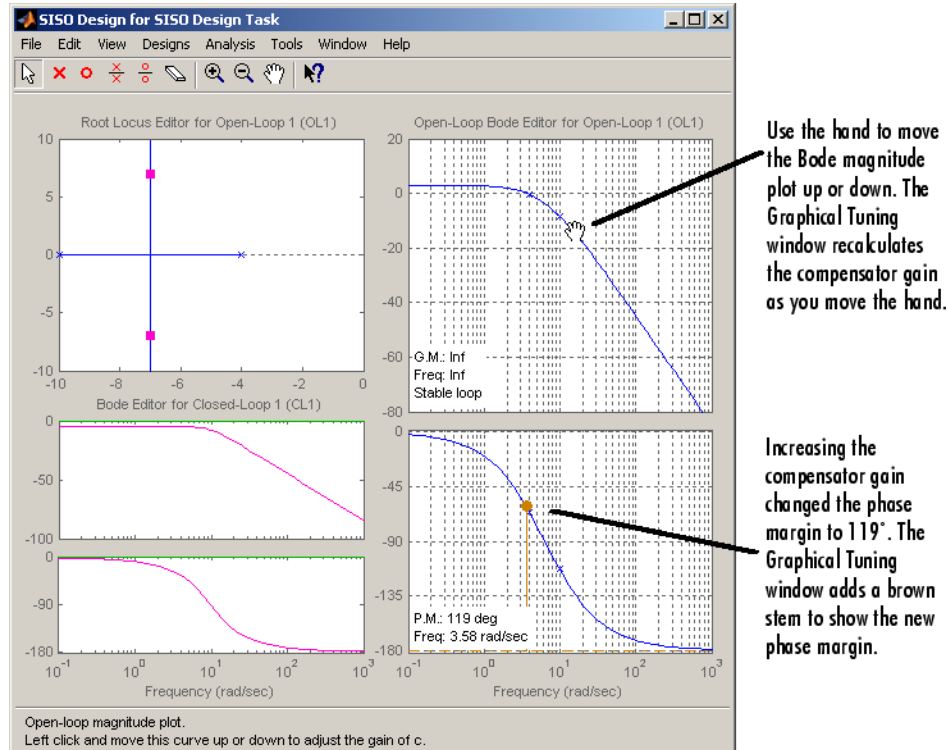
Adjusting Bandwidth in the Graphical Tuning Window

This plot shows the associated closed-loop step response in the LTI Viewer.
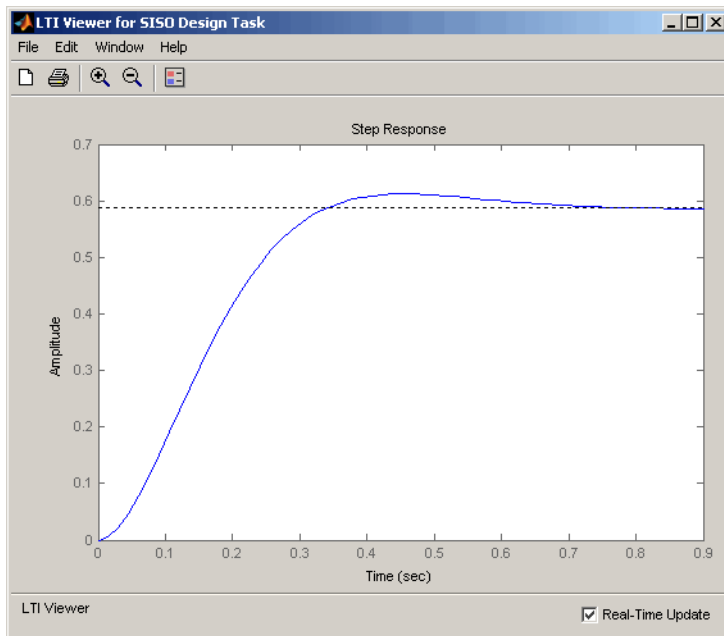
Closed-Loop Step Response for the DC Motor with a Compensator Gain = 38

The step response shows that the steady-state error and rise time have improved somewhat, but you must design a more sophisticated controller to meet all the design specifications, in particular, the steady-state error requirement.

Adding an Integrator

One way to eliminate steady-state error is to add an integrator. To add an integrator:

1 Click the **Compensator Editor** tab to open the **Compensator Editor** page.

2 Right-click anywhere in the **Dynamics** table for the right-click menu, and then select **Add Pole/Zero > Integrator**.

The following figures show this process.
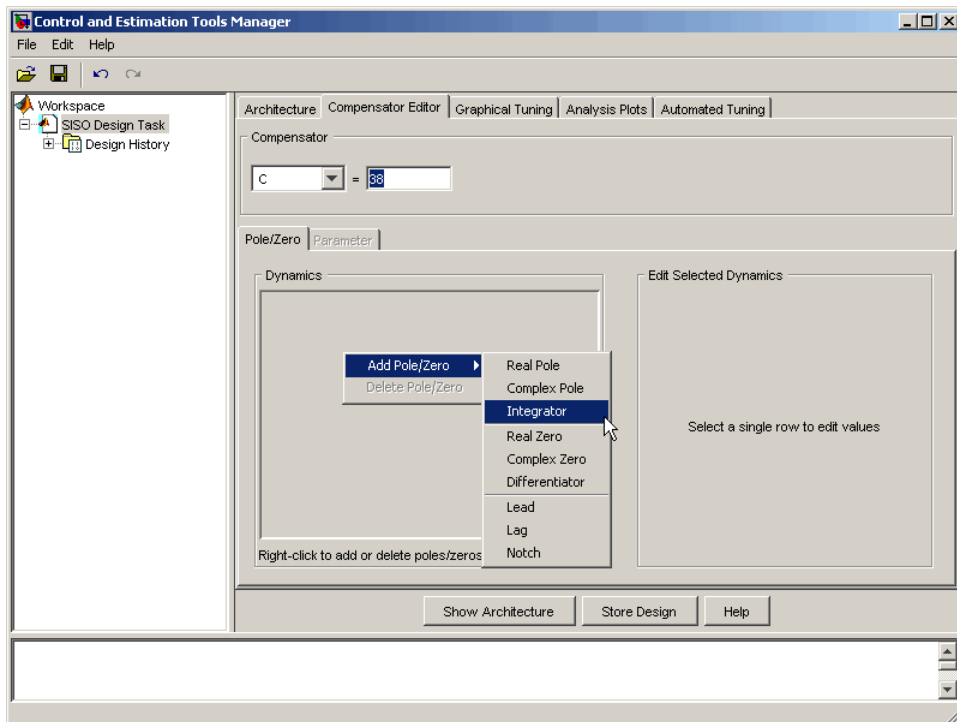
Adding an Integrator in the Dynamics Table
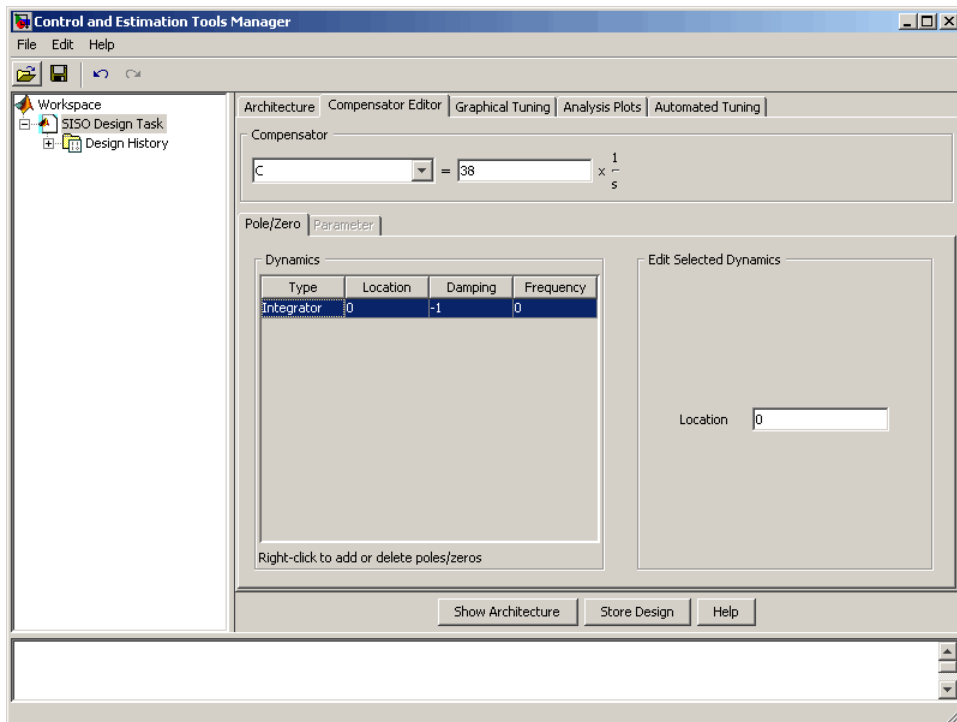
Editable Integrator Parameters

Notice adding the integrator changed the crossover frequency of the system. Readjust the compensator gain in the **Compensator Editor** page to bring the crossover back to 3 rad/ s; the gain should be 99.

After you have added the integrator and readjusted the compensator gain, the Graphical Tuning window shows a red `x' at the origin of the root locus plot.
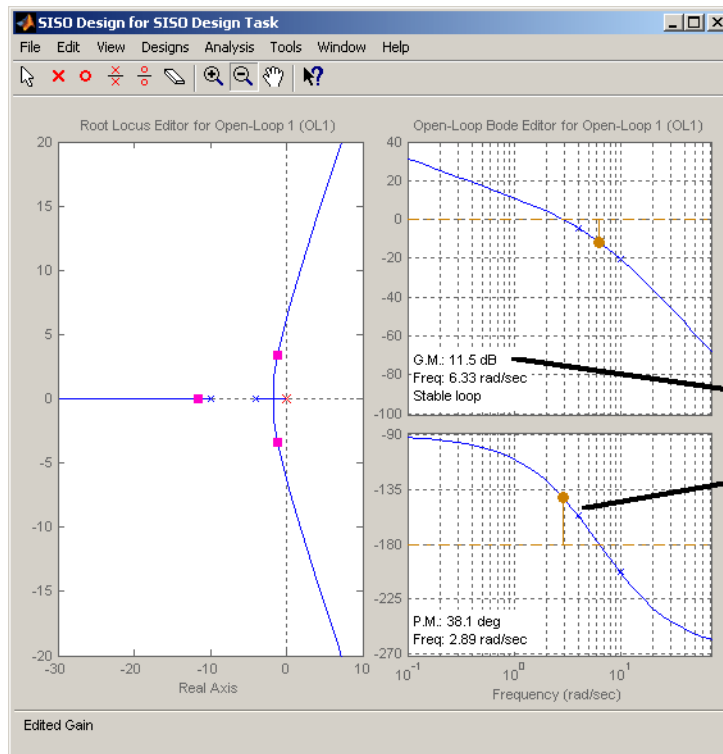
Integrator on the Root Locus Plot

The following figure shows the closed-loop step response.
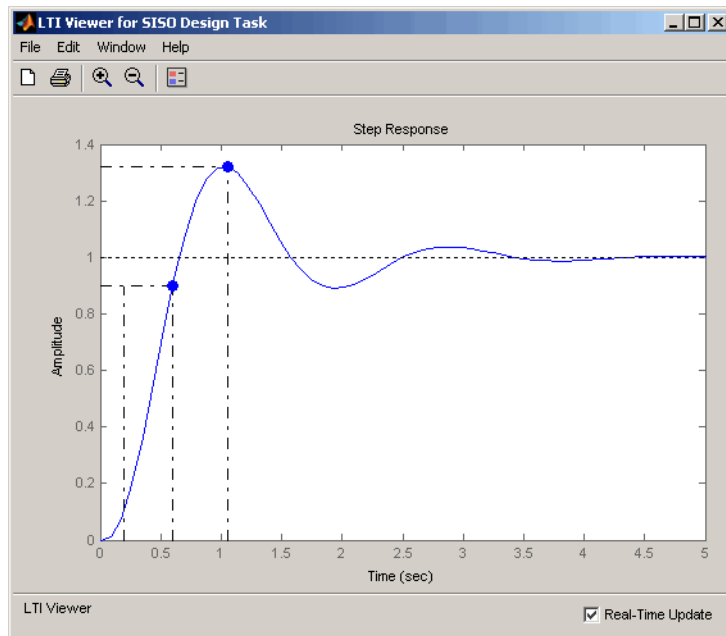
Step Response for the DC Motor with an Integrator in the Compensator

Use the right-click menu to show the peak response and rise time (listed under the Characteristics). The step response is settling around 1, which satisfies the steady-state error requirement. This is because the integrator forces the system to zero steady-state error. The figure shows, however, that the peak response is 1.3, or about 30% overshoot, and that the rise time is roughly 0.4 second. So a compensator consisting of an integrator and a gain is not enough to satisfy the design requirements, which require that the overshoot be less than 10%.

Adding a Lead Network

Part of the design requirements is a gain margin of 20 dB or greater and a phase margin of 40° or more. In the current compensator design, the gain margin is 11.5 dB and the phase margin is 38.1°, both of which fail to meet the design requirements. The rise time needs to be shortened while improving the stability margins. One approach is to increase the gain to speed up the response, but the system is already underdamped, and increasing the gain will decrease the stability margin as well. You might try

experimenting with the compensator gain to verify this. The only option left is to add dynamics to the compensator.

One possible solution is to add a lead network to the compensator. To add the lead network:

1   Click the **Compensator Editor** tab to open the **Compensator Editor** page.

2   In the **Dynamics** table, right-click and then select **Add Pole/Zero > Lead**.

The following figures show the process of adding a lead network to your controller.
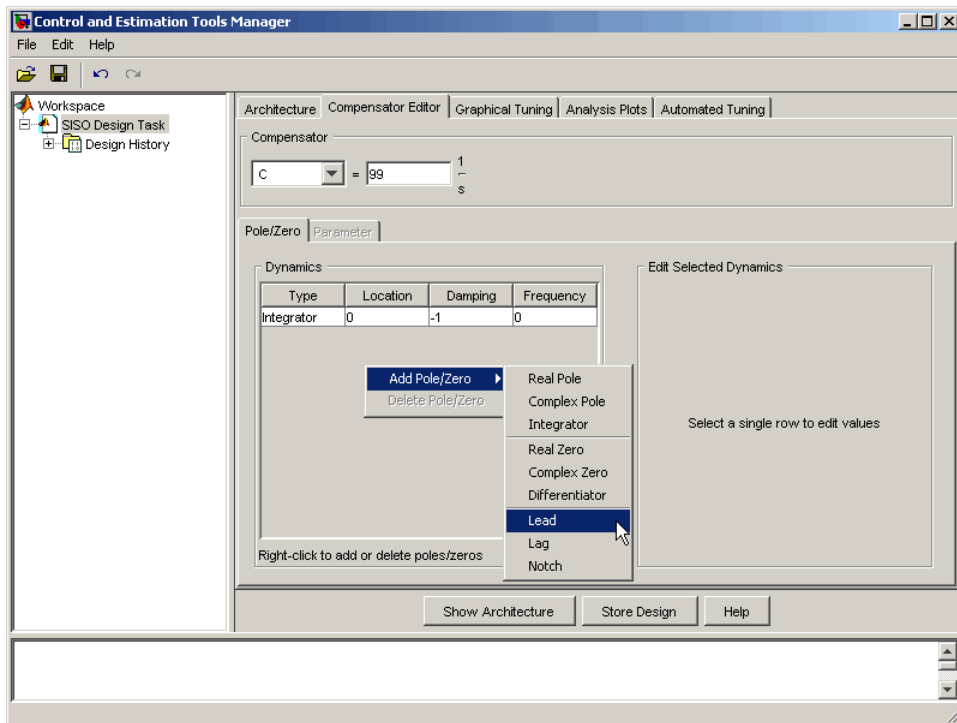
Adding a Lead Network to the DC Motor Compensator on the Compensator Editor Page
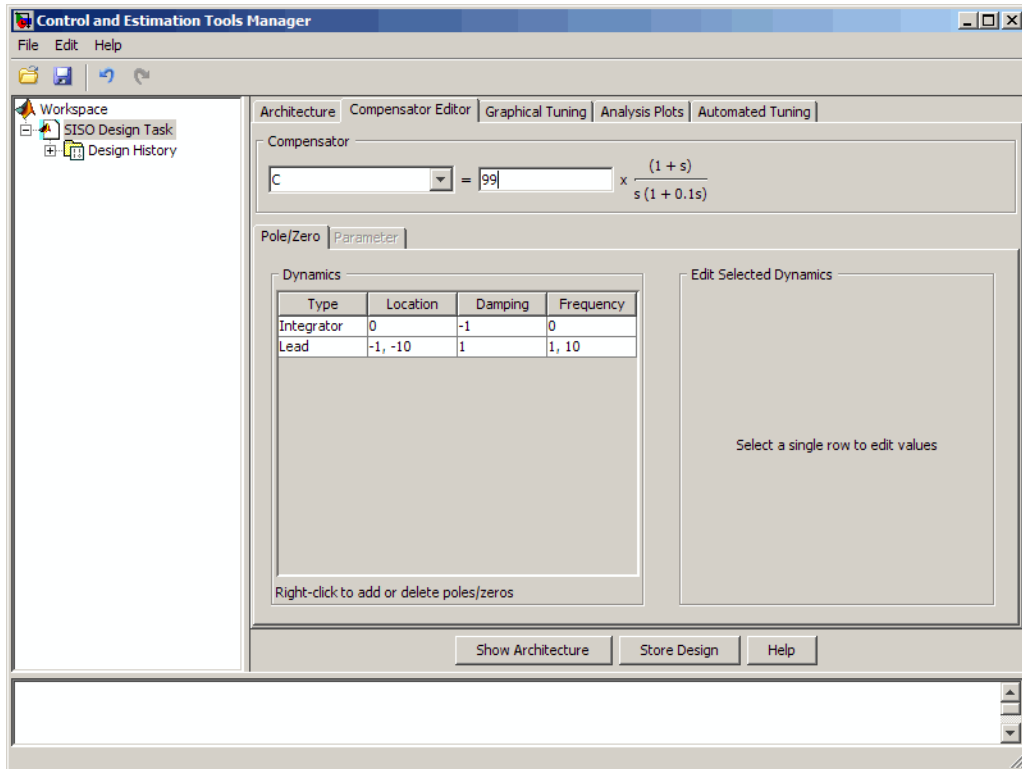
Lead Network Added

Editable fields are shown in the **Edit Selected Dynamics** group box (right side of page) when an item in the **Dynamics** table has been selected, as shown in the following figure.
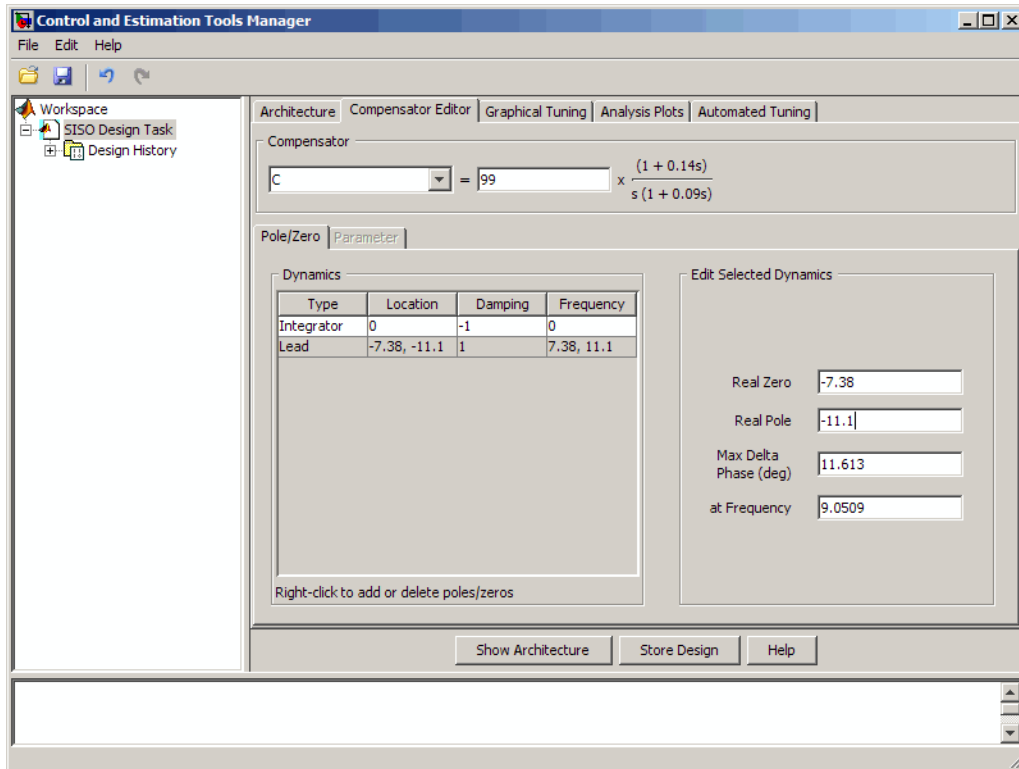
For this example, change Real Zero to -7.38 and change Real Pole to -11.1.

You can also add a lead network using the Graphical Tuning window. Right-click in the Bode graph, select **Add Pole/Zero > Lead**, place the `x' on the plot where you want to add the lead network, and then left-click to place it. The **Compensator Editor** page is updated to include the new lead network in the **Dynamics** table.

Your Graphical Tuning window and LTI Viewer plots should now look similar to these.
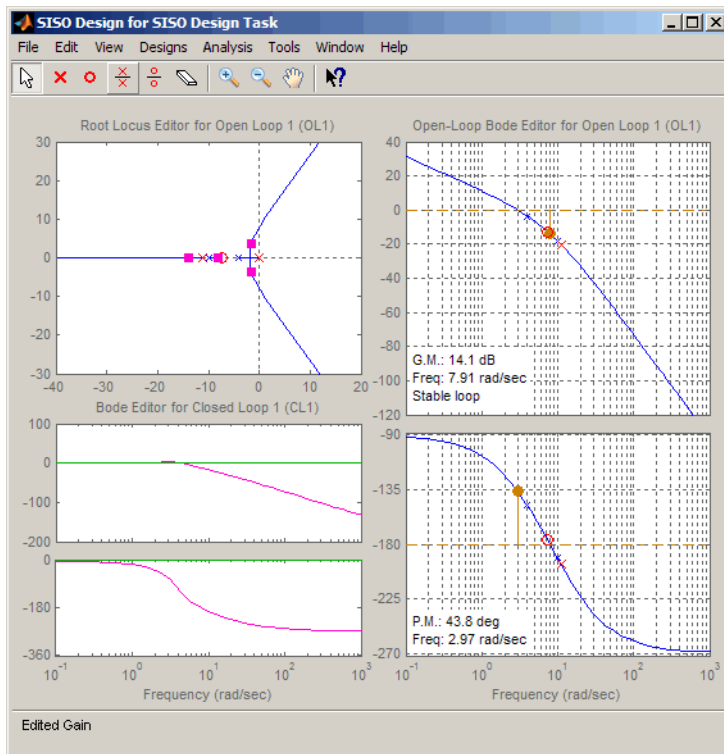
Root Locus, Bode, and Step Response Plots for the DC Motor with a Lead Network
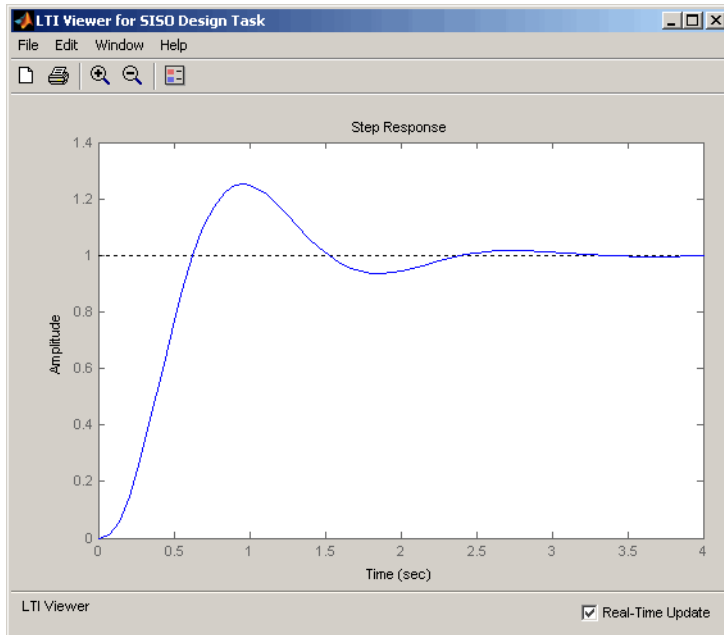
The Step Response plot shows that the rise time is now about 0.4 second and peak response is 1.24 rad/s (i.e., the overshoot is about 25%). Although the rise time meets the requirement, the overshoot is still too large, and the stability margins are still unacceptable, so you must tune the lead parameters.

Moving Compensator Poles and Zeros

To improve the response speed, edit the selected dynamics for the lead network in the **Edit Selected Dynamics** group box on the **Compensator Editor** page.

1 Change the value of the lead network zero (Real Zero) to move it closer to the left-most (slowest) pole of the DC motor plant (denoted by a blue `x').

2 Change the value of the lead network pole (Real Pole) to move it to the right. Notice how the gain margin increases (as shown in the Graphical Tuning window) as you do this.

As you tune these parameters, look at the LTI Viewer. You will see the closed-loop step response alter with each parameter change you make. The following figure shows the final values for a design that meets the specifications.
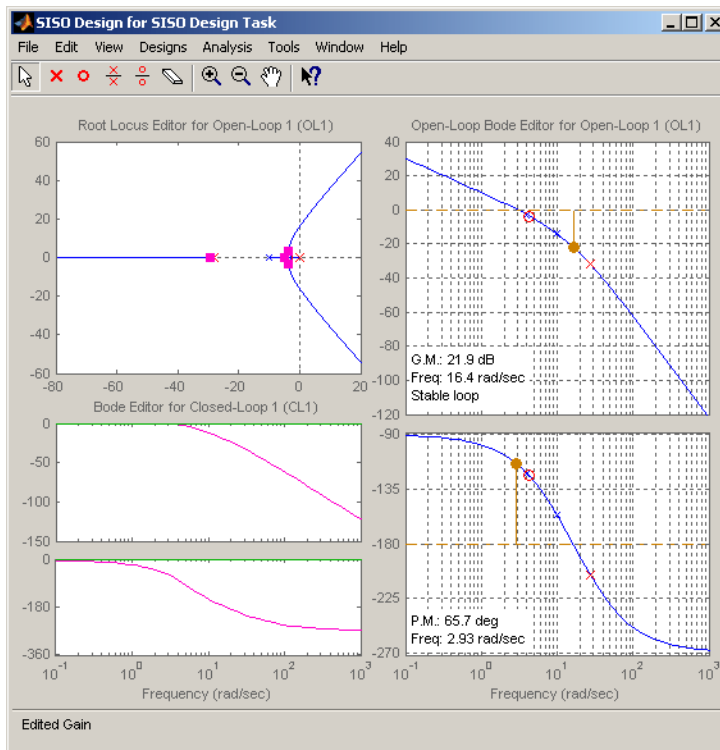
Graphical Tuning Window with Final Design Parameters for the DC Motor Compensator

The values for this final design are as follows:

- Poles at 0 and -28
- Zero at -4.3
- Gain = 84

Enter these values directly in the **Edit Selected Dynamics** group box in the **Compensator Editor** page, shown as follows (Integrator is already set to 0).
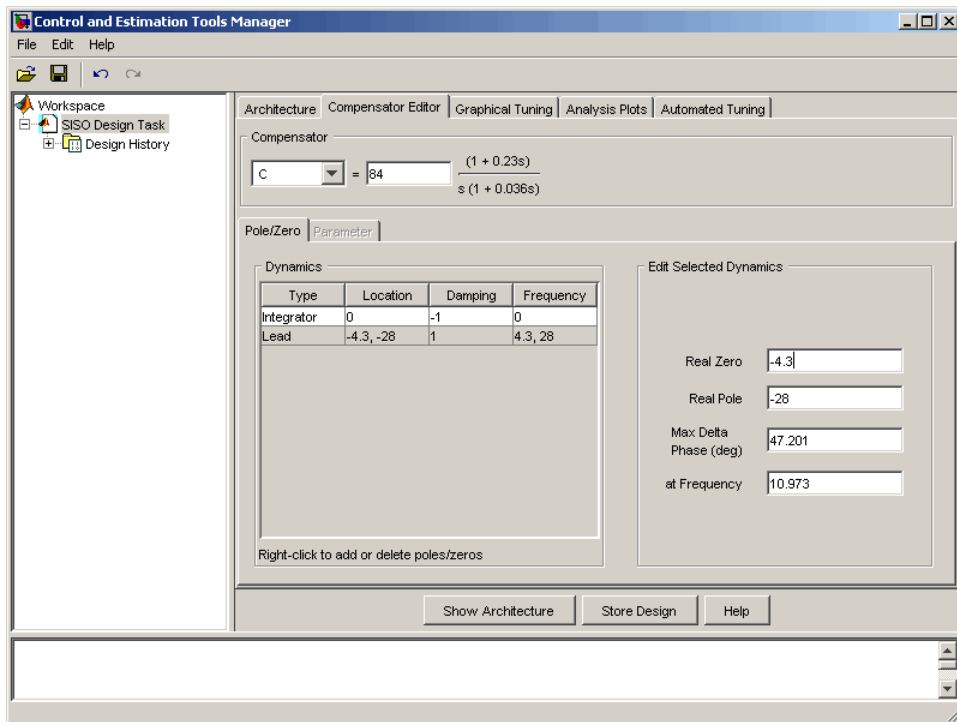
Entering Final Design Parameters on the Compensator Editor Page

The following figure shows the step response for the final compensator design.
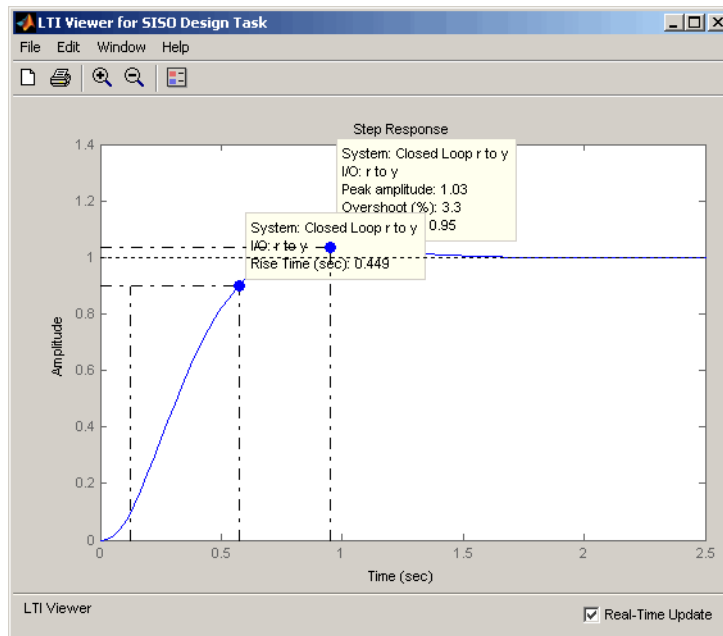
Step Response for the Final Compensator Design

In the LTI Viewer's right-click menu, select **Characteristics** > **Peak Response** and **Characteristics** > **Rise Time** to show the peak response and rise time, respectively. Hover the mouse over the blue dots to show the data markers. The step response shows that the rise time is 0.45 second, and the peak amplitude is 1.03 rad/s, or an overshoot of 3%. These results meet the design specifications.

Adding a Notch Filter

If you know that you have disturbances to your system at a particular frequency, you can use a notch filter to attenuate the gain of the system at that frequency. To add a notch filter, click the **Compensator Editor** tab to open the **Compensator Editor** page. Right-click in the **Dynamics** table and select **Add Pole/Zero > Notch**, as shown next.
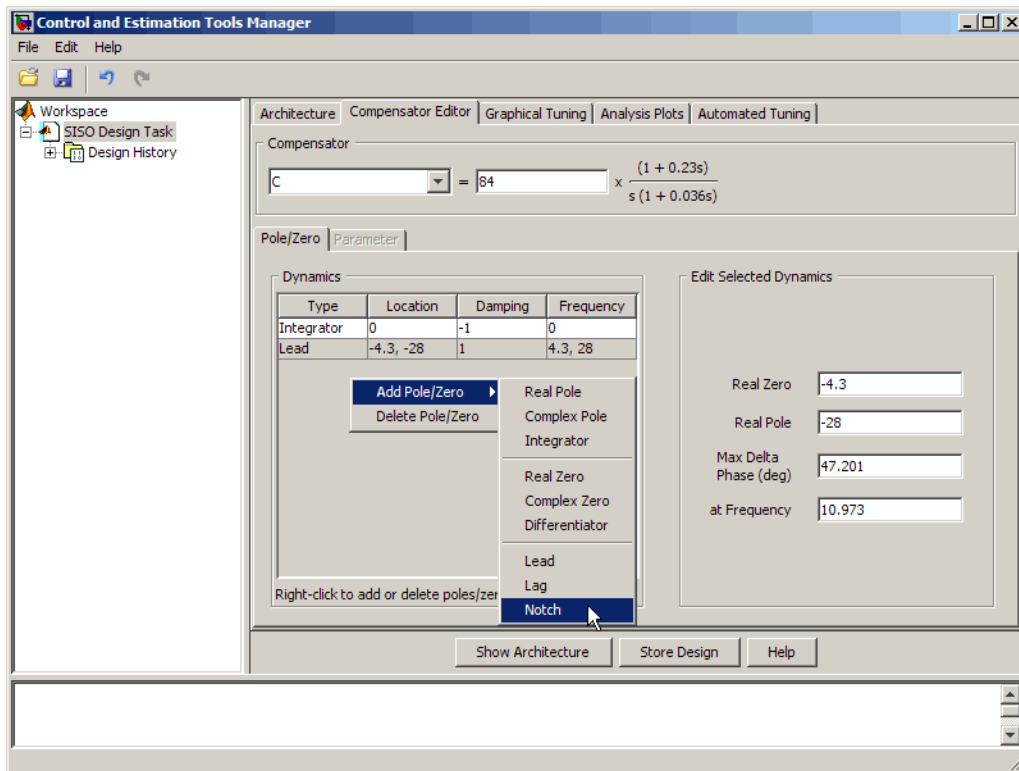
Adding a Notch Filter with the Dynamics Right-Click Menu

Default values for the filter are supplied, as shown next.
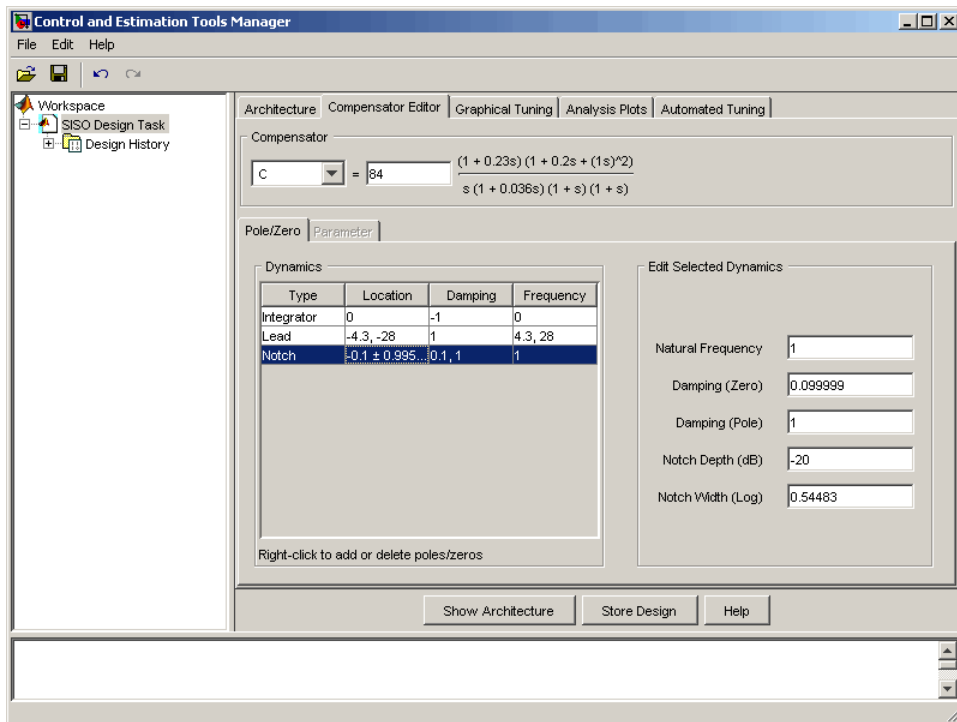
Notch Filter Default Values

The following figure shows the result in the Graphical Tuning window.
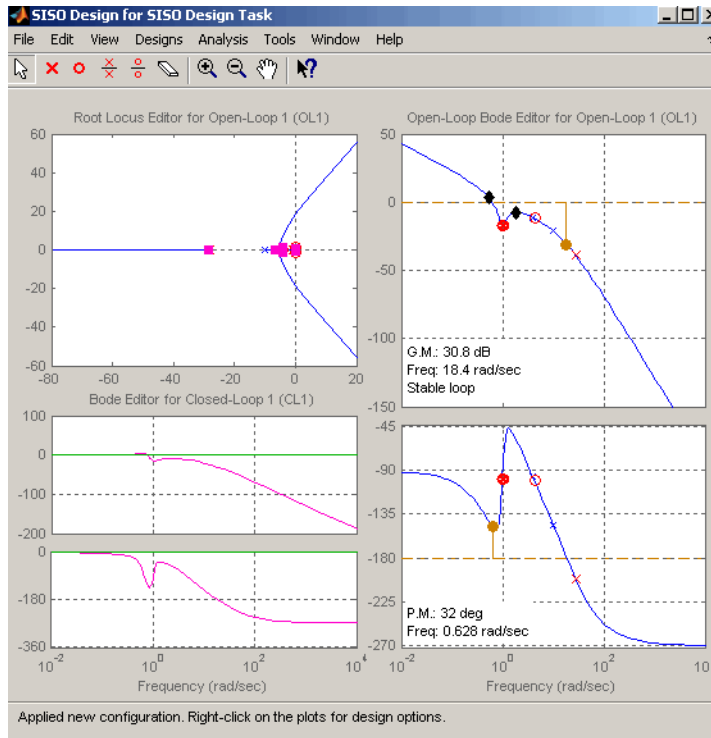
Notch Filter Added to the DC Motor Compensator

To see the notch filter parameters in more detail, click the Zoom In

icon on the Graphical Tuning window. In the Open-Loop Bode Editor, press the left mouse button and drag your mouse to draw a box around the notch filter. When you release the mouse, the Graphical Tuning window will zoom in on the selected region.

To understand how adjusting the notch filter parameters affects the filter, consider the notch filter transfer function.

$$\frac{s^2 + 2\xi_1\omega_n s + \omega_n^2}{s^2 + 2\xi_2\omega_n s + \omega_n^2}$$

The three adjustable parameters are ξ_1, ξ_2, and ω_n. The ratio of ξ_2/ξ_1 sets the depth of the notch, and ω_n is the natural frequency of the notch.

This diagram shows how moving the red # and black diamonds changes these parameters, and hence the transfer function of the notch filter.
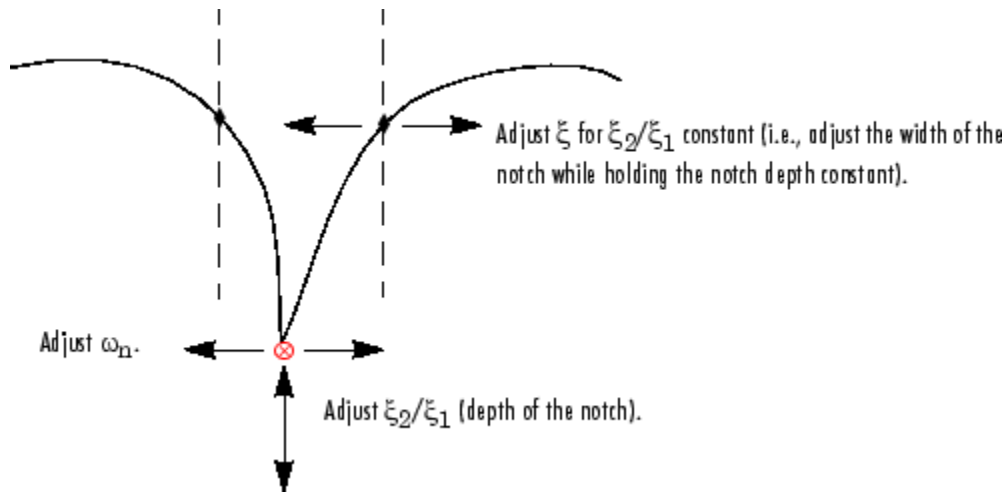
A Close Look at Notch Filter Parameters

In the **Dynamics** table on the **Compensator Editor** page, select the row containing the newly added notch filter. The editable fields appear in the **Edit Selected Dynamics** group box, as shown next.
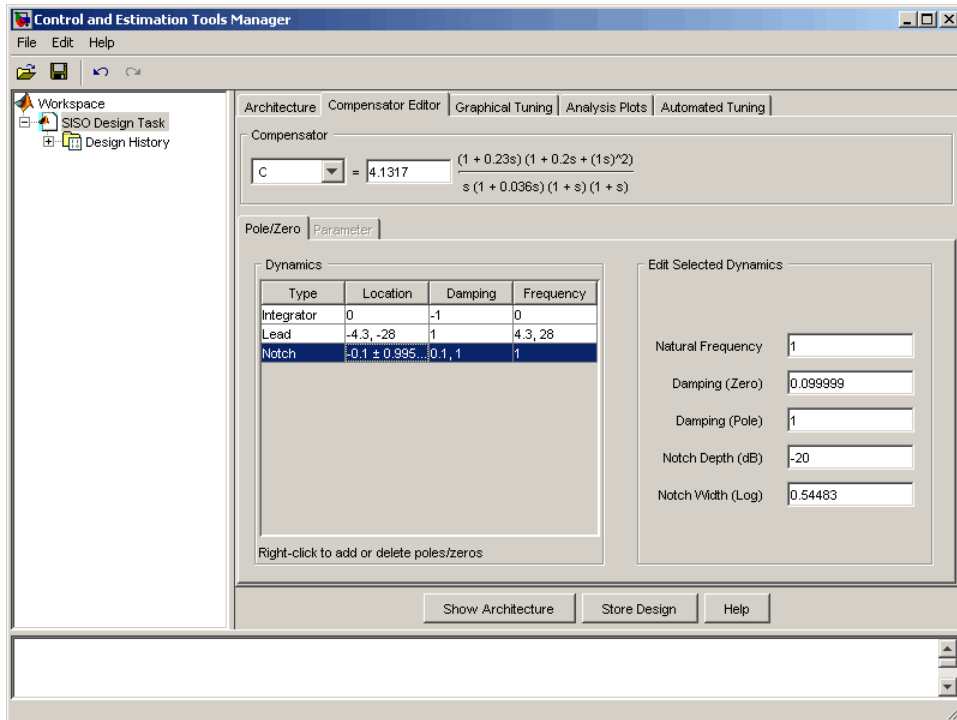
Editing Notch Filter Parameters

Modifying a Prefilter

You can use the SISO Design Tool to modify the prefilter in your design. Typical prefilter applications include:

- Achieving (near) feedforward tracking to reduce load on the feedback loop (when stability margins are poor)
- Filtering out high frequency content in the command (reference) signal to limit overshoot or to avoid exciting resonant modes of the plant

A common prefilter is a simple lowpass filter that reduces noise in the input signal.

Open the Bode diagram for the prefilter by opening the right-click menu in the Closed-Loop Bode Editor in the Graphical Tuning window, and then selecting **Select Compensators > F(F)**.
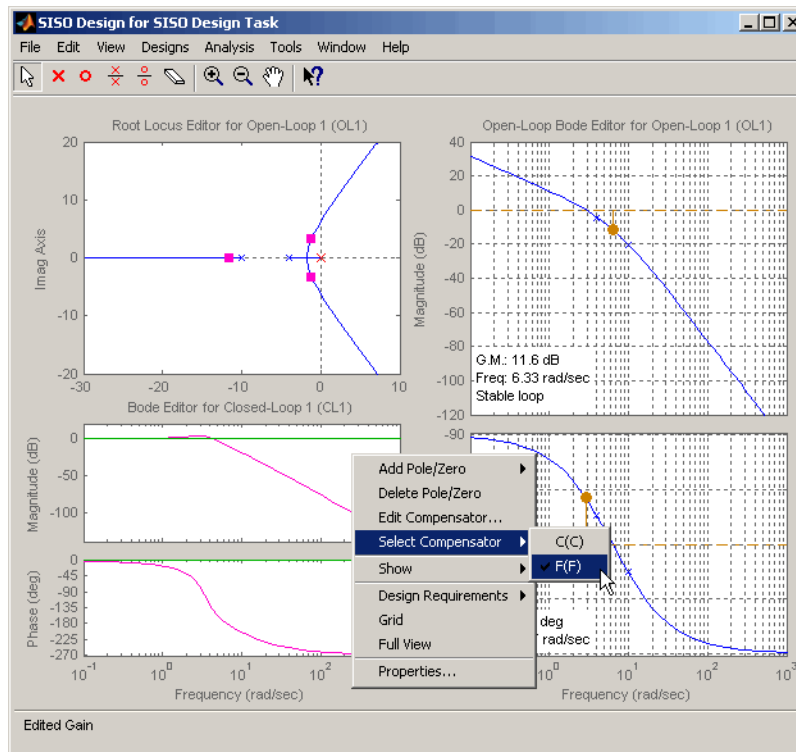
Selecting the Prefilter in the Graphical Tuning Window

For clarity, the previous figure does not show the open-loop Bode diagram for the compensator (**C**). To remove the Bode diagram from the Graphical Tuning window, go to the **SISO Design Task** node on the Control and Estimation Tools Manager, click the **Graphical Tuning** tab, and for Plot 2, Open Loop 1, select Plot type None.
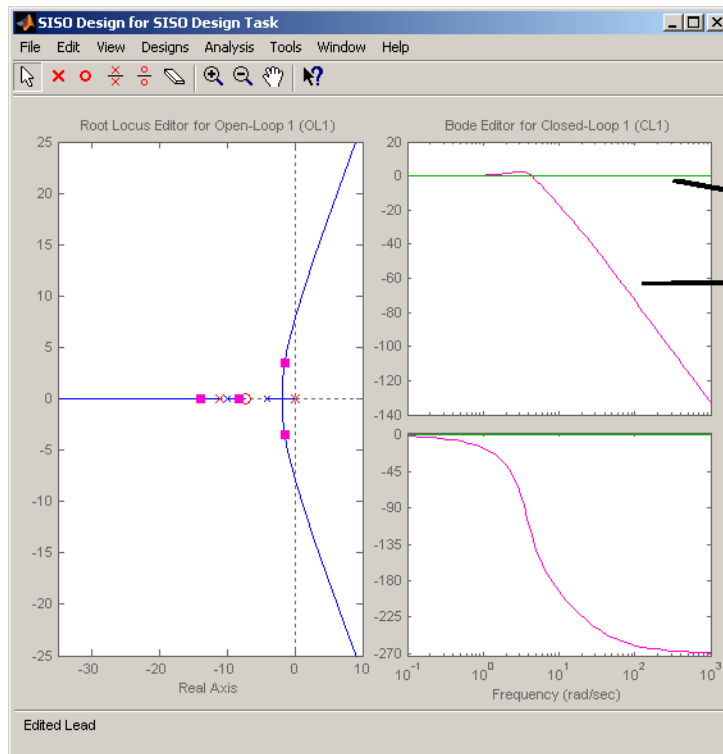
The green curve is
the prefilter Bode
magnitude.

The magenta curve is
the closed-loop response
from the prefilter input
to the plant output.

Prefilter Bode Diagram

If you haven't imported a prefilter, the default is a unity gain. You can add poles and
zeros and adjust the gain using the same methods as you did when designing the
compensator (C) on the **Compensator Editor** page.

A quick way to create a lowpass roll-off filter is to add a pair of complex poles. To do this,
first click the **Compensator Editor** tab and change the compensator to F. Right-click
in the **Dynamics** table and select **Add Pole/Zero > Complex Pole**. Select this line
to show the editable parameters in the **Edit Selected Dynamics** group box. For this
example, try to place the poles at about 50 rad/s. The following figure shows the poles
added to the prefilter Bode diagram.
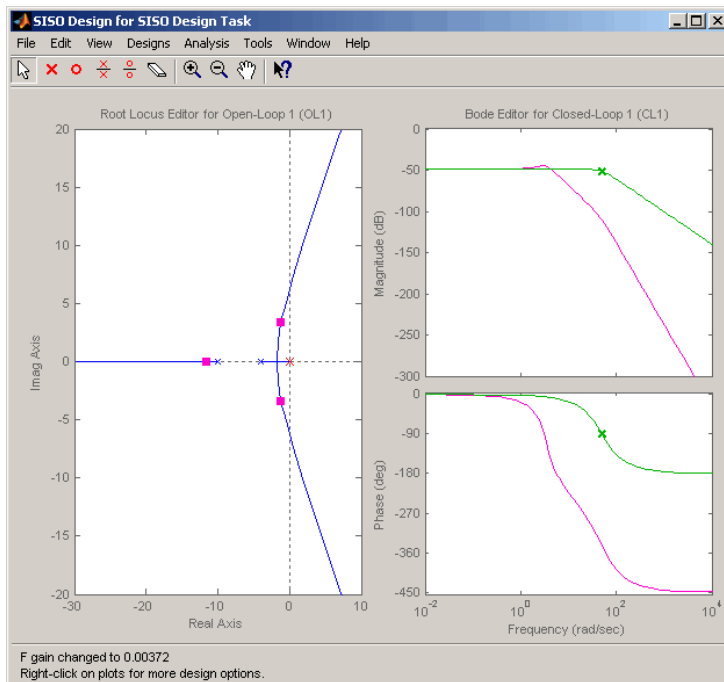
Adding a Complex Pair of Poles to the Prefilter Bode Diagram

By default, the damping ratio of the complex pair is 1.0, which means that there are two real-valued poles at about -50 rad/s. The green curve, which represents the prefilter Bode response, shows the -3 dB point for the roll-off is at about 50 rad/s. The magenta curve, which represents the closed-loop response from the prefilter to the plant output, shows that after the -3 dB point, the closed-loop gain rolls off at -40 dB/decade to provide some noise disturbance rejection.

Importing a Prefilter

As an alternative approach, you can design a prefilter using the Control System Toolbox commands like `ss` or `tf` and then import the design directly into the prefilter. For example, to create the lowpass filter using `zpk`, try

```
prefilt=zpk([],[-35 + 35i, -35 - 35i],1)
```

and import `prefilt` by clicking **System Data** on the **Architecture** page. This opens the System Data dialog box. Click **Browse** to open the Model Import dialog box, as shown next.
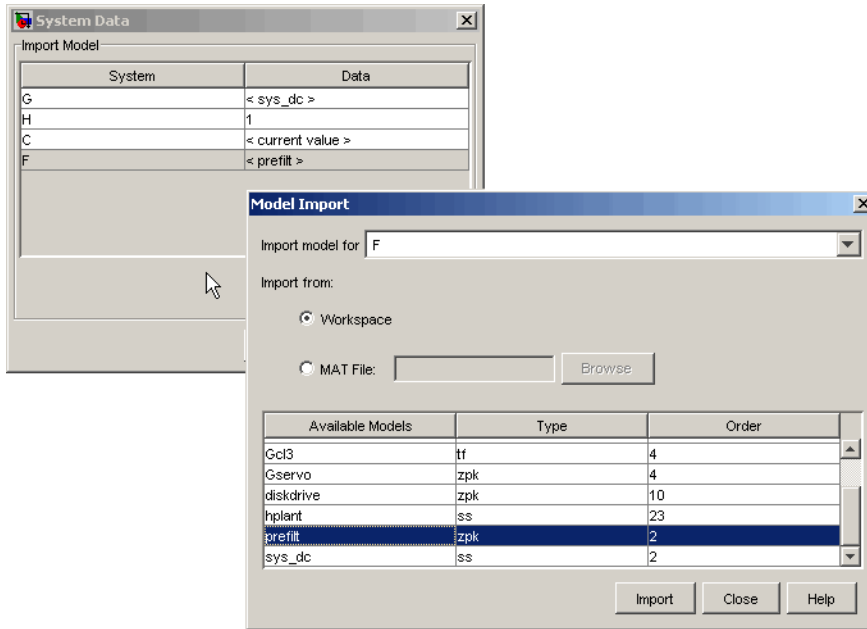
Importing a Prefilter

Select `prefilt` from the **Available Models** list and click **Import** to import the prefilter model. Click **Close** to close the Import Model dialog box. After you have imported the prefilter model, you can modify it using the same methods as described in this chapter for compensator design.

Root Locus Design

In this section...

What Is Root Locus Design?

A common technique for meeting design criteria is root locus design. This approach involves iterating on a design by manipulating the compensator gain, poles, and zeros in the root locus diagram.

As system parameter k varies over a continuous range of values, the root locus diagram shows the trajectories of the closed-loop poles of the feedback system. Typically, the root locus method is used to tune the loop gain of a SISO control system by specifying a designed set of closed-loop pole locations.

Consider, for example, the tracking loop
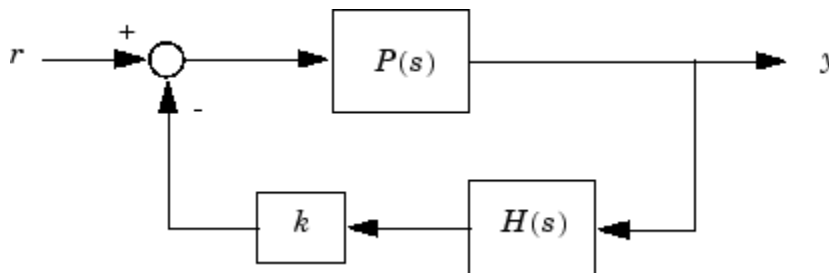
where $P(s)$ is the plant, $H(s)$ is the sensor dynamics, and k is a scalar gain to be adjusted. The closed-loop poles are the roots of

$$q(s) = 1 + k\ P(s)H(s)$$

The root locus technique consists of plotting the closed-loop pole trajectories in the complex plane as k varies. You can use this plot to identify the gain value associated with a desired set of closed-loop poles.

The DC motor design example focused on the Bode diagram feature of the SISO Design Tool. Each of the design options available on the Bode diagram side of the tool have a counterpart on the root locus side. To demonstrate these techniques, this example presents an electrohydraulic servomechanism.

The SISO Design Tool's root locus and Bode diagram design tools provide complementary perspectives on the same design issues; each perspective offers insight into the design process. Because the SISO Design Tool shows both root locus and Bode diagrams, you can also choose to combine elements of both perspectives in making your design decisions.

Root Locus Design for Electrohydraulic Servomechanism

A simple version of an electrohydraulic servomechanism model consists of

- A push-pull amplifier (a pair of electromagnets)
- A sliding spool in a vessel of high-pressure hydraulic fluid
- Valve openings in the vessel to allow for fluid to flow
- A central chamber with a piston-driven ram to deliver force to a load
- A symmetrical fluid return vessel

This figure shows a schematic of this servomechanism.

Return

Amplifier

Pressure

Push-Pull
Amplifier

Piston-driven ram

Load
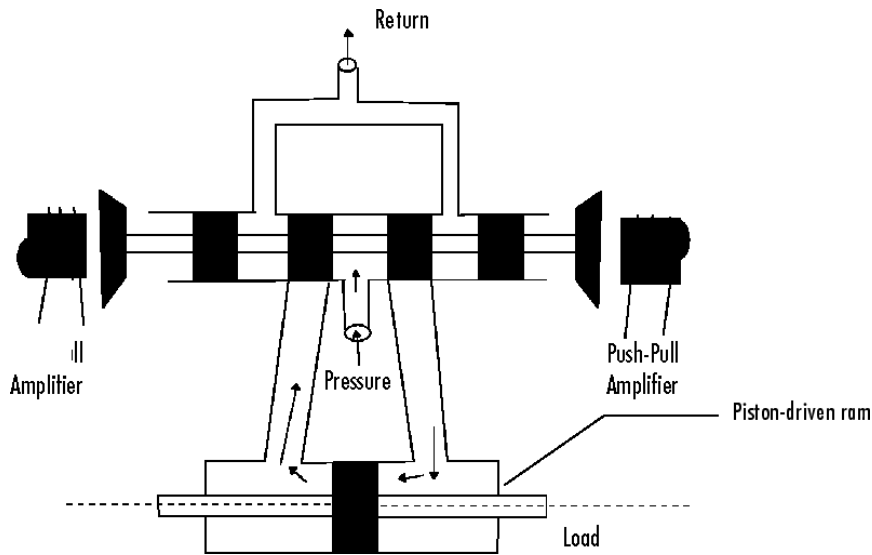
Electrohydraulic Servomechanism

The force on the spool is proportional to the current in the electromagnet coil. As the spool moves, the valve opens, allowing the high-pressure hydraulic fluid to flow through the chamber. The moving fluid forces the piston to move in the opposite direction of the spool. *Control System Dynamics,* by R. N. Clark, (Cambridge University Press, 1996) derives linearized models for the electromagnetic amplifier, the valve spool dynamics, and the ram dynamics; it also provides a detailed description of this type of servomechanism.

If you want to use this servomechanism for position control, you can use the input voltage to the electromagnet to control the ram position. When measurements of the ram position are available, you can use feedback for the ram position control, as shown in the following figure.

4-131
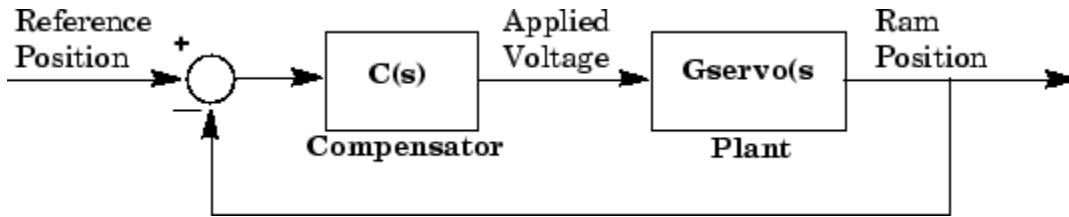
Feedback Control Structure for an Electrohydraulic Servomechanism

Your task is to design the compensator, **C(s)**.

Plant Transfer Function

If you have not already done so, type

```
load ltiexamples
```

to load a collection of linear models that include `Gservo`, which is a linearized plant transfer function for the electrohydraulic position control mechanism. Typing `Gservo` at the MATLAB prompt opens the servomechanism (plant) transfer function.

```
Gservo

Zero/pole/gain from input "Voltage" to output "Ram position":
        40000000
----------------------------
s (s+250) (s^2 + 40s + 9e004)
```

Design Specifications

For this example, you want to design a controller so that the step response of the closed-loop system meets the following specifications:

- The 2% settling time is less than 0.05 second.
- The maximum overshoot is less than 5%.

The remainder of this topic discusses how to use the SISO Design Tool to design a controller to meet these specifications.

Opening the SISO Design Tool

Open the SISO Design Tool and import the model by typing

```
sisotool(Gservo)
```

at the MATLAB prompt. This opens the **SISO Design Task** node in the Control and Estimation Tools Manager and the Graphical Tuning window with the servomechanism plant imported.
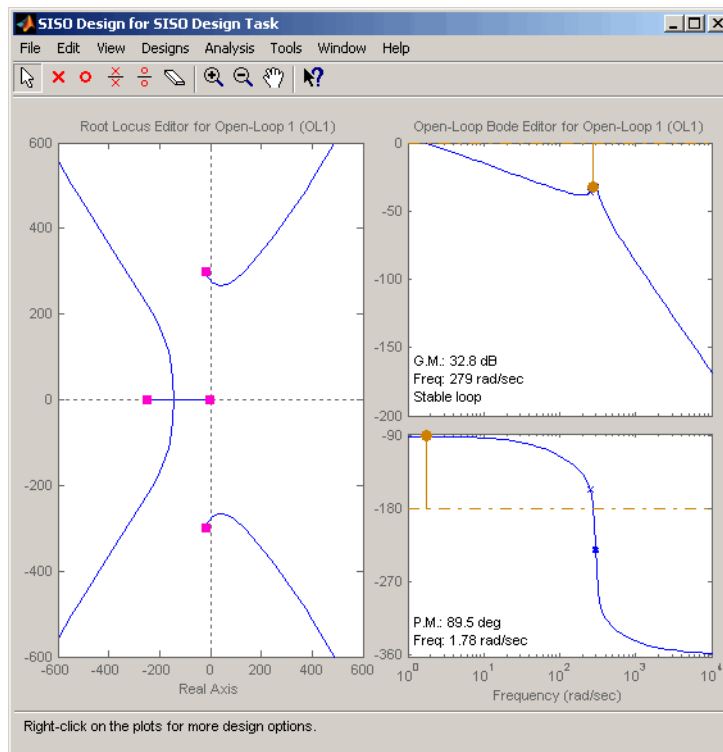
Graphical Tuning Window Showing the Root Locus and Bode Plots for the Electrohydraulic Servomechanism Plant

Zooming

Click the Zoom In

icon in the Graphical Tuning window. Press and hold the mouse's left button and drag the mouse to select a region for zooming. For this example, reduce the root locus region to about -500 to 500 in both the *x*- and *y*-axes. This figure illustrates the zooming in process.
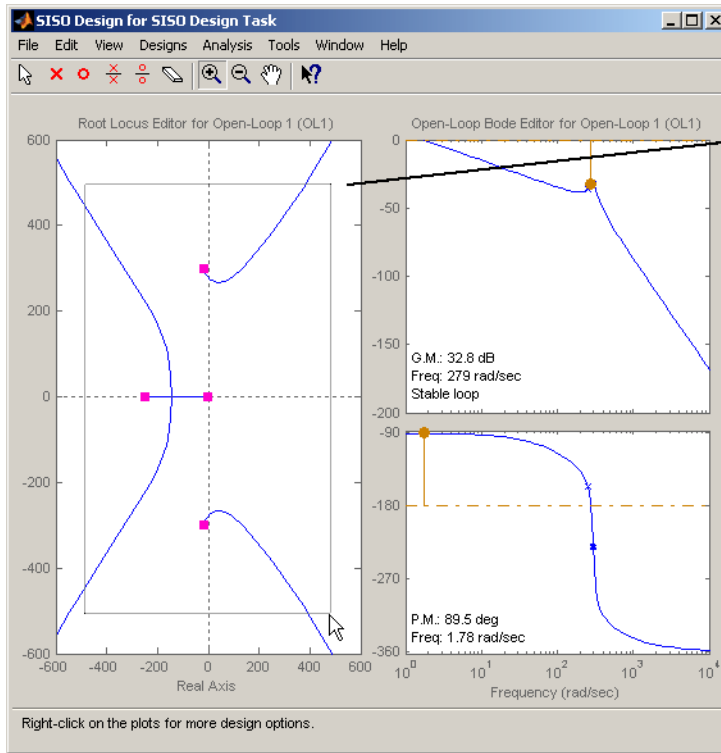
Hold the left mouse button down and drag the mouse to select a rectangular region for zooming in. When you release the mouse button, the Graphical Tuning window replots the root locus with the new axis boundaries.

To undo the zoom, right-click in the Root Locus Editor graph and select **Reset to Original View** from the pop-up menu.

When you are done zooming, click the Zoom In icon again to turn zooming off.

Zooming In on a Region in the Root Locus Plot

As in the DC motor example, click the **Analysis Plots** tab to set up loop responses. Select Plot Type Step for Plot 1, then select plot 1 for Closed-Loop r to y, shown as follows.
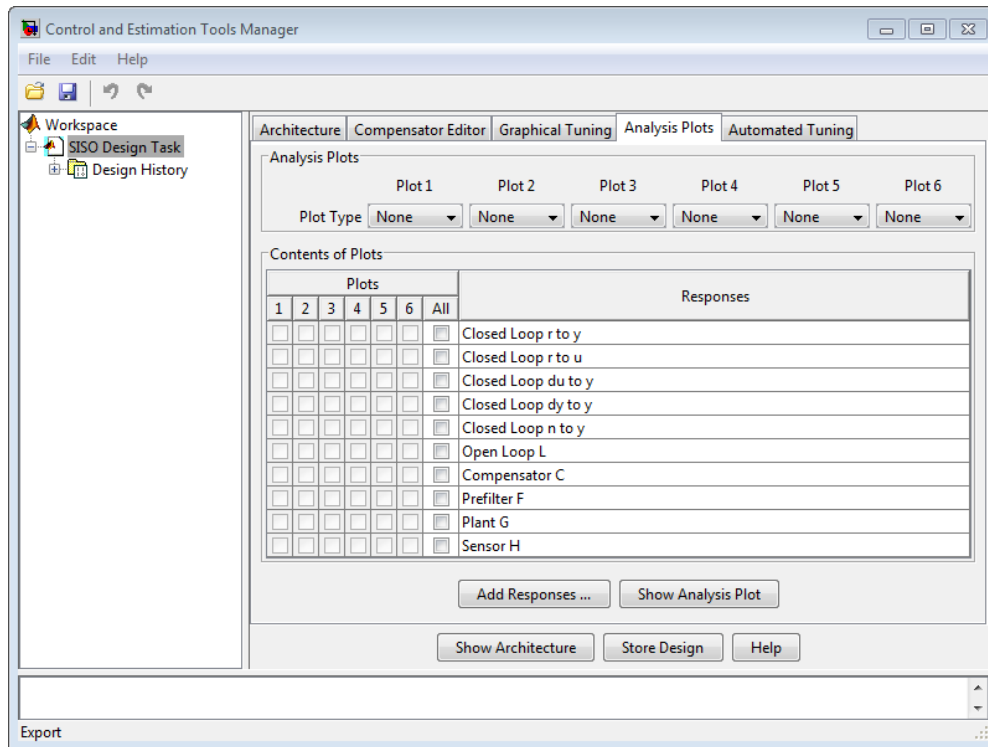
Analysis Plots Loop Response Selection

For more information about the **Analysis Plots** page, see "Analysis Plots" in "Using the SISO Design Tool and LTI Viewer."

Selecting the plot for Closed-Loop r to y opens the associated LTI Viewer.

Your LTI Viewer should look like the following figure.
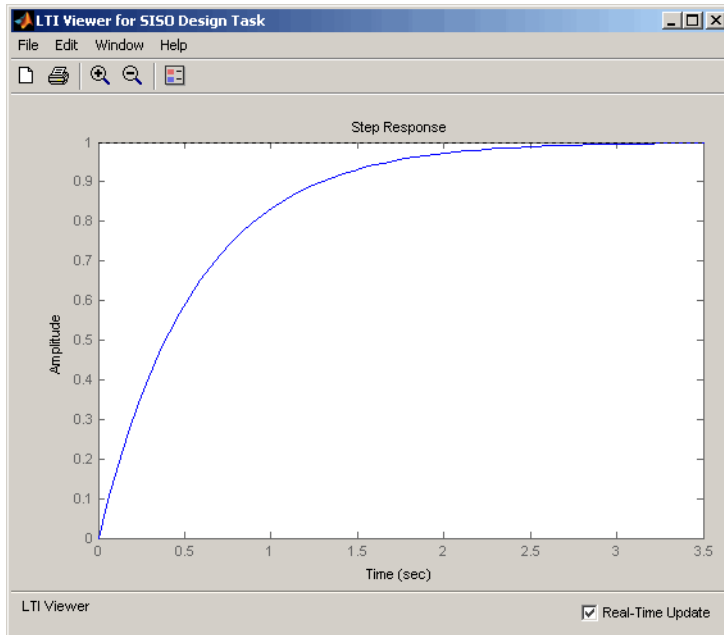
LTI Viewer for the Electrohydraulic Servomechanism

The step response plot shows that the rise time is on the order of 2 seconds, which is much too slow given the system requirements. The following topics describe how to use frequency design techniques in the SISO Design Tool to design a compensator that meets the requirements specified in "Design Specifications" on page 4-132.

Changing the Compensator Gain

The simplest thing to do is change the compensator gain, which by default is unity. You can change the gain by entering the value directly in the **Compensator Editor** page.

The following figure shows this procedure.
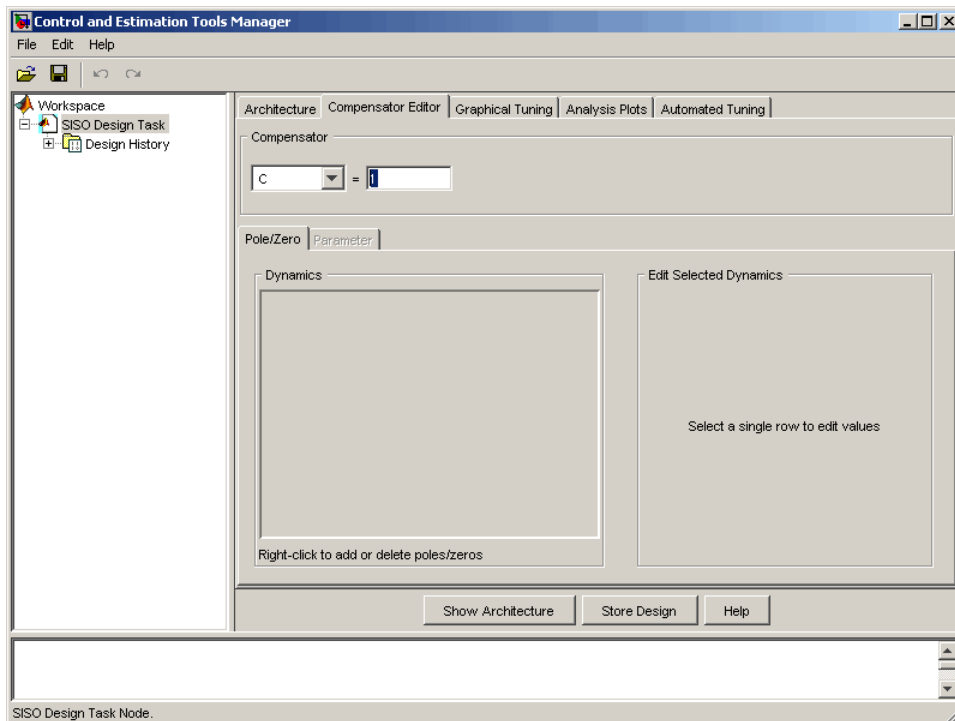
Changing the Compensator Gain in the Root Locus Plot with the Compensator Editor Page

Enter the compensator gain in the text box to the right of the equal sign as shown in the previous figure. The Graphical Tuning window automatically replots the graphs with the new gain.

Experiment with different gains and view the closed-loop response in the associated LTI Viewer.

Alternatively, you can change the gain by grabbing the red squares on the root locus plot in the Graphical Tuning window and moving them along the curve.

Closed-Loop Response

Change the gain to 20 by editing the text box next to **the equal sign** on the **Compensator Editor** page. Notice that the locations of the closed-loop poles on the root locus are recalculated for the new gain.

4-137

This figure shows the associated closed-loop step response for the gain of 20.
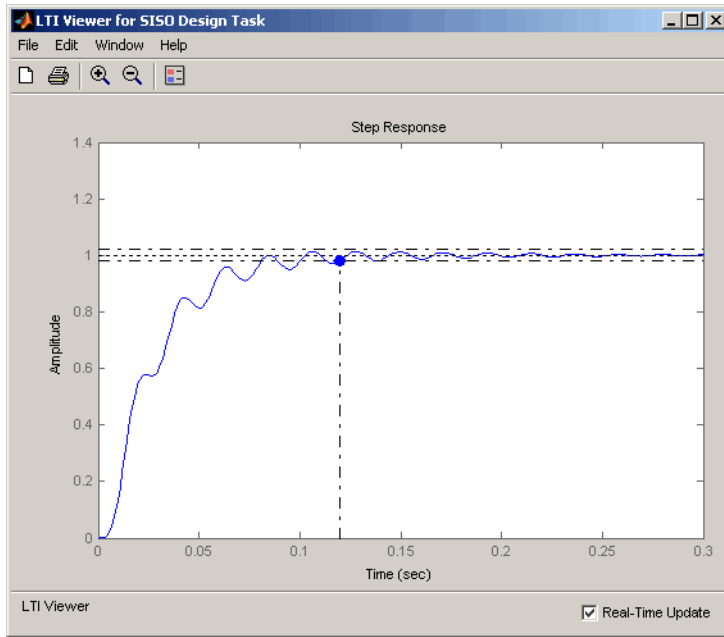
Step Response with the Settling Time for C(s) = 20

In the LTI Viewer's right-click menu, select **Characteristics** > **Settling Time** to show the settling for this response. This closed-loop response does not meet the desired settling time requirement (0.05 seconds or less) and exhibits unwanted ringing. "Adding Poles and Zeros to the Compensator" on page 4-138 shows how to design a compensator so that you meet the required specifications.

Adding Poles and Zeros to the Compensator

You may have noticed that increasing the gain makes the system under-damped. Further increases force the system into instability, so meeting the design requirements with only a gain in the compensator is not possible.

There are three sets of parameters that specify the compensator: poles, zeros, and gain. After you have selected the gain, you can add poles or zeros to the compensator.

Adding Poles to the Compensator

You can add complex poles on the **Compensator Editor** page. Click the **Compensator Editor** tab, make sure **C** is selected, and then right click in the **Dynamics** table. Select **Add Pole/Zero > Complex Pole**. Use the **Edit Selected Dynamics** group box to modify pole parameters, as shown in the following figure. For more about entering pole parameters directly, see "Editing Compensator Pole and Zero Locations" on page 4-143.
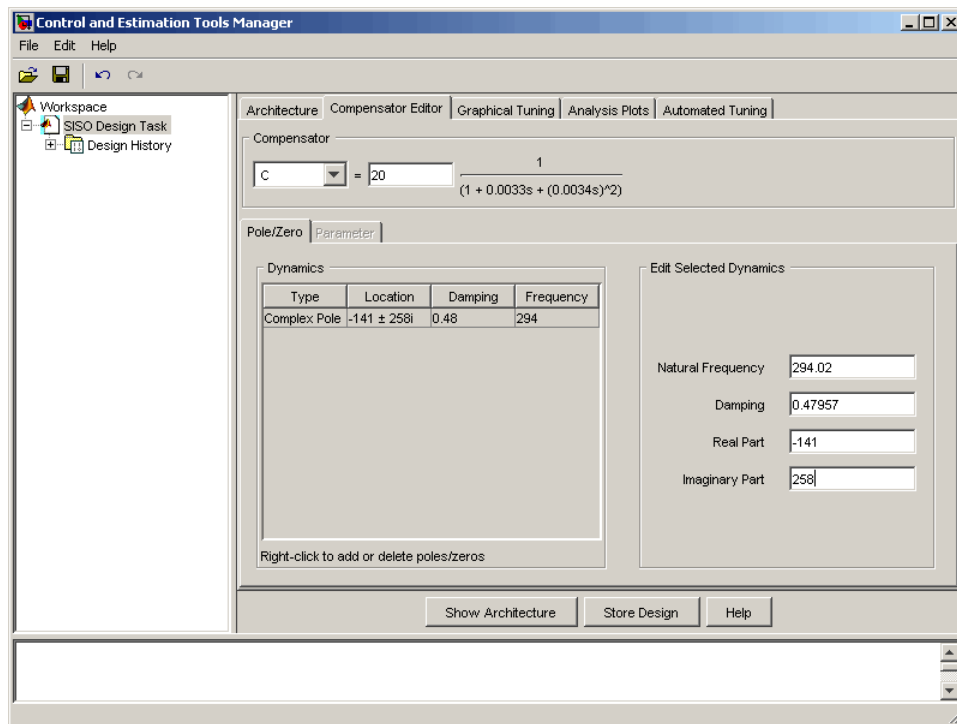
Adding a Complex Pair of Poles to the Compensator on the Compensator Editor Page

You can also add a complex pole pair directly on the root locus plot using the Graphical Tuning window. Right-click in the root locus plot and select **Add Pole/Zero > Complex Pole**. Click in the root locus plot region where you want to add one of the complex poles.

Complex poles added in this manner are automatically added to the **Dynamics** table in the **Compensator Editor** page.

After you have added the complex pair of poles, the LTI Viewer response plots change and both the root locus and Bode plots show the new poles.

This figure shows the Graphical Tuning window with the new poles added. For clarity, you may want to zoom out further, as was done here.
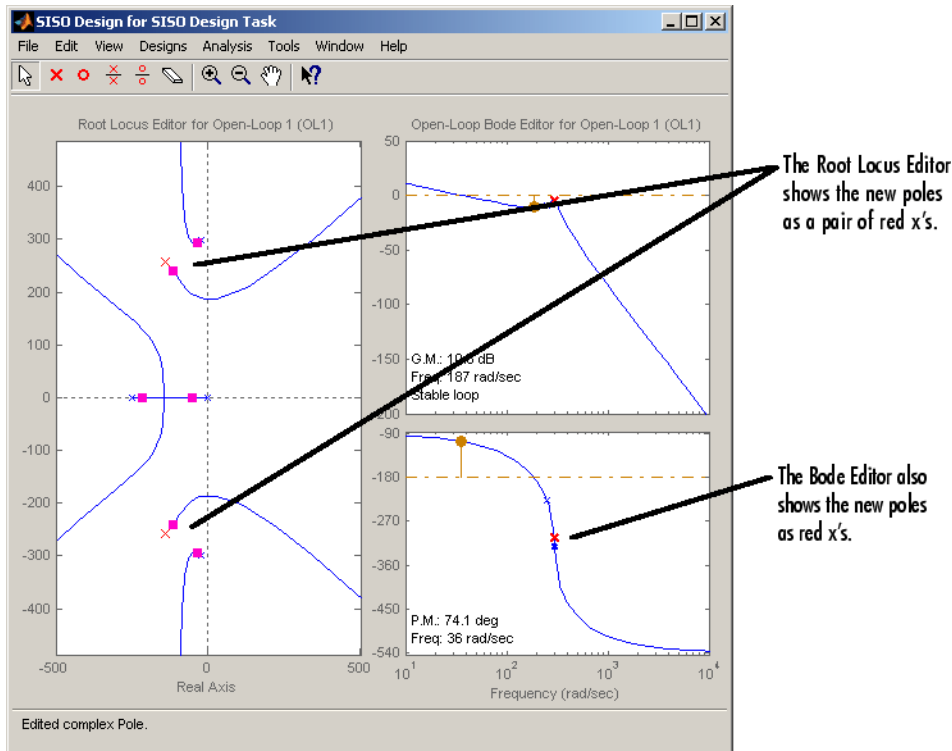
Result of Adding a Complex Pair of Poles to the Compensator

Adding Zeros to the Compensator

You can add zeros in the **Dynamics** table on the **Compensator Editor** page or directly on the Root Locus plot in the Graphical Tuning window.

To add the zeros using the **Compensator Editor** page, click the **Compensator Editor** tab, make sure **C** is selected, and then right click in the **Dynamics** table. Select **Add Pole/Zero > Complex Zero**. Use the **Edit Selected Dynamics** group box to modify

zero parameters, as shown. For more about entering zero parameters directly, see "Editing Compensator Pole and Zero Locations" on page 4-143.
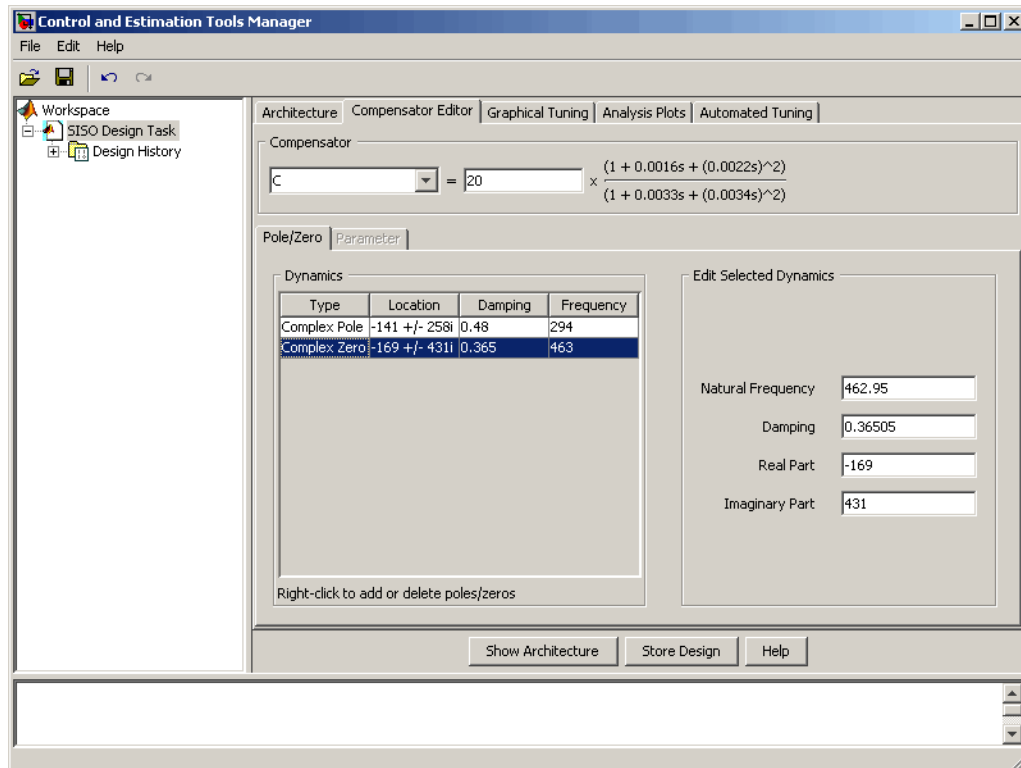
Adding Complex Zeros to the Compensator on the Compensator Editor Page

You can also add complex zeros directly on the root locus plot using the Graphical Tuning window by right-clicking in the root locus plot, selecting **Add Pole/Zero > Complex Zero**, and then clicking in the root locus plot region where you want to add one of the zeros.

Complex zeros added in this manner are automatically added to the **Dynamics** table on the **Compensator Editor** page.

After you have added the complex zeros, the LTI Viewer response plots change and both the root locus and Bode plots show the new zeros.
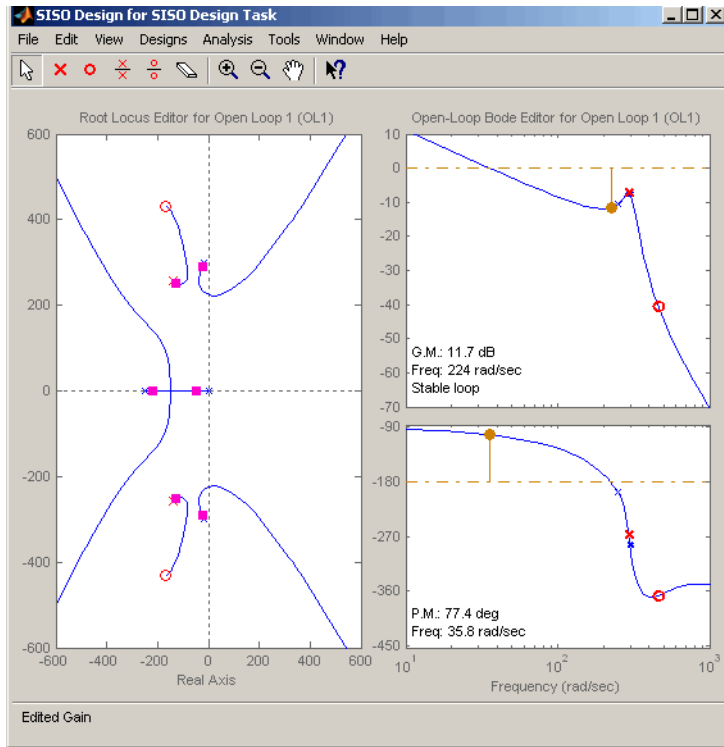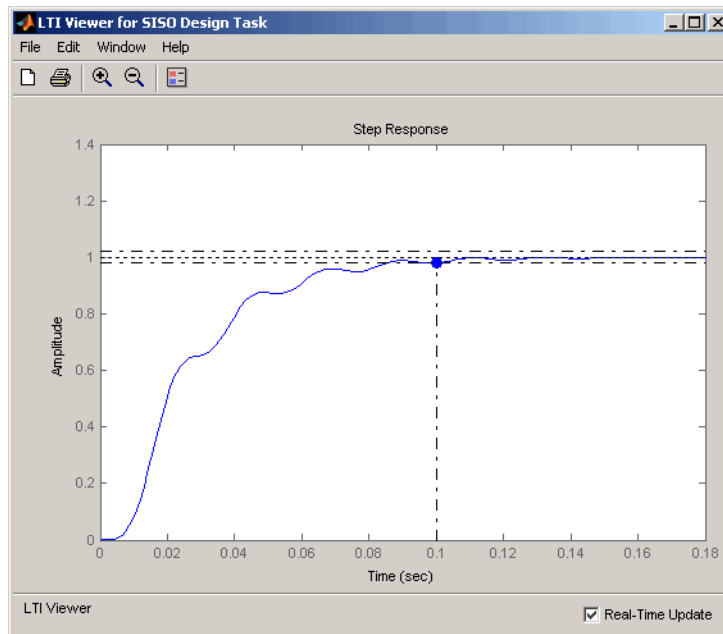
Electrohydraulic Servomechanism Example with Complex Zeros Added

If your step response is unstable, lower the gain by grabbing a red box in the right-side plane and moving it into the left-side plane. In this example, the resulting step response is stable, but it still doesn't meet the design criteria since the 2% settling time is greater than 0.05 second.

As you can see, the compensator design process can involve some trial and error. You can try dragging the compensator poles, compensator zeros, or the closed-loop poles around the root locus until you meet the design criteria.

"Editing Compensator Pole and Zero Locations" on page 4-143, shows you how to modify the placement of poles and zeros by specifying their numerical values on the **Compensator Editor** page. It also presents a solution that meets the design specifications for the servomechanism example.

Editing Compensator Pole and Zero Locations

A quick way to change poles and zeros is simply to grab them with your mouse and move them around the root locus plot region. If you want to specify precise numerical values,

however, you should use the **Compensator Editor** page in the **SISO Design Task** node on the Control and Estimation Tools Manager to change the gain value and the pole and zero locations of your compensator, as shown.
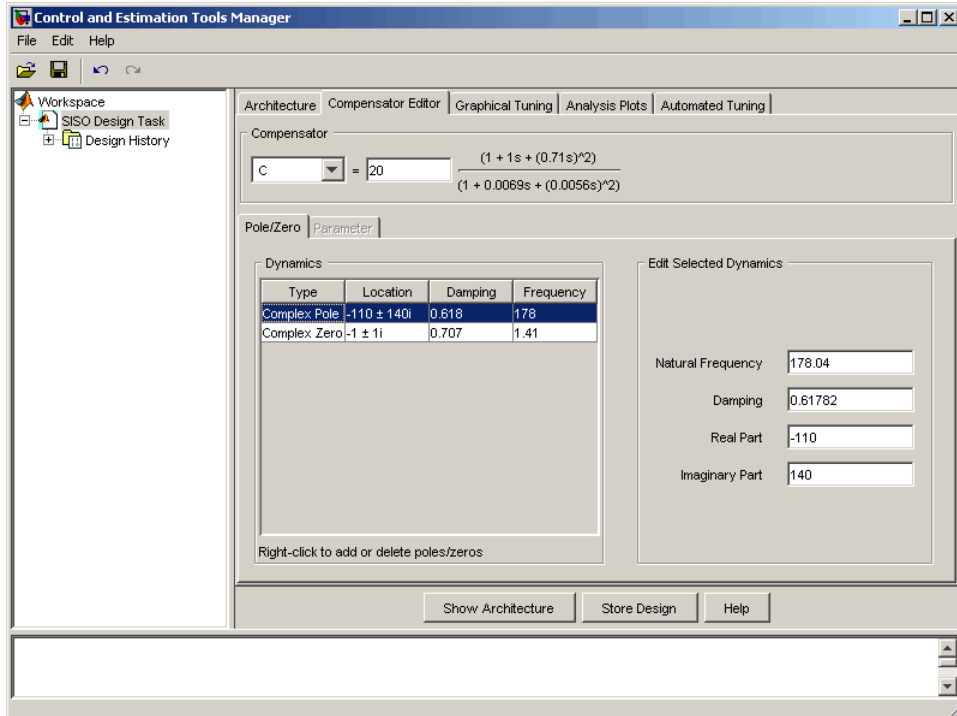
Using the Compensator Editor Page to Add, Delete, and Move Compensator Poles and Zeros

You can use the **Compensator Editor** page to

- Add compensator poles and zeros.
- Delete compensator poles and zeros.
- Edit the compensator gain.
- Edit the locations of compensator poles and zeros.

Adding Compensator Poles and Zeros

To add compensator poles or zeros:

1 Select the compensator (in this example, **C**) in the list box to the left of the equal sign.

2 Right-click in the **Dynamics** table for the pop-up menu.

3 From the pop-up menu, select **Add Pole/Zero > Complex Pole** or **Add Pole/Zero > Complex Zero**.

4 Use the **Edit Selected Dynamics** group box to modify pole or zero parameters.

Deleting Compensator Poles and Zeros

To delete compensator poles or zeros:

1 Select the compensator (in this example, **C**) in the list box to the left of the equal sign.

2 Select the pole or zero in the **Dynamics** table that you want to delete.

3 Right-click and select **Delete Pole/Zero** from the pop-up menu.

Editing Gain, Poles, and Zeros

To edit compensator gain:

1 Select the compensator to edit in the list box to the left of the equal sign in the **Compensator** area.

2 Enter the gain value in the text box to the right of the equal sign in the **Compensator** area.

To edit pole and zero locations:

1 Select the pole or zero you want to edit in the **Dynamics** table.

2 Change current values in the **Edit Selected Dynamics** group box.

For this example, edit the poles to be at $-110 \pm 140i$ and the zeros at $-70 \pm 270i$. Set the compensator gain to `23.3`.

Your Graphical Tuning window now looks like this.
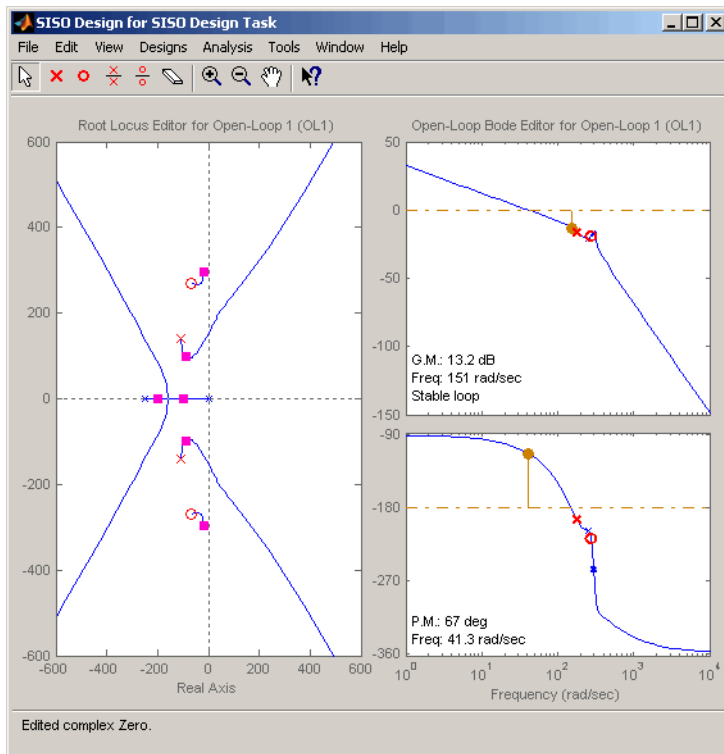
Graphical Tuning Window with the Final Values for the Electrohydraulic Servomechanism Design Example

To see that this design meets the design requirements, look at the step response of the closed-loop system.
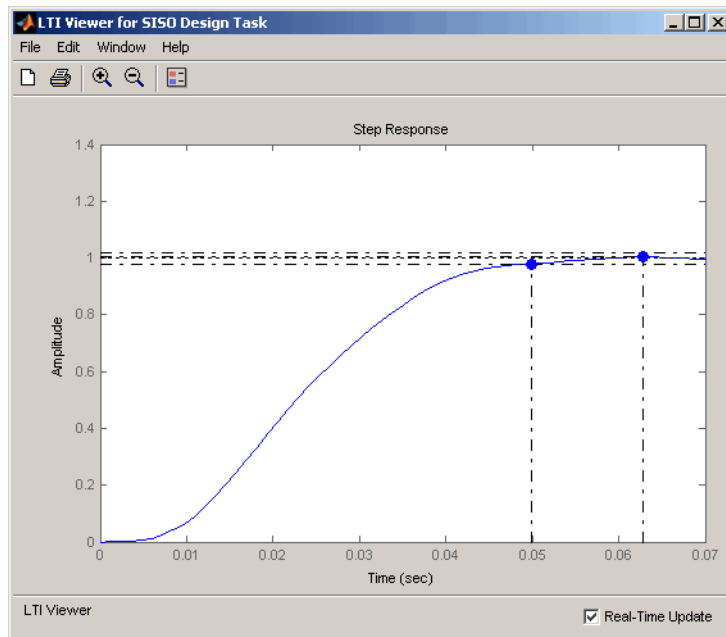
Closed-Loop Step Response for the Final Design of the Electrohydraulic Servomechanism Example

The step response looks good. The settling time is less than 0.05 second, and the overshoot is less than 5%. You have met the design specifications.

Viewing Damping Ratios

The Graphical Tuning window provides design requirements that can make it easier to meet design specifications. If you want to place, for example, a pair of complex poles on your diagram at a particular damping ratio, select **Design Requirements > New** from the right-click menu in the root locus graph.

This opens the **New Design Requirement** dialog box.
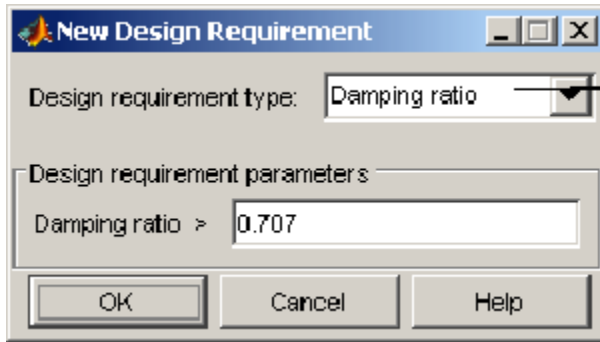
Select **Damping ratio** from this list. The default value 0.707 sets lines on the root locus corresponding to a 0.707 damping ratio.

Applying damping ratio requirements to the root locus plot results in a pair of shaded rays at the desired slope, as this figure shows.
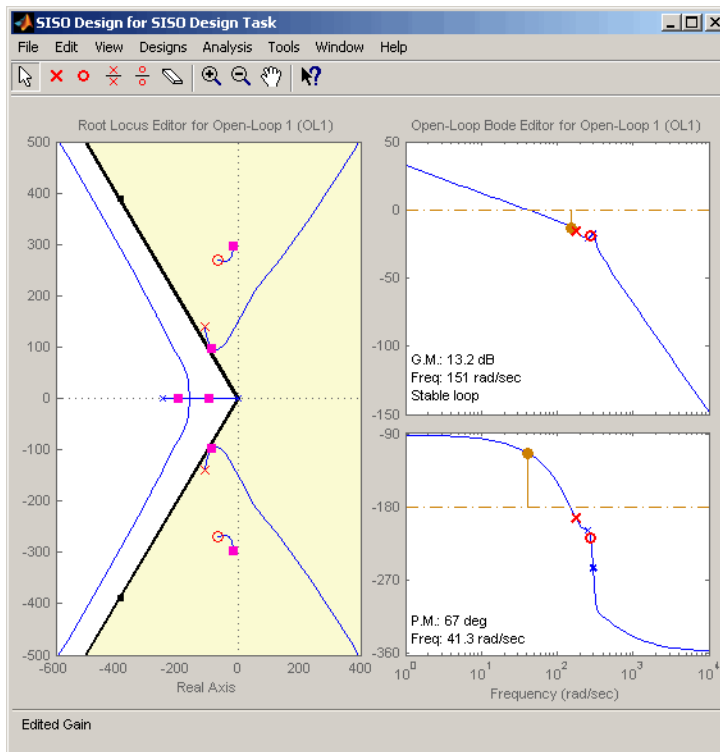
Root Locus with 0.707 Damping Ratio Lines

Try moving the complex pair of poles you added to the design so that they are on the 0.707 damping ratio line. You can experiment with different damping ratios to see the effect on the design.

If you want to change the damping ratio, select **Design Requirements > Edit** from the right-click menu. This opens the Edit Design Requirements dialog box.

Specify the new damping ratio requirement in this dialog box.

An alternate way to adjust a requirement is to left-click the requirement itself to select it. Two black squares appear on the requirement when it is selected. You can then drag it with your mouse anywhere in the plot region.

If you want to add a different set of requirements, for example, a settling time requirement, again select **Design Requirements > New** from the right-click menu to open the New Requirements dialog box and choose **Settling time** from the pull-down menu. You can have multiple types of design requirements in one plot, or more than one instance of any type.

The types of requirements available depend on which view you use for your design. See "Design Requirements" for a description of all the design requirement options available in the SISO Design Tool.

Nichols Plot Design

What Is Nichols Plot Design?

An alternative method for designing compensators is to use the Nichols plot, which combines gain and phase information in a single plot. The combination of the two is useful when you are designing to gain and phase margin specifications.

You can design compensators with the SISO Design Tool by using Nichols plot techniques. This topic repeats the DC motor compensator design presented in "Bode Diagram Design for DC Motor" on page 4-103, only this time the focus is on Nichols plot techniques. The design strategy, however, is the same.

1 Adjust the compensator gain to improve the rise time.
2 Add an integrator to eliminate steady-state error.
3 Add a lead network to further improve the rise time while minimizing overshoot.

Nichols Plot Design for DC Motor

From "SISO Example: The DC Motor" on page 2-3, the transfer function of the DC motor is

```
Transfer function:
      1.5
-----------------
s^2 + 14 s + 40.02
```

This example uses the design criteria specified in "Design Specifications" on page 4-132:

·   Rise time of less than 0.5 second
·   Steady-state error of less than 5%

- Overshoot of less than 10%
- Gain margin greater than 20 dB
- Phase margin greater than 40 degrees

Opening a Nichols Plot

To open the SISO Design Tool with a Bode diagram and a Nichols plot, use these commands:

```
load ltiexamples
sisotool({'bode','nichols'},sys_dc)
```

The **SISO Design Task** node on the Control and Estimation Tools Manager opens and the Graphical Tuning window with sys_dc opens.
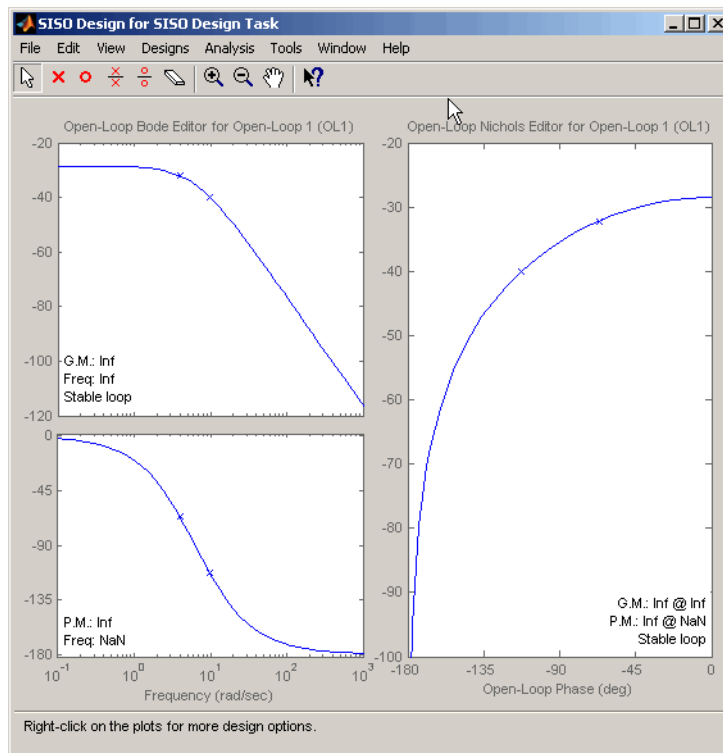
Graphical Tuning Window with a Bode Diagram and a Nichols Plot

Adjusting the Compensator Gain

You can adjust the compensator gain by entering a value in the **Compensator Editor** page.

1 Click the **Compensator Editor** tab to open the **Compensator Editor** page.

2 Select **C** from the compensator selection list.

3 In the text box to the right of the equal sign in the **Compensator** area, enter the gain amount and press **Enter**.
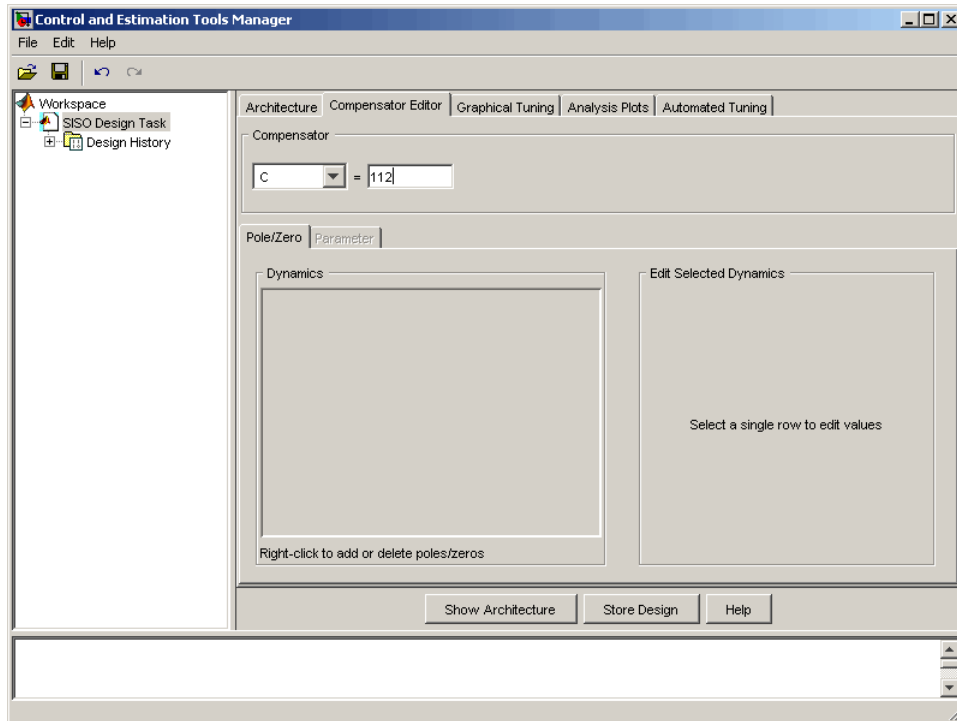
Adjusting Compensator Gain in the Compensator Editor Page

In this example, the new gain is 112.

You can also adjust the compensator gain in the Graphical Tuning window by moving the Nichols curve up and down with your mouse. To do this, place your mouse over the

curve. The cursor turns into a hand. Left-click and move the curve up to increase the gain. When you adjust the gain in this manner, the compensator gain is automatically updated in the **Compensator Editor** page.

Click the **Analysis Plots** tab to set the analysis plots. Select Plot Type `Step` for Plot 1, and then select plot 1 for `Closed-Loop r to y`, as shown in the following figure, to open a linked LTI Viewer with the closed-loop step response from reference signal *r* to output signal *y*.
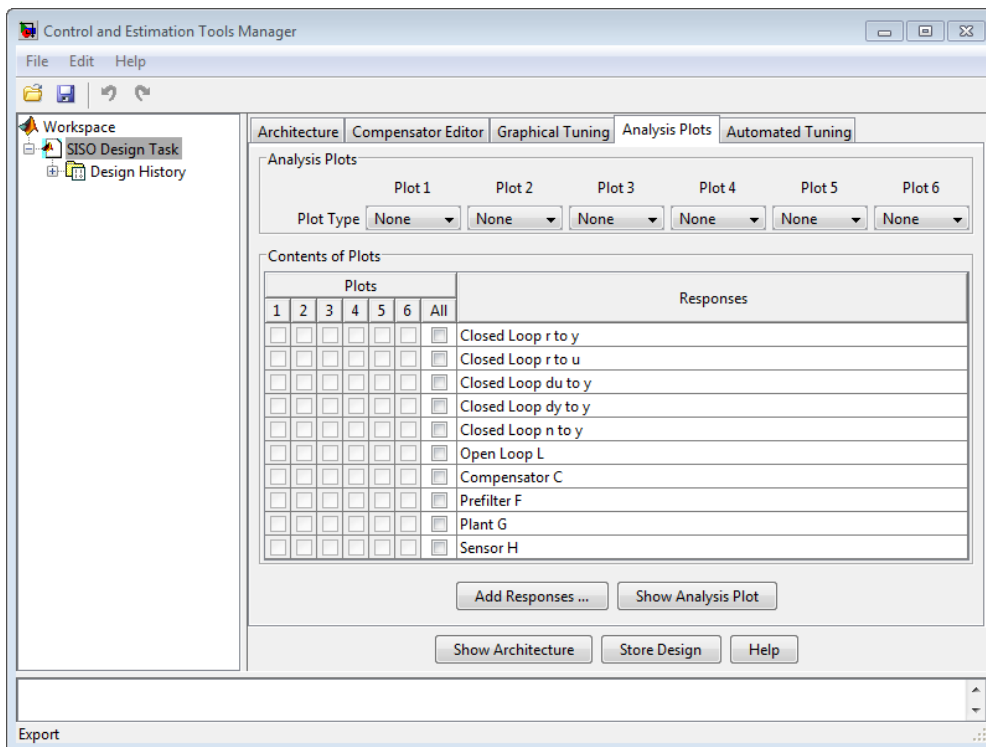
Analysis Plots Loop Response Selection
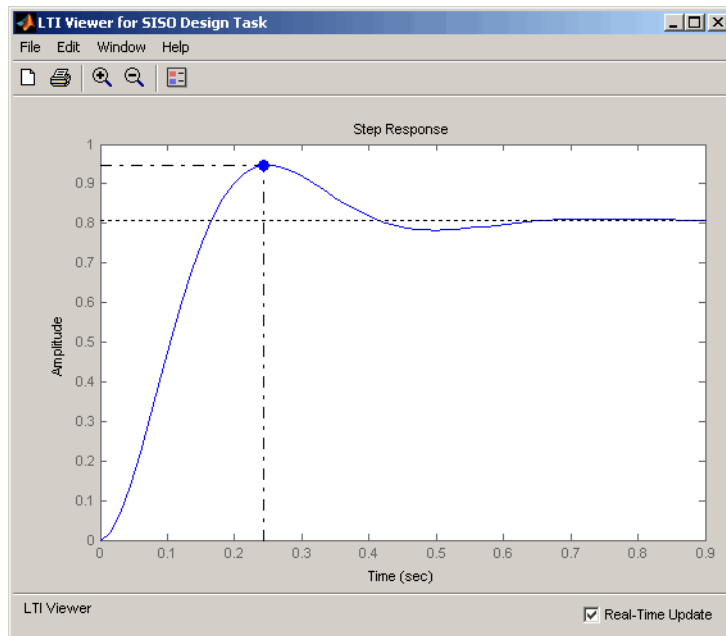
LTI Viewer Step Response for Compensator Gain = 112

The rise time is quite fast, about 0.15 second, but the overshoot is 18.4% and the steady-state is about 0.82.

Adding an Integrator

One approach to eliminating the steady-state error is to add an integrator.

To add an integrator:

1   Click the **Compensator Editor** tab to open the **Compensator Editor** page.

2   Right-click in the **Dynamics** table and select **Add Pole/Zero > Integrator**.

   This figure shows the process.
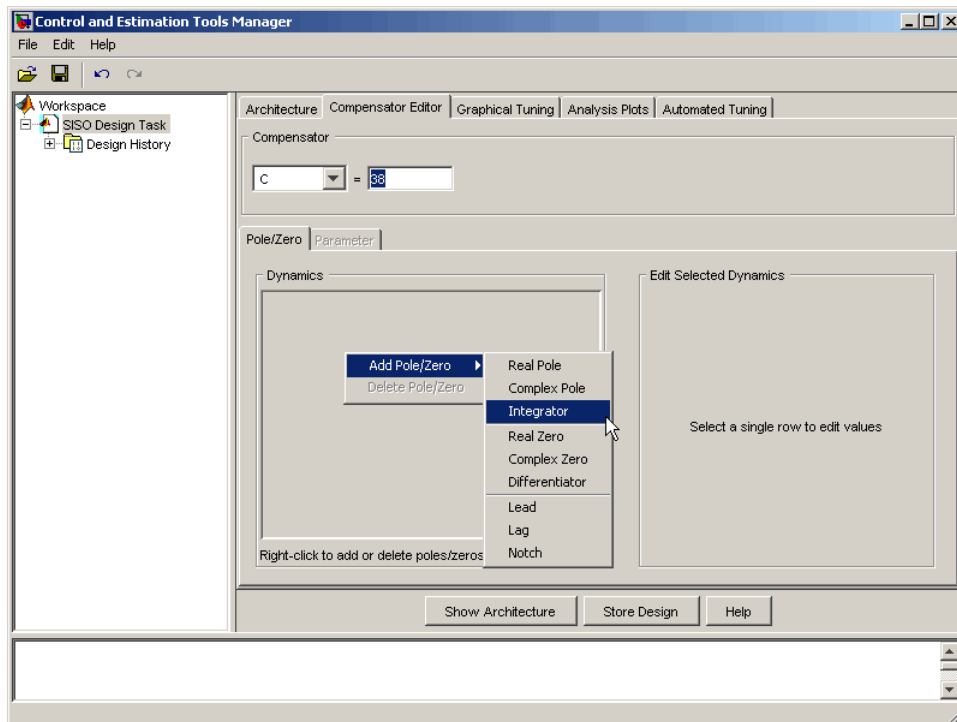
Adding an Integrator in the Dynamics Table

You can also add an integrator by selecting **Add Pole/Zero > Integrator** from the right-click menu in the Graphical Tuning window. When you add the integrator in this manner, it is automatically added to the **Dynamics** table on the **Compensator Editor** page.

Adding an integrator changes the gain margin from infinity to 10.5 dB. Since the gain and phase margins are now both finite, the Nichols plot shows a vertical line for the gain margin and a horizontal line for the phase margin.

The linked LTI Viewer automatically updates to show the new response.
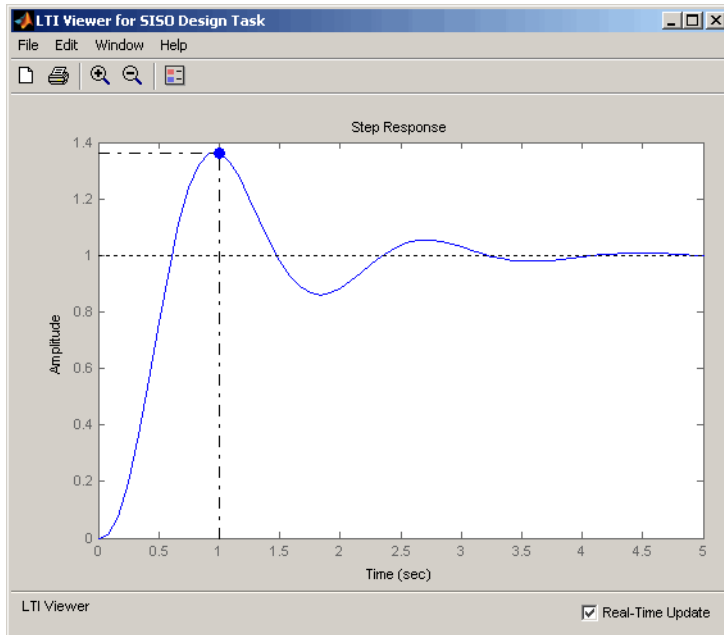
Step Response for a Compensator Consisting of a Gain and an Integrator

The steady-state value is now 1, which means the steady-state error has been eliminated, but the overshoot is 34% and the rise time is about 0.7 second. You must do more work to create a good design.

Adding a Lead Network

Improving the rise time requires that you increase the compensator gain, but increasing the gain further deteriorates the gain and phase margins while increasing the overshoot. You need to add a lead network to selectively raise the gain about the frequency crossover. To add the lead network:

1 Click the **Compensator Editor** tab to open the **Compensator Editor** page.

2 Right-click in the **Dynamics** table and select **Add Pole/Zero > Lead**.

This figure shows the process of adding a lead network on the **Compensator Editor** page.
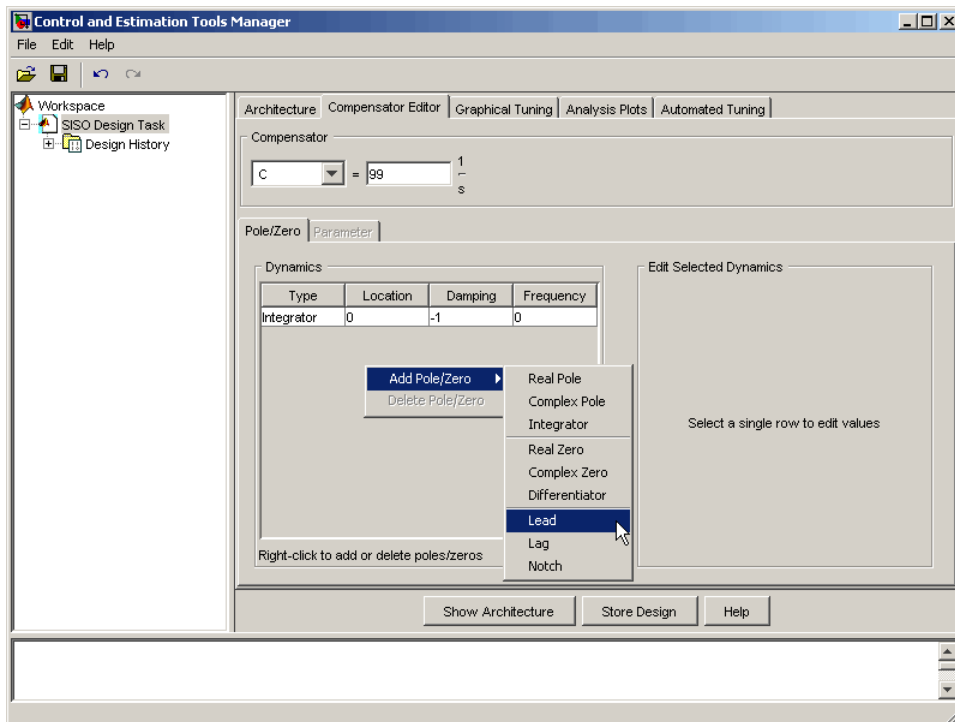
You can also add a lead network in the Graphical Tuning window. To add a lead network, select **Add Pole/Zero > Lead** from the right-click menu. Your cursor turns into a red `x'. Left-click along the Nichols curve to add the lead network. To move the lead network along the curve, left-click the pole or zero and drag.

You can track the pole's movement in the status bar at the bottom of the Graphical Tuning window. The status bar tells you the current location of the pole.

Using the **Compensator Editor** page, move the lead network pole to -28 and the zero to -4.3 for this design. The zero should be almost on top of the right-most pole of the plant (the DC motor model). Adjust the compensator gain to 84. This gives the final design.
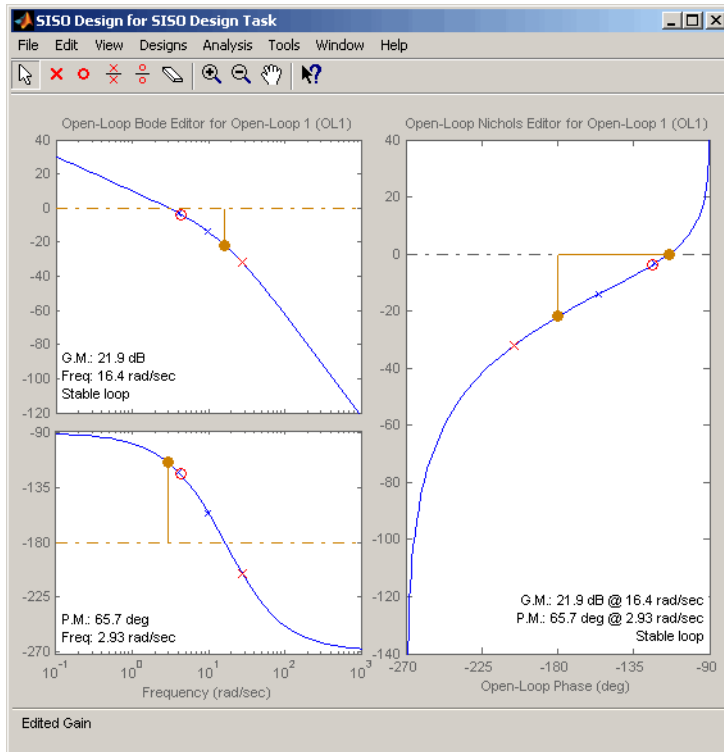
Final Nichols Plot Design for the DC Motor Compensator

The gain and phase margins are 21.9 dB and 65.7 degrees, respectively. Inspect the closed-loop step response to see if the rise time and overshoot requirements are met.
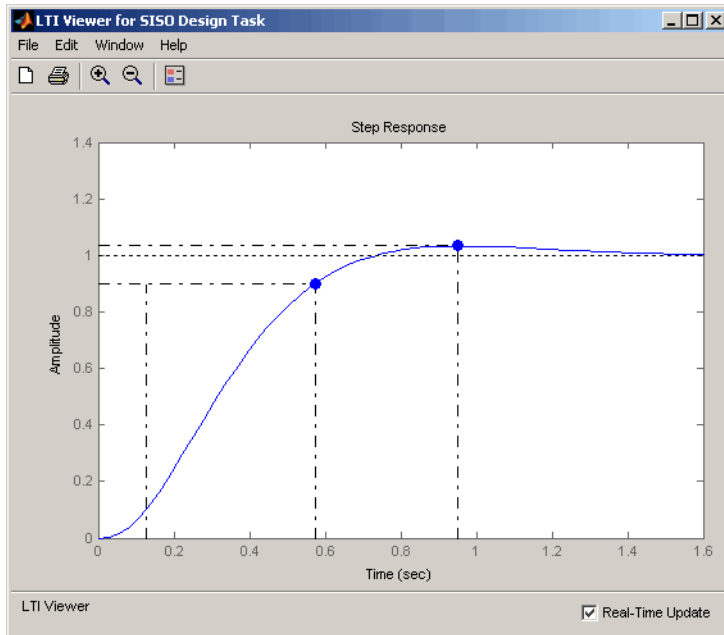
Closed-Loop Step Response for the Final Compensator Design

As this figure shows, the rise time is 0.448 second, and the overshoot is a little over 3%. This satisfies the rest of the design requirements.

Automated Tuning Design

In this section...
"Supported Automated Tuning Methods" on page 4-160
"Loading and Displaying the DC Motor Example for Automated Tuning" on page 4-160
"Applying Automated PID Tuning" on page 4-162

Supported Automated Tuning Methods

The SISO Design Tool simplifies the task of designing and tuning compensators. There are five automated tuning methods in the SISO Design Tool to help you design an initial stabilizing compensator for a SISO loop on-the-fly or refine existing compensator design so that it satisfies a certain user-defined design specification.

The available design methods are:

- Optimization-based tuning
- PID tuning
- Internal Model Control (IMC) tuning
- LQG synthesis
- Loop shaping

For a detailed discussion of these, see "Automated Tuning".

Loading and Displaying the DC Motor Example for Automated Tuning

Follow these steps to load and display the DC Motor example for automated tuning:

1 If you have not yet built the DC Motor example, type

    load ltiexamples

2 To open the SISO Design Tool and import the DC motor, type

    sisotool(sys_dc)

at the MATLAB prompt. This opens both the **SISO Design Task** node on the Control and Estimation Tools Manager and the Graphical Tuning window with `sys_dc` loaded.

3 Click the **Analysis Plots** tab to set the analysis plots. Select the plot type as `Step` for Plot 1. Then, check the box for plot 1 to the left of `Closed-Loop r to y`, as shown in the following figure, to open a linked LTI Viewer with the closed-loop step response from reference signal *r* to output signal *y*.
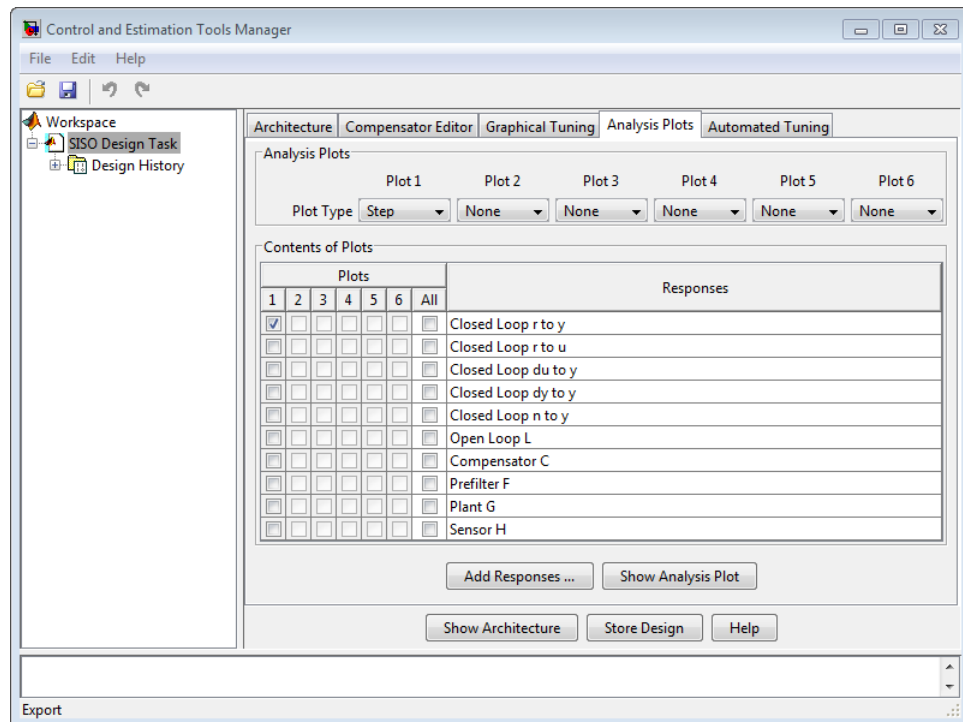
4 In the LTI Viewer that appears, use the right-click menu to add rise time and steady state values to your plot.
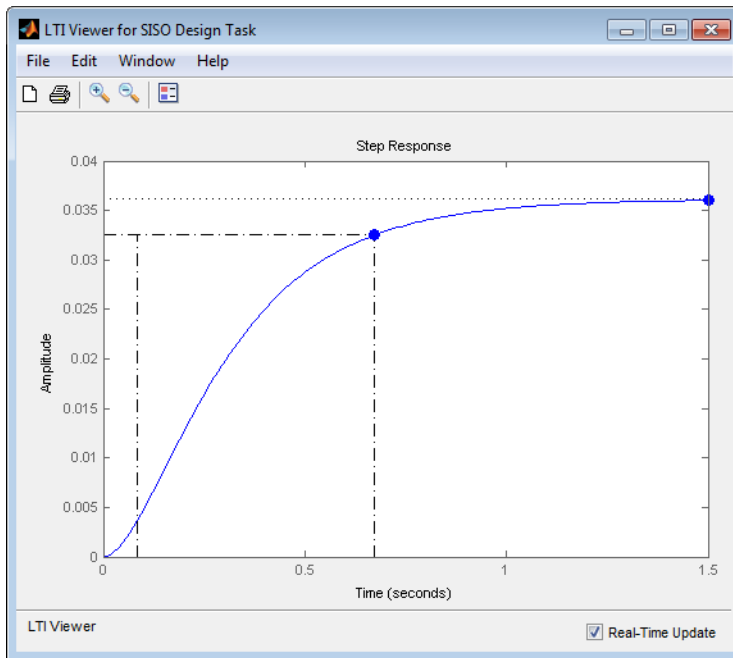
Step Response When Compensator = 1

Note that by default, the compensator is 1 and unit negative feedback is used (see "Architecture"). When a unit step is applied to the setpoint change, the steady state value of the system output is 0.0361, which is far from the setpoint, and its rise time is 0.589.

Applying Automated PID Tuning

1 Click the **Automated Tuning** tab.

2 Select PID tuning from the **Design method** list.

3 Leave C as the default compensator.
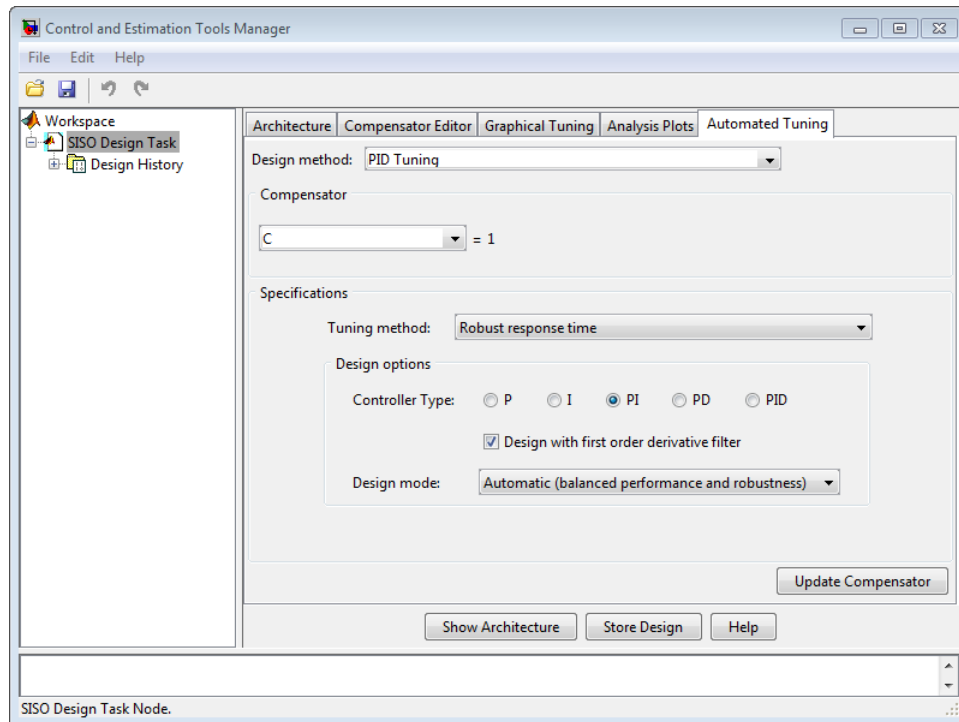
Design a Proportional-Only Controller

1 In the **Tuning method** menu, select `Classical design formulas`.

2 In the **Design options** area, select **P** for proportional-only control ($C = K_p$).

3 In the **Formula** menu, select `Ziegler-Nichols step response`.

Note: For more information about the automated tuning methods and formulas, see "Automated Tuning".

---

4 Click **Update Compensator**. The LTI Viewer is updated with the application of PID automated tuning with a proportional-only compensator. The compensator value is now 203.75.
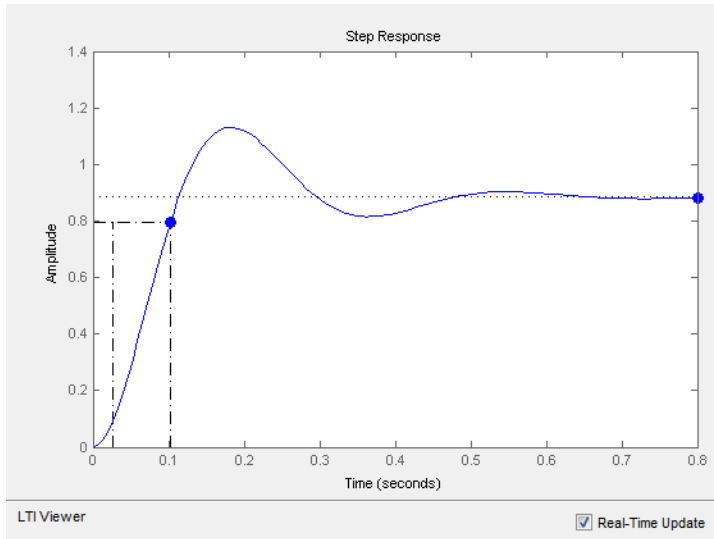
Note that the rise time is reduced to 0.0769 seconds, compared with 0.589 when C = 1. However, the steady state value of 0.885 can still be improved by setting the automated tuning controller type to `PI`.

Designing a Proportional-Integral Controller

1   In the **Design options** area, select **PI** for proportional-integral control

$(C = K_p + \dfrac{K_I}{s})$.

2   In the **Formula** menu, select `Ziegler-Nichols step response`.

3   Click **Update Compensator**. The LTI Viewer is updated with the application of PID automated tuning with a proportional-integral compensator. The compensator

value is now $1448.8 \times \dfrac{(1 + 0.13s)}{s}$.
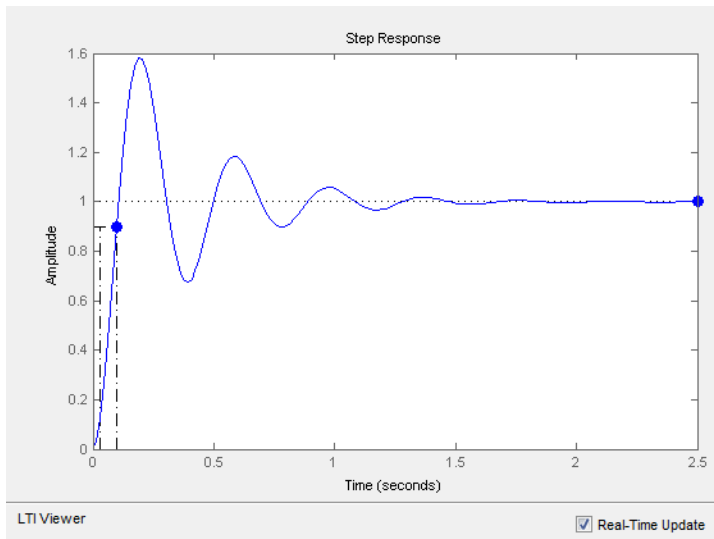
Now the steady state value is 1. Applying automated tuning using PID tuning set to PI guarantees zero offset.

This compensator design has a high degree of overshoot and ringdown. For further improvements on the result, try the Robust response time tuning method.

Multi-Loop Compensator Design

In this section...
"When to Use Multi-Loop Compensator Design" on page 4-166
"Workflow for Multi-Loop Compensator Design" on page 4-166
"Position Control of a DC Motor" on page 4-166

When to Use Multi-Loop Compensator Design

In many applications, a single-loop design is not feasible. If you have a design with inner loops, you can use the SISO Design Tool to design a compensator that meets your specifications.

Workflow for Multi-Loop Compensator Design

A typical procedure is to design the innermost loop on its own. You can use the SISO Design Tool to isolate the design on individual loops. When used this way, the tool ignores outer loop dynamics. Once the inner loop is designed, you can move on to the design of the outer loop compensator to achieve the desired closed-loop behavior. "Position Control of a DC Motor" on page 4-166 shows an example of this procedure.

Position Control of a DC Motor

Instead of controlling the angular rate of a DC motor, this example develops a control law for controlling the position (angle) of the motor shaft. The block diagram of the DC motor, as shown in the following figure, has an integrator added as an outer loop.
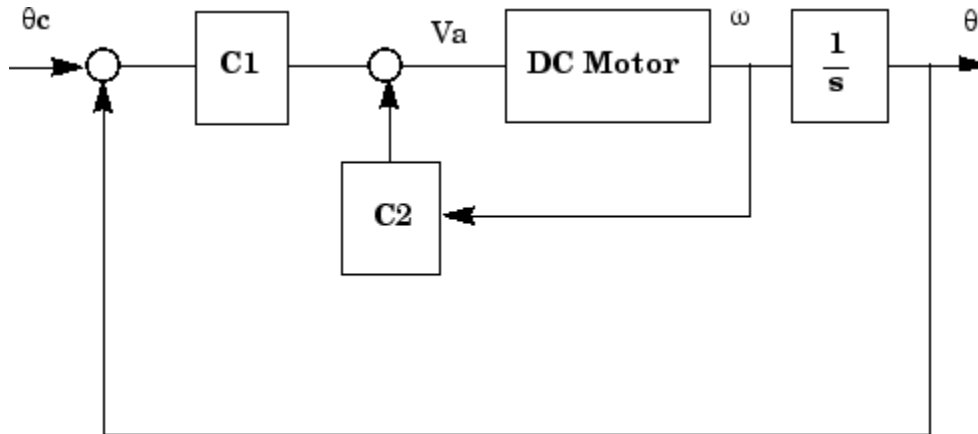
Block Diagram of the Position-Controlled DC Motor

The design goal for this example is the minimize the closed-loop step response settling time while maintaining an inner loop phase margin of at least 65º with maximum bandwidth.

For details on how to derive state-space and transfer function representations of a DC motor, see "SISO Example: The DC Motor" on page 2-3.

Designing a multi-loop compensator for a DC motor involves the following steps:

- "Developing a Mathematical Model of the DC Motor" on page 4-167
- "Selecting the Architecture and Importing the Model" on page 4-169
- "Designing the Inner Loop" on page 4-171
- "Tuning the Outer Loop" on page 4-173
- "Validating the Design with the LTI Viewer for SISO Design" on page 4-175

Developing a Mathematical Model of the DC Motor

These are the relevant physical constants:

```
R=2.0            % Ohms
L = 0.5          % Henrys
Km=0.1; Kb = 0.1 % Torque and back emf constants
Kf= 0.2;         % Nms
J = 0.02         % kg.m^2/s^2
```

4-167

First, construct a state-space model of the DC motor with one input, the applied voltage (Va). The output is the angular rate w.

```
h1 = tf(Km,[L,R]); % Armature
h2 = tf(1,[J, Kf]) % Equation of motion
dcm = ss(h2) *h1; % w = h2 cascaded with h1
dcm = feedback(dcm, Kb, 1, 1);% Closes back emf loop
```

Adapting the Model to SISO Tool Architecture

One possible choice for your architecture is this multi-loop configuration.
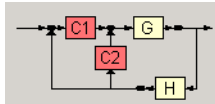
Comparing this architecture to the original Block Diagram of the Position-Controlled DC Motor, it is evident that the two do not match. Using block diagram algebra, you can manipulate the original figure into one that fits this architecture.
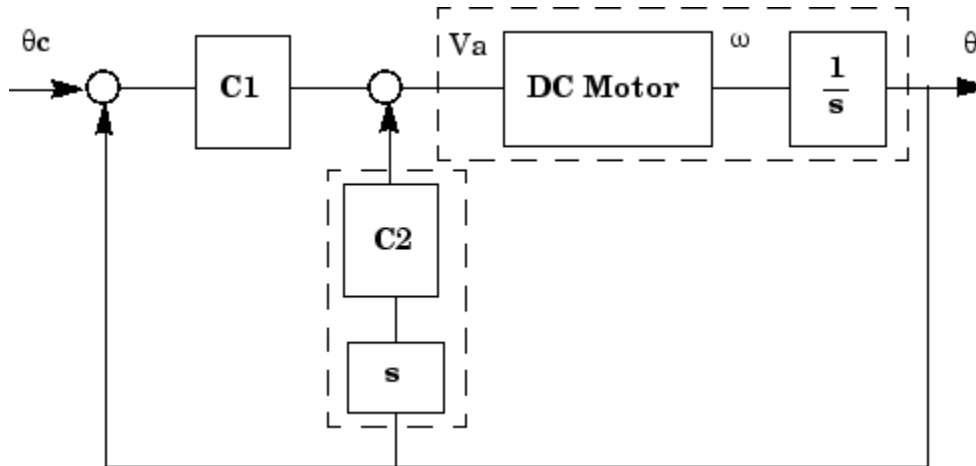
Position-Controlled DC Motor Rearchitected

To create this representation, add an integrator to get Θ, the angular displacement, and a pure differentiator in the inner loop's return path. The channel from Va to w is dcm(1), making it the appropriate channel for adding the integrator.

```
G = dcm*tf(1,[1,0]) % Motor with integrator; output is theta.
C2 = tf('s') % Differentiator
```

Selecting the Architecture and Importing the Model

Open the SISO Design Tool by typing

```
sisotool
```

at the MATLAB prompt. Once the Controls & Estimation Tools Manager opens, click **Control Architecture** on the **Architecture** page. Select the multi-loop configuration with two compensators, **C1** in the forward path and C2 in the inner feedback loop — located in the lower-right corner.
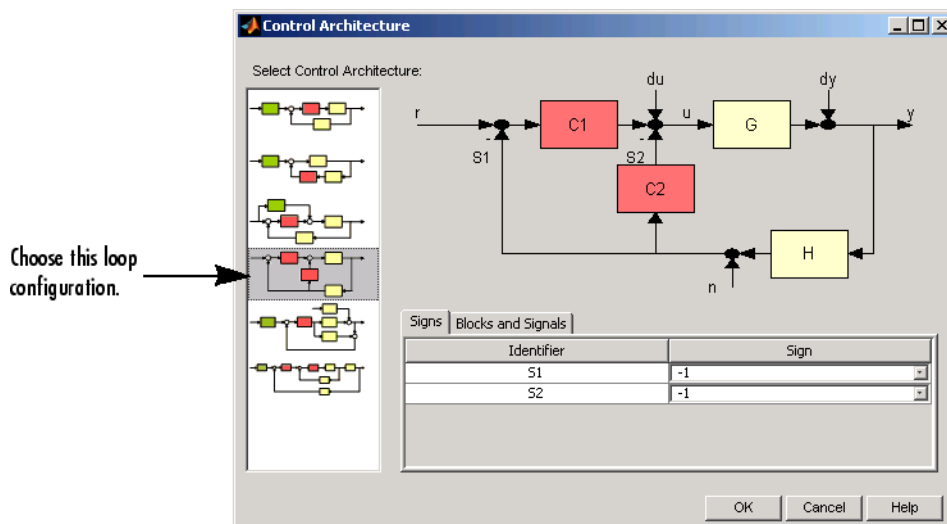
Control Architecture Window

Next, import the model parameters by clicking **System Data** on the **Architecture** tab. This opens the System Data dialog box. Set G to G from the workspace. Assume a perfect sensor and set H to 1. C1 and C2 are the gains you will use to design a compensator. Set C1 to 1 and C2 to C2 from the workspace. Your System Data dialog box should look like this.
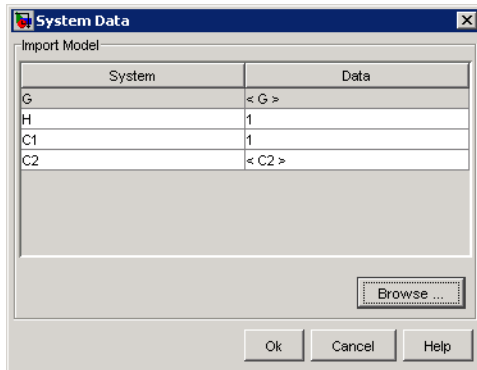
Selecting SISO Design Graphical Editor Views

Once you have selected the multi-loop architecture, click the **Graphical Tuning** tab. Set the plot types as follows:

1   Open-Loop 1 to "Root-Locus"

2   Open-Loop 2 to "Open-Loop Bode"

Your **Graphical Tuning** page should look like this.
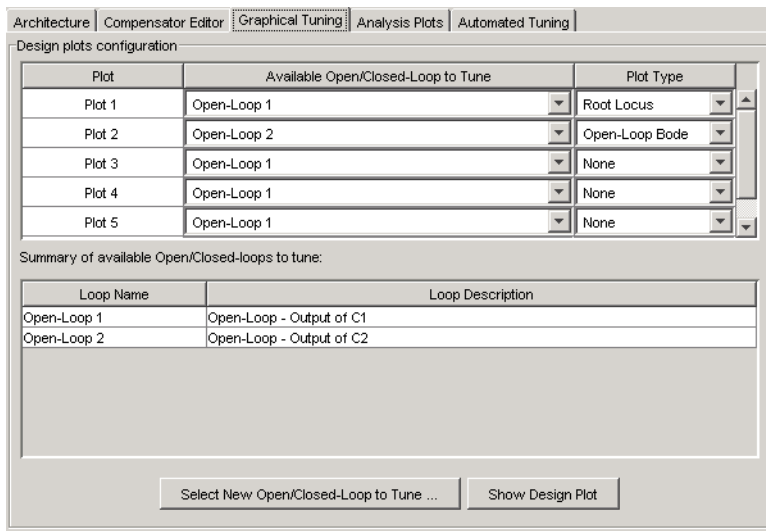
Graphical Tuning Page Set for DC Motor Multi-Loop Design

Click **Show Design Plot** to see the SISO Design Graphical editor.
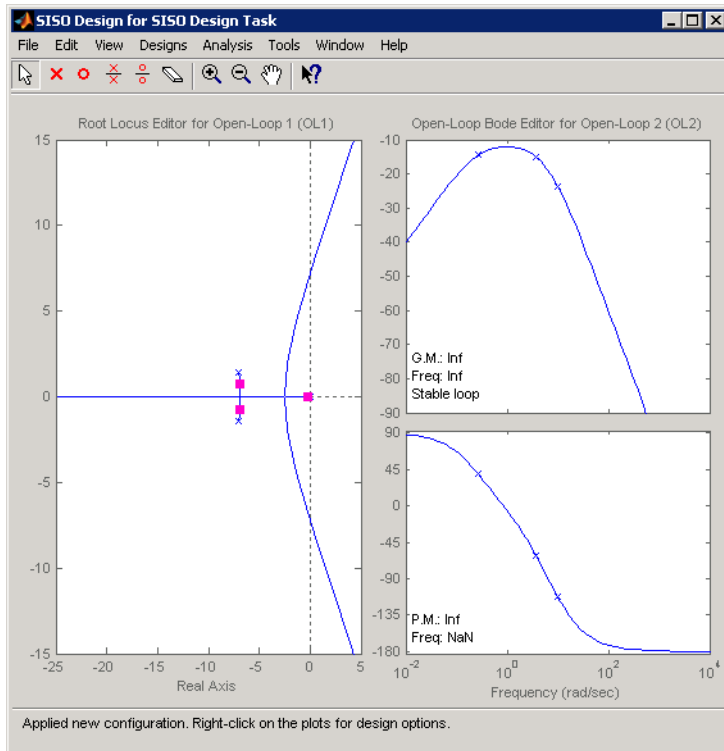
Designing the Inner Loop

You are now in a position to do the design. Start with the design of the inner loop. To do this, go to the **Architecture** page and remove the effects of the outer loop by following these steps:

1   Click **Loop Configuration**. This opens the Open-Loop Configuration dialog box.

2   From the pull-down menu, select **Open-Loop Output of C2**.

3   Click **Highlight feedback loop**. This opens a figure of the control architecture showing the loop configuration.
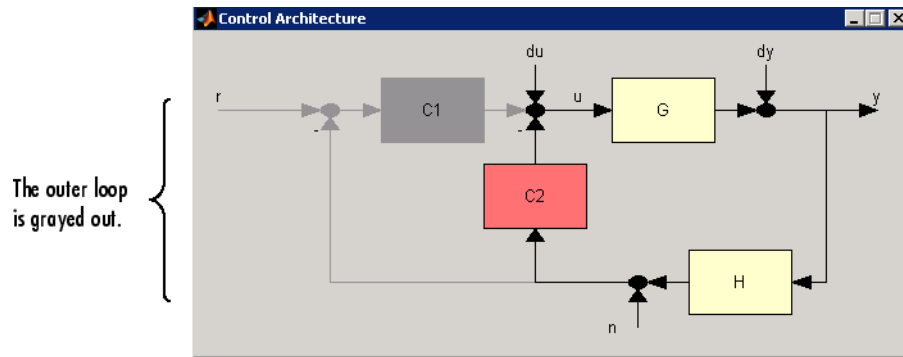
Notice how the C1 piece of the compensator and the outer loop are grayed out. This means that they will have no effect on the inner loop at this time.

Next, turn to the SISO Design Graphical editor. Use the Bode plot for open loop 2 (the inner loop) and increase the gain to maximize bandwidth subject to a 65° phase margin. This turns out to be a gain of about 16.1 for C2.
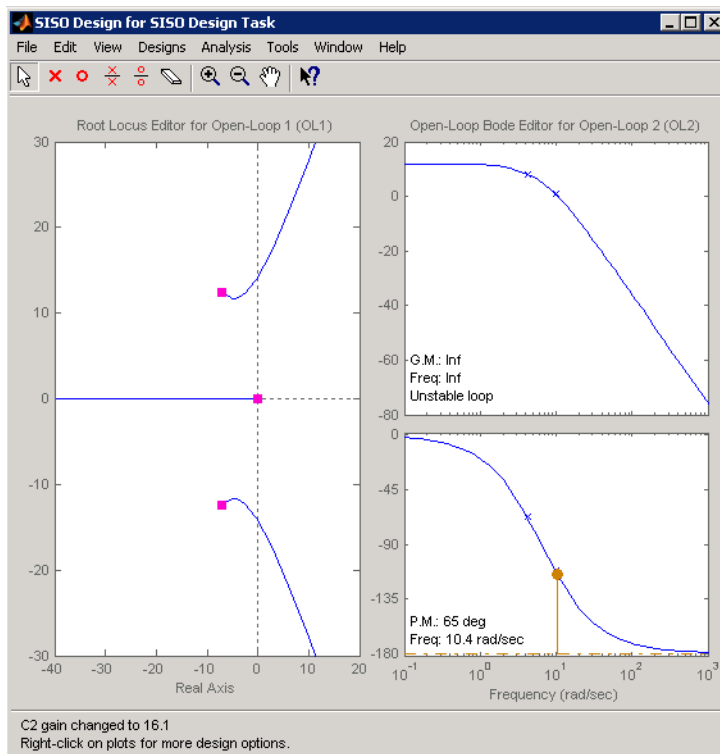
Setting the Inner Loop Gain

This finishes the design of the inner loop.

Tuning the Outer Loop

The goal in designing the outer loop is to minimize the settling time. Note that the outer loop can "see" the inner loop, so that the tuning affects the entire system. Follow these steps:

1   Go to the **Analysis Plot** tab in the Controls & Estimation Tools Manager. Select the **Closed-Loop r to y** check box.

2   Select **Step** from the **Plot 1** pull-down menu. This opens the LTI Viewer for SISO Design.

3   Right-click in the step response plot and select **Characteristics>Settling Time**. Your LTI Viewer should look like this.
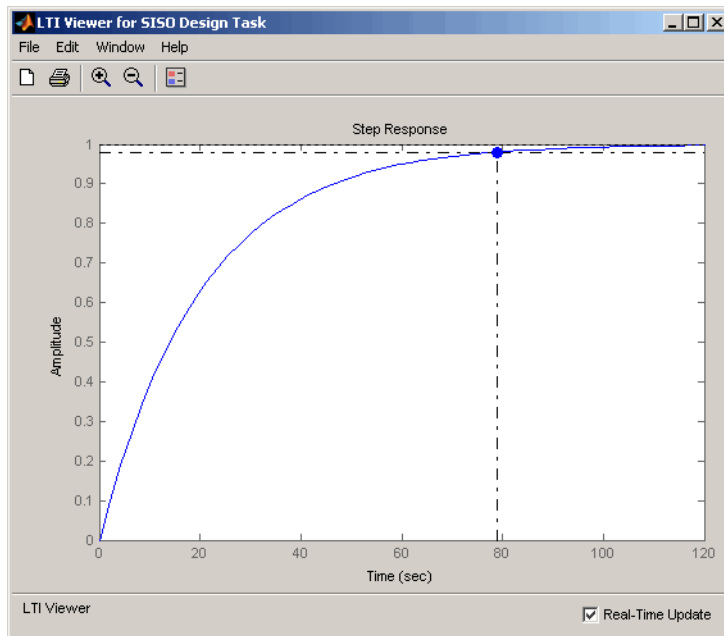
Initial Step Response with Settling Time

The settling time is about 79 s.

Return to the SISO Design Graphical editor and increase the gain of C1 in the root locus plot. At a gain of about 90.2, you will see the complex pair of poles move toward a slower time constant as the third pole moves toward a faster one. You can view the trade-off in the LTI Viewer for SISO Design. As the gain is changed, the closed-loop step response changes.

The 90.2 gain seems to yield a good compromise between rise and settling time.
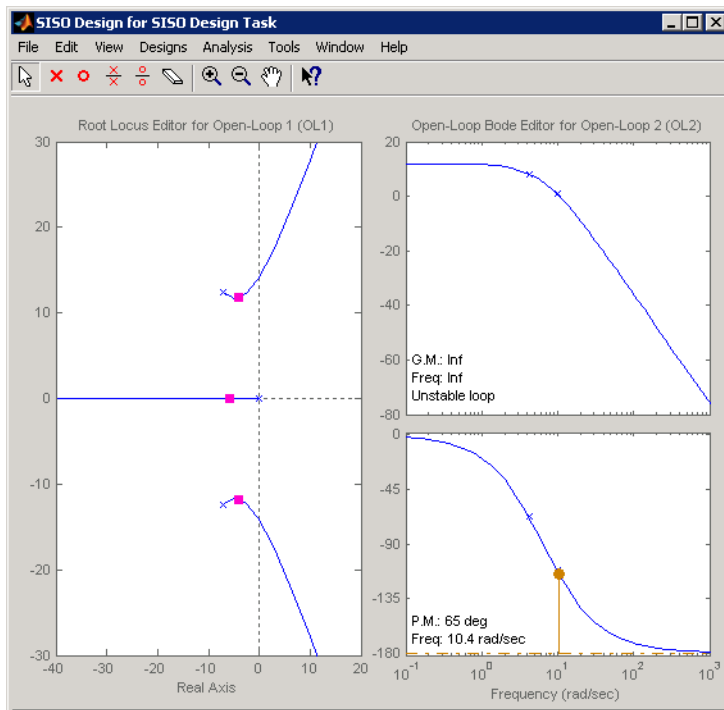
Final Gain Choice for C1

Validating the Design with the LTI Viewer for SISO Design

Turning back to the LTI Viewer for SISO Design, it is evident that the settling time is now much lower than the original 78.9 s.
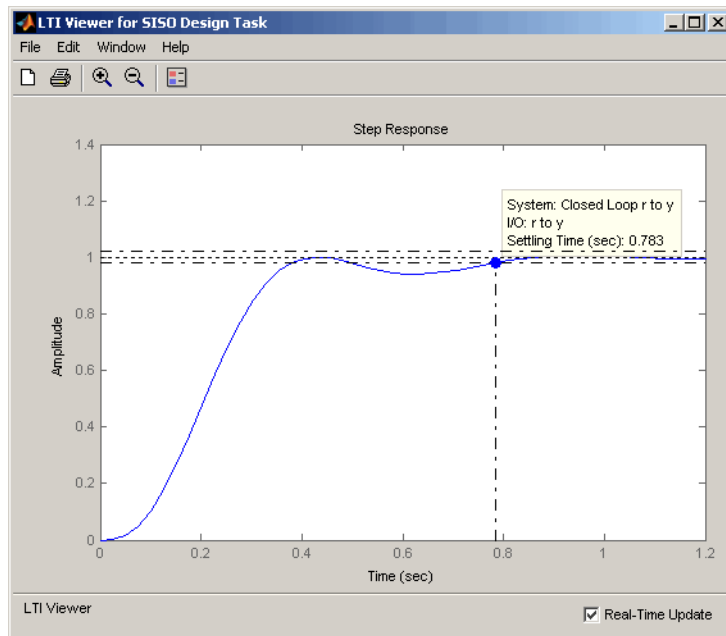
With a settling time of about 0.8 s, and a phase margin of 65° in the inner loop, the design is complete.

Control Design Analysis of Multiple Models

In this section...
"Multiple Models Represent System Variations" on page 4-177
"Control Design Analysis Using the SISO Design Tool" on page 4-177
"Specifying a Nominal Model" on page 4-178
"Frequency Grid for Multimodel Computations" on page 4-180
"How to Analyze the Controller Design for Multiple Models" on page 4-181

Multiple Models Represent System Variations

Typically, the dynamics of a system are not exactly known and may vary. For example, system dynamics can vary because of:

- Parameter value variations caused by manufacturing tolerances. For example, the resistance value of a resistor is typically within a range about the nominal value, 5 Ω +/– 5%.

- Operating conditions. For example, aircraft dynamics change based on its altitude and speed.

The controller you design for such a system must satisfy design requirements for dynamics of all models.

Control Design Analysis Using the SISO Design Tool

To design a controller for a system whose dynamics vary, you sample the variations, create an LTI model for each sample and use the set of models to build an "array of LTI models". Then, design the controller for a representative model from the array.

Control design analysis of multiple models in the SISO Design Tool requires you to specify either the plant G or sensor H or both as row or column arrays of LTI models. If both G and H are arrays, their sizes must match.

Use the SISO Design Tool to:

1   Choose a nominal model from the array of LTI models.

2   (Optional) Specify a frequency grid for multimodel computations.

3   Design a controller for the nominal model.

4   Analyze if the controller meets the design requirements for all models in the array.

If the controller design does not meet design requirements on all the models, specify a different nominal model and redesign the controller. For more information, see "How to Analyze the Controller Design for Multiple Models" on page 4-181.

Specifying a Nominal Model

- "What Is a Nominal Model?" on page 4-178
- "How Is a Nominal Model Computed?" on page 4-178
- "Criteria for Choosing a Nominal Model" on page 4-180

What Is a Nominal Model?

The *nominal model* is a representative model in the array of LTI models that you use to design the controller or for loop shaping in the SISO Design Tool.

You use the design and analysis plots to visualize and analyze the effect of the controller on the remaining plants in the array.

How Is a Nominal Model Computed?

The following table summarizes how the software computes the nominal model when the plant G and sensor H are arrays of LTI models in a control architecture.

Array of LTI Models	Nominal Model

Both G and H
 	
 Note: The sizes of both arrays must match.
	• By default, computed using the first element in both arrays.
	 • For a different index, computed using the specified index in both arrays.
	    • The same index is also used when you import new arrays in the SISO Design Tool.
	    • If the specified index does not exist, the nominal model index reverts to a value you previously specified or the first element.

Array of LTI Models	Nominal Model
Only G or H	Uses scalar expansion for the specified index and G or H value.

For example, in the following default control architecture:
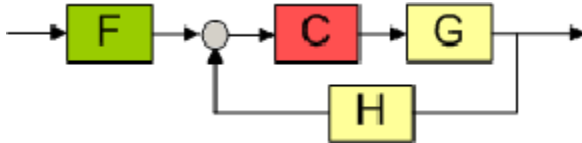
if

- G and H are arrays of LTI models of length 3
- Nominal model index is 2

the software uses the second element in both the arrays to compute the nominal model:

Nominal Model
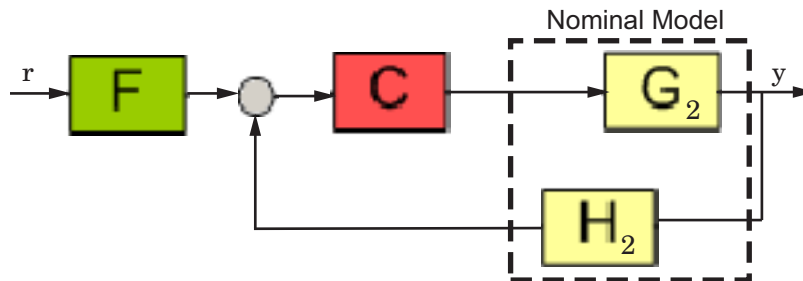
The nominal response from r to y is:

$$T = \frac{CG_{nom}}{1 + CG_{nom}H_{nom}},$$

where $G_{nom} = G_2$, $H_{nom} = H_2$ and $G_{nom}H_{nom}$ is the open-loop response.

The software also computes and plots the responses showing the effect of C on the remaining pairs of plant and sensor combinations—G_1H_1 and G_3H_3.

If only *G* is an array of LTI models, and the specified nominal model is 2, then the control architecture for nominal response is:
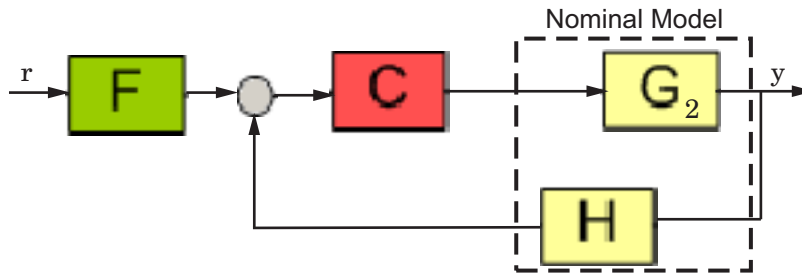
The software also computes and plots the responses showing the effect of *C* for the remaining pairs of plant indices and sensor—G_1H and G_3H.

Criteria for Choosing a Nominal Model

The nominal model is the model that you design a controller for. Typically, you choose a model that:

- Represents an average of the multiple models. For example, the open-loop response of the model lies midway among the responses of all models in the array.
- Represents a worst-case plant.
- Lies closet to the stability point.
- You want to work with for control design.

Tip You can plot and analyze the open-loop dynamics of the system on a Bode plot to determine which model to choose as nominal.

If the controller design for the nominal model does not meet the design requirements on the remaining plants in the array, you can specify a different nominal model and redesign the controller. See "How to Analyze the Controller Design for Multiple Models" on page 4-181 for more information.

Frequency Grid for Multimodel Computations

- "Algorithm for Frequency Grid Computation" on page 4-181

- "When to Specify Custom Frequency Grid" on page 4-181

Algorithm for Frequency Grid Computation

The frequency response of a system is computed at points in the frequency, called *frequency grid*.

To compute the frequency grid, the software computes a logarithmic equally spaced grid, based on the dynamics (dynamic range) of each model in the array.

When to Specify Custom Frequency Grid

You can specify a custom frequency grid if:

- The automatic grid does not capture the system dynamics sufficiently.

  This happens because the grid is not sufficiently dense in a particular frequency range. For example, for an underdamped systems the model response shows sharp and tall peaks. Examine the analysis plots to verify if these dynamics are captured in the response. If the response does not capture these dynamics, specify a denser gridding.
- The grid computed automatically has more points in the response than you require.
- You are interested only in a specific frequency range in the response.

Tip Specifying a less dense grid reduces the number of computations and is less expensive computationally.

For more information, see "How to Analyze the Controller Design for Multiple Models" on page 4-181.

How to Analyze the Controller Design for Multiple Models

Prerequisites

An array of LTI models that model variations in the plant *G*, sensor *H* or both. *G* and *H* must be row or column vectors of the same length.

Obtain the array using one of the following methods:

- Create multiple LTI models using the `tf`, `ss`, `zpk`, or `frd` commands. For example:

```
% Specify system parameters.
m = 3;
b = 0.5;
k = 8:1:10;
T = 0.1:.05:.2;

% Create an array of LTI models to model plant (G) variations.
for ct = 1:length(k);
    G(:,:,ct) = tf(1,[m,b,k(ct)]);
end

% Create an array of LTI models to model sensor (H) variations.
for ct = 1:length(T);
    H(:,:,ct) = tf(1,[1/T(ct), 1]);
end
```

In this case, *G* and *H* each contain 3 elements.

- Create an array of LTI models using the `stack` command. For example:

```
% m = 3, b = 0.5, k = 8:1:10.
% Create an array of LTI models to model plant (G) variations.
G1 = tf(1, [1 1 8]);
G2 = tf(1, [1 1 9]);
G3 = tf(1, [1 1 10]);
% Create an array of LTI models
G = stack(1, G1, G2, G3);

 % Create an array of LTI models to model sensor (H) variations.
H1 = tf(1,[1/0.1, 1]);
H2 = tf(1,[1/0.15, 1]);
H3 = tf(1,[1/0.2, 1]);
% Create array of LTI models.
H=stack(1,H1,H2,H3);
```

- (Requires Simulink Control Design software) Perform batch linearizations at multiple operating points. Then export the computed LTI models to create an array of LTI models. See the example Reference Tracking of a DC Motor with Parameter Variations.

- (Requires Simulink Control Design and Robust Control Toolbox™ software) Specify a core Simulink block to linearize to an uncertain state-space (`uss` or `ufrd`) model. See "Specify Uncertain Linearization for Core or Custom Simulink Blocks".

- (Requires Robust Control Toolbox software) Compute uncertain state-space (`uss`) models from Simulink models. Then use `usubs` or `usample` to create an array of LTI models. See "Obtain Uncertain State-Space Model from Simulink Model".

1 Open SISO Design Tool.

```
sisotool(G,1,H)
```

This command opens the following window.
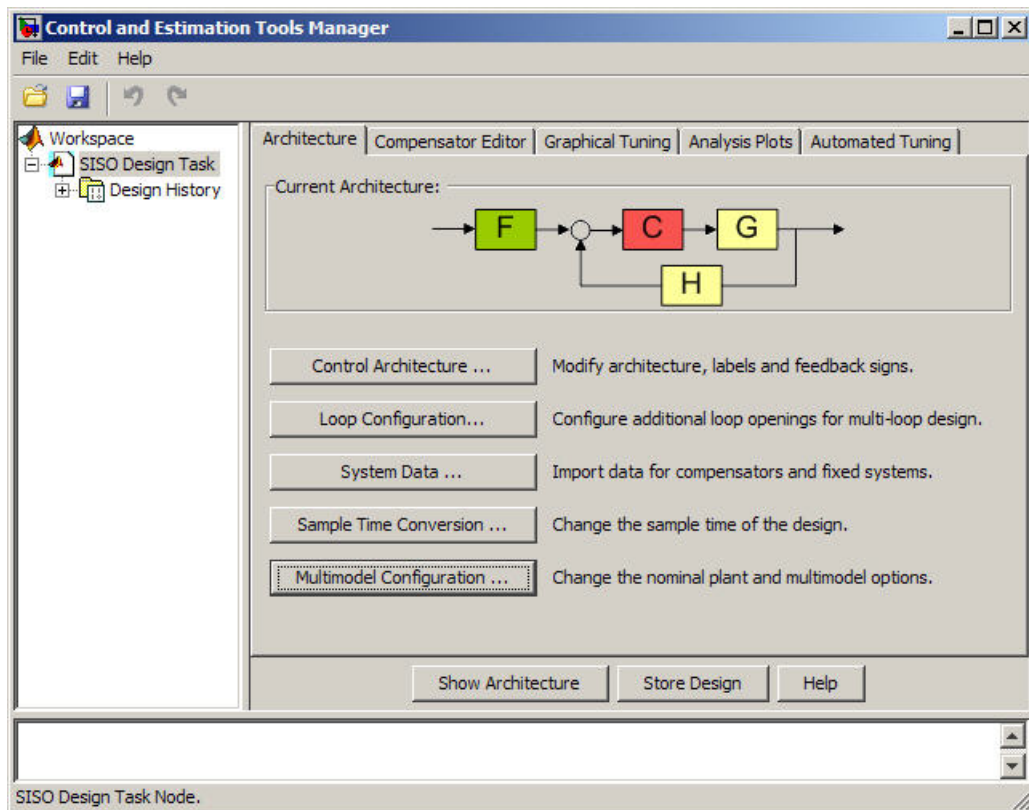
By default, the combination of the first plant and sensor in the arrays is the *nominal model* on which you perform the control design. For more information, see "How Is a Nominal Model Computed?" on page 4-178.

The Graphical Tuning window, which opens with the SISO Design Tool, shows the individual responses of all models in the arrays.
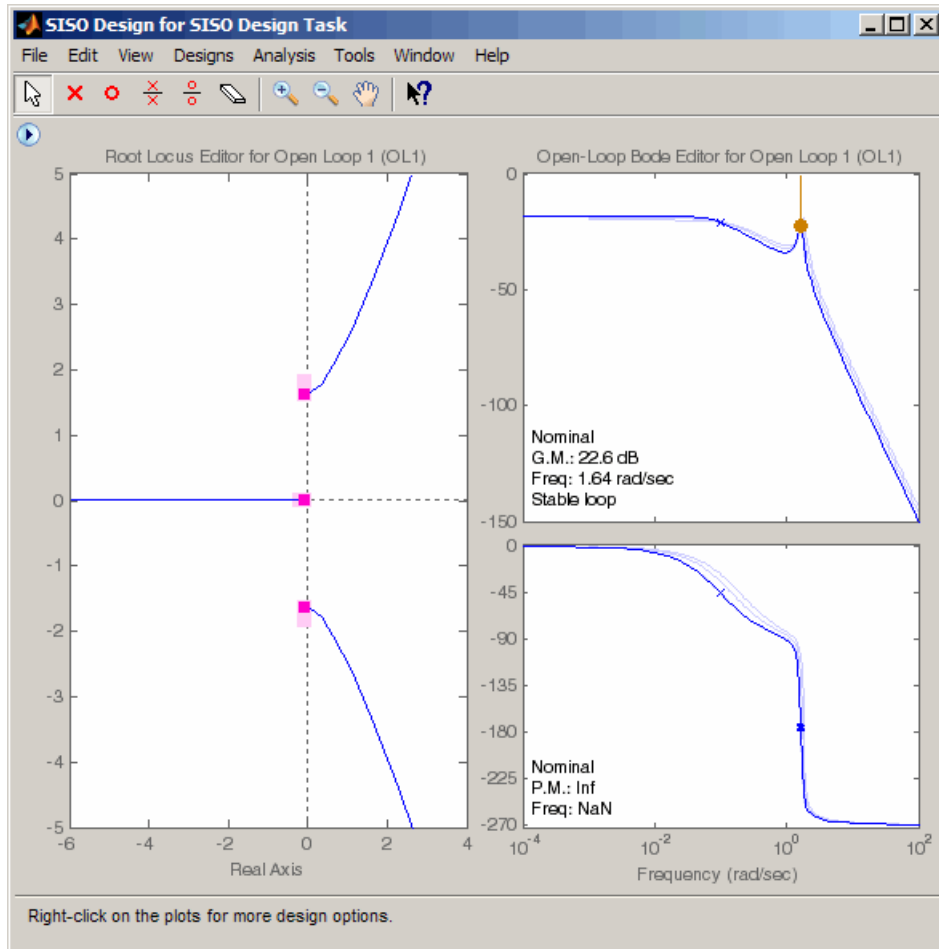
Tip You can view the envelope of the Bode response instead by right-clicking the plot and selecting **Multimodel Display** > **Bounds**. See "Using the Graphical Tuning Window" on page 4-91 for more information.

Alternatively, to view the responses of all models in the arrays:

- Open an empty SISO Design Tool by typing `sisotool` and import *G* and *H*. See "Importing Models into the SISO Design Tool" on page 4-80 for more information.

- Configure SISO Design Tool using `sisoinit` with the arrays of LTI models and then open the SISO Design Tool.

2 Configure the analysis plots in the **Analysis Plots** tab in Control and Estimation Tools Manager.

By default, the plots show only the nominal response.
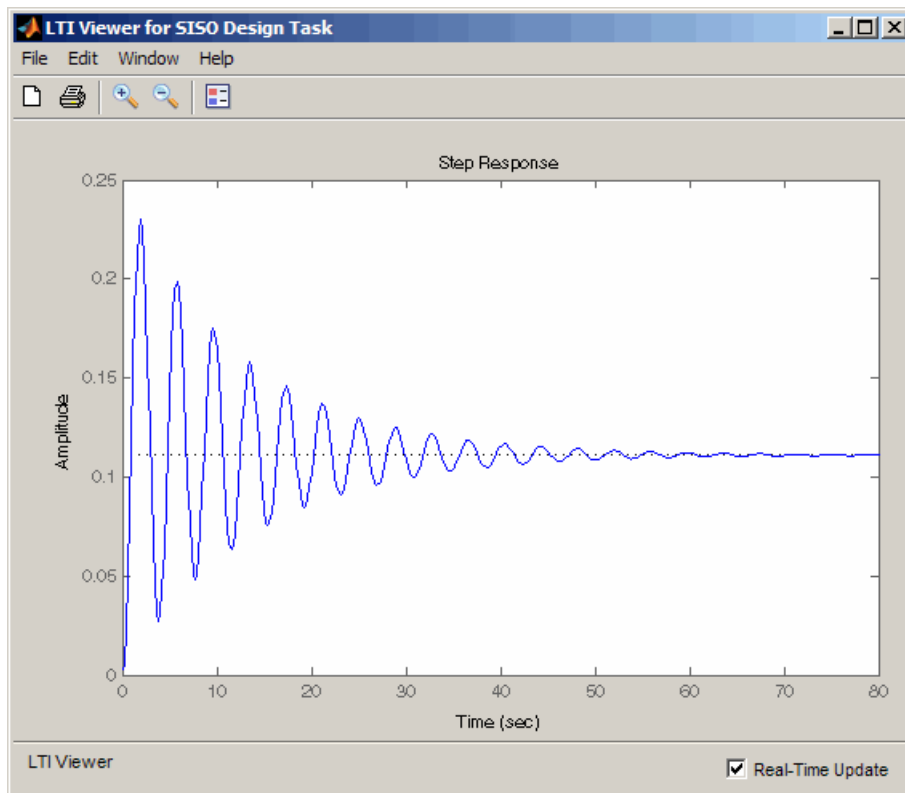
Right-click the plot, and select **Multimodel Configuration** > **Bounds** or **Multimodel Configuration** > **Individual Responses** to see the individual response or envelope of all models, respectively.
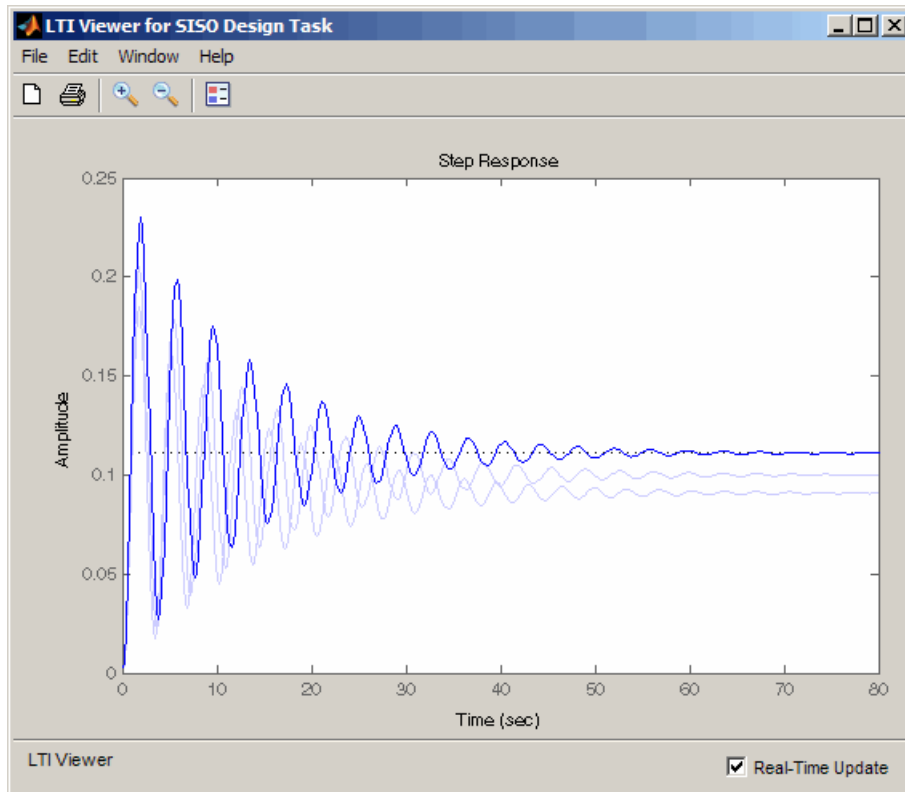
See "Analysis Plots for Loop Responses" on page 4-86 for more information.

3 (Optional) If you want to specify a different nominal plant, click **Multimodel Configuration** in the **Architecture** tab.

The Multimodel Configuration Dialog window opens.
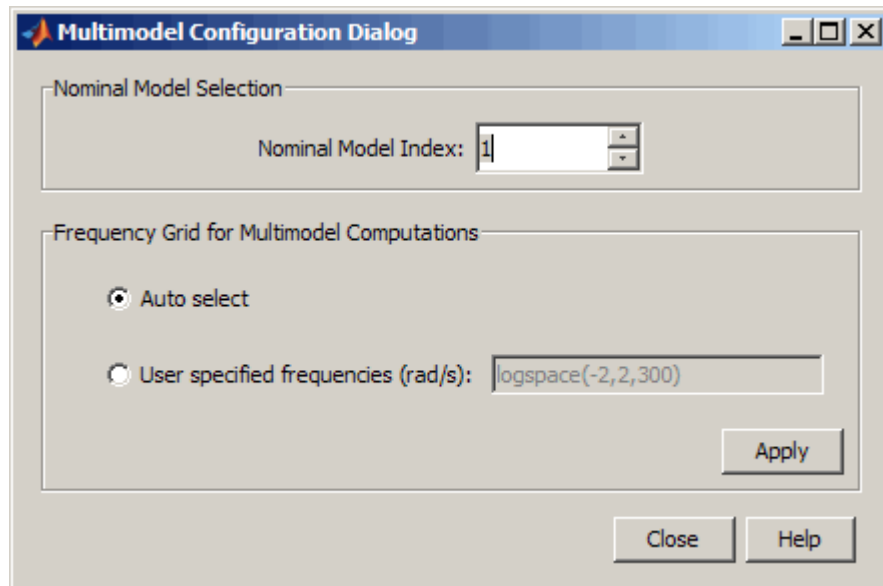
Specify a different nominal model in **Nominal Model Index** field. If you specify an index value greater than the maximum index of the arrays, the field reverts back to a value you specified previously or 1.

Tip You can keep the Multimodel Configuration Dialog window open as you are following the next steps.

---

4 (Optional) If the grid computed by **Auto select** is not dense enough to capture all system dynamics, specify a different frequency grid.

   a Select the **User specified frequencies (rad/sec)** option, and enter the frequency grid in log scale.

   b Click **Apply**.

For more information, see "Frequency Grid for Multimodel Computations" on page 4-180.

5 Design the controller for the nominal plant using graphical or automated tuning.

For more information on designing a controller, see the following topics:

- "Bode Diagram Design" on page 4-103
- "Root Locus Design" on page 4-129
- "Nichols Plot Design" on page 4-150
- "Automated Tuning Design" on page 4-160

As you design the controller, use the design and analysis plots analyze to analyze the controller's effects on the remaining models in the array.

6 (Optional) If the controller design for the nominal model does not meet the design requirements for the remaining models in the array:

a Specify a different nominal model in the **Nominal Model Index** field of the Multimodel Configuration Dialog window.
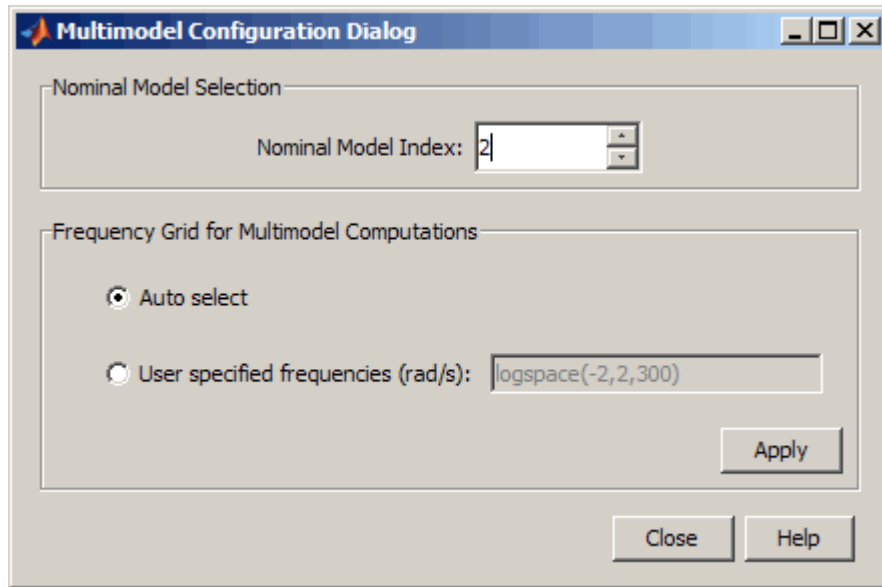
The design and analysis plots update to show the updated nominal model. For example, for a nominal model value of 2, the plots appear as shown in the next figures.
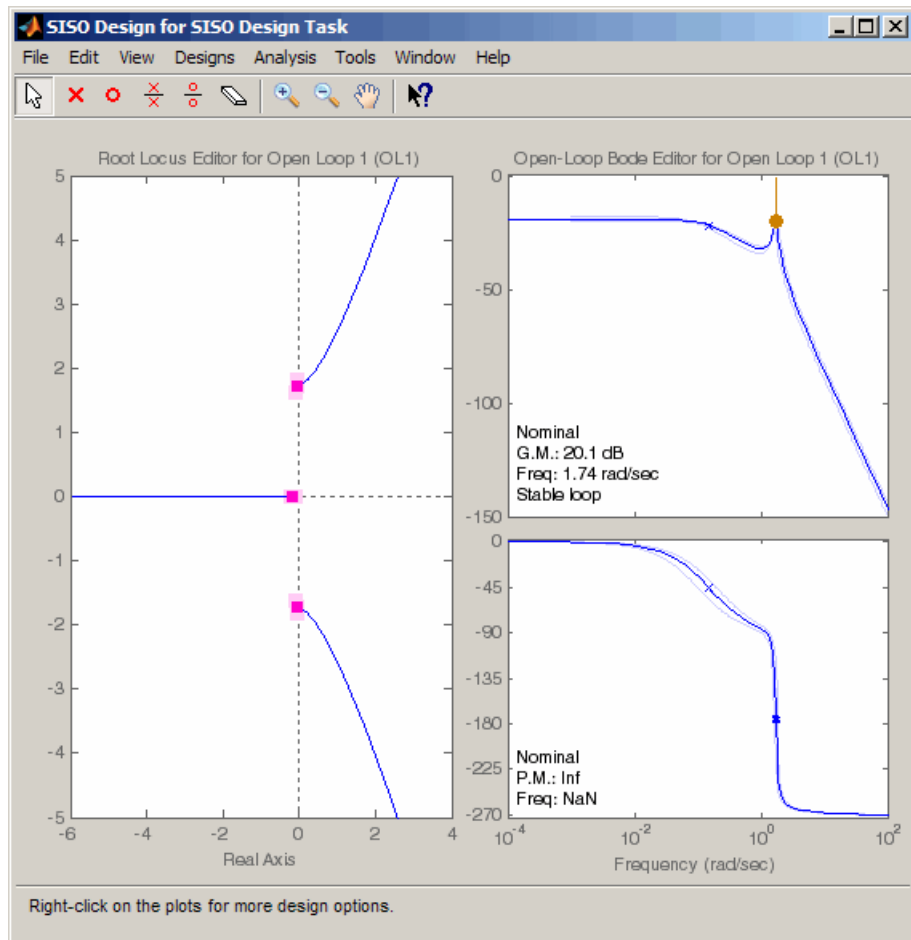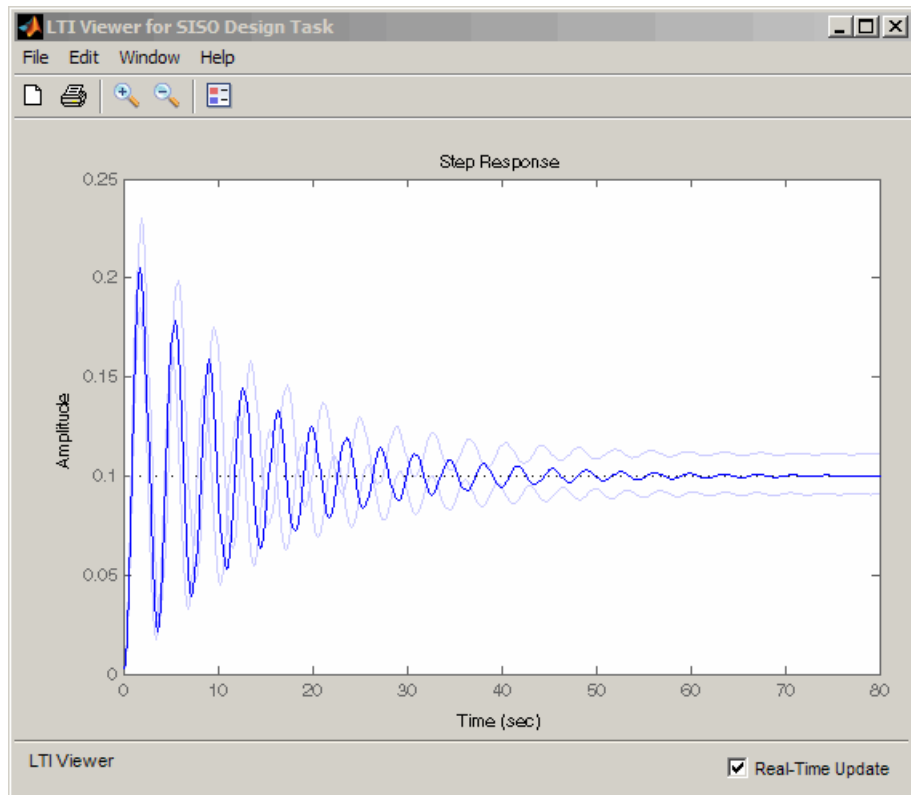
b Redesign the controller iteratively.

Functions for Compensator Design

In this section...

When to Use Functions for Compensator Design

The term control system *design* refers to the process of selecting feedback gains that meet design specifications in a closed-loop control system. Most design methods are iterative, combining parameter selection with analysis, simulation, and insight into the dynamics of the plant.

In addition to the SISO Design Tool, you can use functions for a broader range of control applications, including

- Classical SISO design
- Modern MIMO design techniques such as pole placement and linear quadratic Gaussian (LQG) methods

Root Locus Design

The following table summarizes the functions for designing compensators using root locus design techniques.

Function	Description
pzmap	Pole-zero map
rlocus	Evans root locus plot
sgrid	Continuous ω_n, ζ grid for root locus plots

4-191

Function	Description
sisotool	Root Locus Design GUI
zgrid	Discrete ω_n, ζ grid for root locus plots

Pole Placement

The closed-loop pole locations have a direct impact on time response characteristics such as rise time, settling time, and transient oscillations. Root locus uses compensator gains to move closed-loop poles to achieve design specifications for SISO systems. You can, however, use state-space techniques to assign closed-loop poles. This design technique is known as *pole placement*, which differs from root locus in the following ways:

- Using pole placement techniques, you can design dynamic compensators.
- Pole placement techniques are applicable to MIMO systems.

Pole placement requires a state-space model of the system (use ss to convert other model formats to state space). In continuous time, such models are of the form

$$\dot{x} = Ax + Bu$$
$$y = Cx + Du$$

where u is the vector of control inputs, x is the state vector, and y is the vector of measurements.

State-Feedback Gain Selection

Under state feedback $u = -Kx$, the closed-loop dynamics are given by

$$\dot{x} = (A - BK)x$$

and the closed-loop poles are the eigenvalues of *A-BK*. Using the place function, you can compute a gain matrix K that assigns these poles to any desired locations in the complex plane (provided that (*A,B*) is controllable).

For example, for state matrices *A* and *B*, and vector *p* that contains the desired locations of the closed loop poles,

```
K = place(A,B,p);
```

computes an appropriate gain matrix *K*.

State Estimator Design

You cannot implement the state-feedback law $u = -Kx$ unless the full state x is measured. However, you can construct a state estimate ξ such that the law $u = -K\xi$ retains similar pole assignment and closed-loop properties. You can achieve this by designing a state estimator (or observer) of the form

$$\dot{\xi} = A\xi + Bu + L(y - C\xi - Du)$$

The estimator poles are the eigenvalues of *A-LC*, which can be arbitrarily assigned by proper selection of the estimator gain matrix *L*, provided that (*C, A*) is observable. Generally, the estimator dynamics should be faster than the controller dynamics (eigenvalues of *A-BK*).

Use the `place` function to calculate the *L* matrix

```
L = place(A',C',q).'
```

where `A` and `C` are the state and output matrices, and `q` is the vector containing the desired closed-loop poles for the observer.

Replacing *x* by its estimate ξ in $u = -Kx$ yields the dynamic output-feedback compensator

$$\dot{\xi} = [A - LC - (B - LD)K]\xi + Ly$$
$$u = -K\xi$$

Note that the resulting closed-loop dynamics are

$$\begin{bmatrix} \dot{x} \\ \dot{e} \end{bmatrix} = \begin{bmatrix} A - BK & BK \\ 0 & A - LC \end{bmatrix} \begin{bmatrix} x \\ e \end{bmatrix}, \quad e = x - \xi$$

Hence, you actually assign all closed-loop poles by independently placing the eigenvalues of *A-BK* and *A-LC*.

Example

Given a continuous-time state-space model

```
sys_pp = ss(A,B,C,D)
```

with seven outputs and four inputs, suppose you have designed

- A state-feedback controller gain K using inputs 1, 2, and 4 of the plant as control inputs
- A state estimator with gain L using outputs 4, 7, and 1 of the plant as sensors
- Input 3 of the plant as an additional known input

You can then connect the controller and estimator and form the dynamic compensator using this code:

```
controls = [1,2,4];
sensors = [4,7,1];
known = [3];
regulator = reg(sys_pp,K,L,sensors,known,controls)
```

Pole Placement Tools

You can use functions to

- Compute gain matrices K and L that achieve the desired closed-loop pole locations.
- Form the state estimator and dynamic compensator using these gains.

The following table summarizes the functions for pole placement.

Functions	Description
estim	Form state estimator given estimator gain
place	Pole placement design
reg	Form output-feedback compensator given state-feedback and estimator gains

Caution

Pole placement can be badly conditioned if you choose unrealistic pole locations. In particular, you should avoid:

- Placing multiple poles at the same location.

- Moving poles that are weakly controllable or observable. This typically requires high gain, which in turn makes the entire closed-loop eigenstructure very sensitive to perturbation.

Linear-Quadratic-Gaussian (LQG) Design

Linear-quadratic-Gaussian (LQG) control is a modern state-space technique for designing optimal dynamic regulators and servo controllers with integral action (also known as *set point trackers*). This technique allows you to trade off regulation/tracker performance and control effort, and to take into account process disturbances and measurement noise.

To design LQG regulators and set point trackers, you perform the following steps:

1 Construct the LQ-optimal gain.

2 Construct a Kalman filter (state estimator).

3 Form the LQG design by connecting the LQ-optimal gain and the Kalman filter.

For more information about using LQG design to create LQG regulators , see "Linear-Quadratic-Gaussian (LQG) Design for Regulation" on page 4-195.

For more information about using LQG design to create LQG servo controllers, see "Linear-Quadratic-Gaussian (LQG) Design of Servo Controller with Integral Action" on page 4-200.

These topics focus on the continuous-time case. For information about discrete-time LQG design, see the `dlqr` and `kalman` reference pages.

Linear-Quadratic-Gaussian (LQG) Design for Regulation

You can design an LQG regulator to regulate the output *y* around zero in the following model.
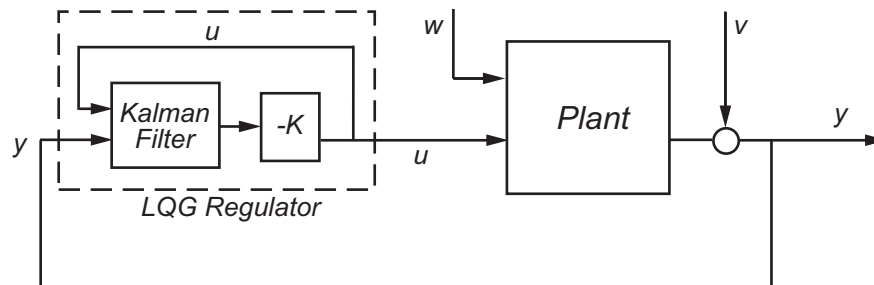
The plant in this model experiences disturbances (process noise) w and is driven by controls u. The regulator relies on the noisy measurements y to generate these controls. The plant state and measurement equations take the form of

$$\dot{x} = Ax + Bu + Gw$$
$$y = Cx + Du + Hw + v$$

and both w and v are modeled as white noise.

Note: LQG design requires a state-space model of the plant. You can use `ss` to convert other model formats to state space.

To design LQG regulators, you can use the design techniques shown in the following table.

To design an LQG regulator using...	Use the following commands:
A quick, one-step design technique when the following is true: • G is an identity matrix and $H = 0$. • All known (deterministic) inputs are control inputs and all outputs are measured. • Integrator states are weighted independently of states of plants and control inputs. • The state estimator of the Kalman filter is based on $\hat{x}[n \mid n-1]$.	`lqg` For more information, see the `lqg` reference page.
A more flexible, three-step design technique that allows you to specify: • Arbitrary G and H. • Known (deterministic) inputs that are not controls and/or outputs that are not measured.	`lqr`, `kalman`, and `lqgreg` For more information, see • "Constructing the Optimal State-Feedback Gain for Regulation" on page 4-197

To design an LQG regulator using...	Use the following commands:
• A flexible weighting scheme for integrator states, plant states, and controls. • The state estimator of the Kalman filter based on either $\hat{x}[n\,\|\,n]$ or $\hat{x}[n\,\|\,n-1]$.	• "Constructing the Kalman State Estimator" on page 4-197 • "Forming the LQG Regulator" on page 4-199

Constructing the Optimal State-Feedback Gain for Regulation

You construct the LQ-optimal gain from the following elements:

- State-space system matrices
- Weighting matrices Q, R, and N, which define the tradeoff between regulation performance (how fast $x(t)$ goes to zero) and control effort.

To construct the optimal gain, type the following command:

K= lqr(A,B,Q,R,N)

This command computes the optimal gain matrix K, for which the state feedback law $u = -Kx$ minimizes the following quadratic cost function for continuous time:

$$J(u) = \int_0^\infty \{x^T Q x + 2x^T N u + u^T R u\}dt$$

The software computes the gain matrix K by solving an algebraic Riccati equation.

For information about constructing LQ-optimal gain, including the cost function that the software minimizes for discrete time, see the lqr reference page.

Constructing the Kalman State Estimator

You need a Kalman state estimator for LQG regulation and servo control because you cannot implement optimal LQ-optimal state feedback without full state measurement.

You construct the state estimate $\hat{x}$ such that $u = -K\hat{x}$ remains optimal for the output-feedback problem. You construct the Kalman state estimator gain from the following elements:

- State-space plant model `sys`
- Noise covariance data, `Qn`, `Rn`, and `Nn`

  The following figure shows the required dimensions for `Qn`, `Rn`, and `Nn`. If `Nn` is 0, you can omit it.

Required Dimensions for Qn, Rn, and Nn

Note: You construct the Kalman state estimator in the same way for both regulation and servo control.

To construct the Kalman state estimator, type the following command:

```
[kest,L,P] = kalman(sys,Qn,Rn,Nn);
```

This command computes a Kalman state estimator, `kest` with the following plant equations:

$$\dot{x} = Ax + Bu + Gw$$
$$y = Cx + Du + Hw + v$$

where w and v are modeled as white noise. `L` is the Kalman gain and `P` the covariance matrix.

The software generates this state estimate using the Kalman filter

$$\frac{d}{dt}\hat{x} = A\hat{x} + Bu + L(y - C\hat{x} - Du)$$

with inputs u (controls) and y (measurements). The noise covariance data

$$E(ww^T) = Q_n, \quad E(vv^T) = R_n, \quad E(wv^T) = N_n$$

determines the Kalman gain L through an algebraic Riccati equation.

The Kalman filter is an optimal estimator when dealing with Gaussian white noise. Specifically, it minimizes the asymptotic covariance

$$\lim_{t \to \infty} E\left((x - \hat{x})(x - \hat{x})^T \right)$$

of the estimation error $x - \hat{x}$.

For more information, see the `kalman` reference page. For a complete example of a Kalman filter implementation, see "Kalman Filtering".

Forming the LQG Regulator

To form the LQG regulator, connect the Kalman filter `kest` and LQ-optimal gain `K` by typing the following command:

```
regulator = lqgreg(kest, K);
```
This command forms the LQG regulator shown in the following figure.
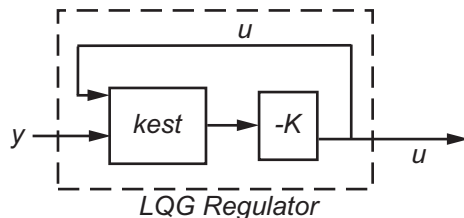
LQG Regulator

The regulator has the following state-space equations:

$$\frac{d}{dt}\hat{x} = [A - LC - (B - LD)K]\hat{x} + Ly$$
$$u = -K\hat{x}$$

For more information on forming LQG regulators, see the `lqgreg` reference page and "LQG Regulation: Rolling Mill Example".

Linear-Quadratic-Gaussian (LQG) Design of Servo Controller with Integral Action

You can design a servo controller with integral action for the following model:

LQG Servo Controller
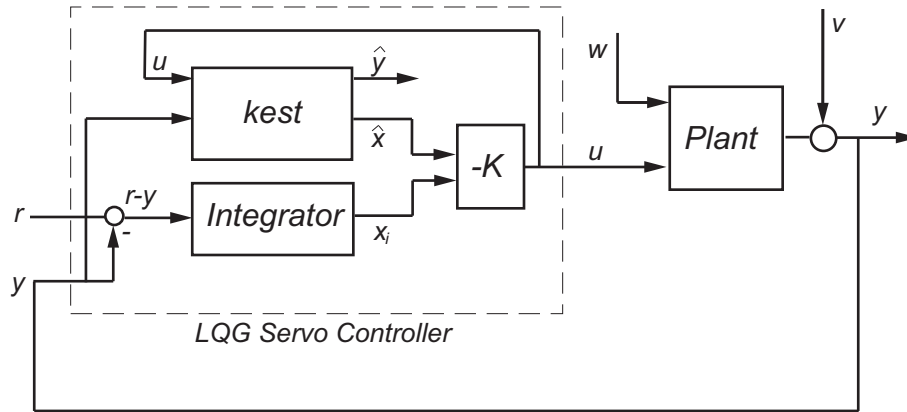
The servo controller you design ensures that the output y tracks the reference command r while rejecting process disturbances w and measurement noise v.

The plant in the previous figure is subject to disturbances w and is driven by controls u. The servo controller relies on the noisy measurements y to generate these controls. The plant state and measurement equations are of the form

$$\dot{x} = Ax + Bu + Gw$$
$$y = Cx + Du + Hw + v$$

and both w and v are modeled as white noise.

Note: LQG design requires a state-space model of the plant. You can use `ss` to convert other model formats to state space.

To design LQG servo controllers, you can use the design techniques shown in the following table.

To design an LQG servo controller using...	Use the following commands:
A quick, one-step design technique when the following is true: • G is an identity matrix and $H = 0$. • All known (deterministic) inputs are control inputs and all outputs are measured. • Integrator states are weighted independently of states of plants and control inputs. • The state estimator of the Kalman filter is based on $\hat{x}[n \mid n-1]$.	`lqg` For more information, see the `lqg` reference page.
A more flexible, three-step design technique that allows you to specify: • Arbitrary G and H. • Known (deterministic) inputs that are not controls and/or outputs that are not measured. • A flexible weighting scheme for integrator states, plant states, and controls. • The state estimator of the Kalman filter based on either $\hat{x}[n \mid n]$ or $\hat{x}[n \mid n-1]$.	`lqi`, `kalman`, and `lqgtrack` For more information, see • "Constructing the Optimal State-Feedback Gain for Servo Control" on page 4-201 • "Constructing the Kalman State Estimator" on page 4-202 • "Forming the LQG Servo Control" on page 4-203

Constructing the Optimal State-Feedback Gain for Servo Control

You construct the LQ-optimal gain from the

- State-space plant model `sys`
- Weighting matrices `Q`, `R`, and `N`, which define the tradeoff between tracker performance and control effort

To construct the optimal gain, type the following command:

```
K= lqi(sys,Q,R,N)
```

This command computes the optimal gain matrix K, for which the state feedback law $u = -Kz = -K[x; x_i]$ minimizes the following quadratic cost function for continuous time:

$$J(u) = \int_0^\infty \left\{ z^T Q z + u^T R u + 2 z^T N u \right\} dt$$

The software computes the gain matrix K by solving an algebraic Riccati equation.

For information about constructing LQ-optimal gain, including the cost function that the software minimizes for discrete time, see the lqi reference page.

Constructing the Kalman State Estimator

You need a Kalman state estimator for LQG regulation and servo control because you cannot implement LQ-optimal state feedback without full state measurement.

You construct the state estimate $\hat{x}$ such that $u = -K\hat{x}$ remains optimal for the output-feedback problem. You construct the Kalman state estimator gain from the following elements:

- State-space plant model sys
- Noise covariance data, Qn, Rn, and Nn

  The following figure shows the required dimensions for Qn, Rn, and Nn. If Nn is 0, you can omit it.

Required Dimensions for Qn, Rn, and Nn

Note: You construct the Kalman state estimator in the same way for both regulation and servo control.

To construct the Kalman state estimator, type the following command:

```
[kest,L,P] = kalman(sys,Qn,Rn,Nn);
```

This command computes a Kalman state estimator, `kest` with the following plant equations:

$$\dot{x} = Ax + Bu + Gw$$
$$y = Cx + Du + Hw + v$$

where w and v are modeled as white noise. `L` is the Kalman gain and `P` the covariance matrix.

The software generates this state estimate using the Kalman filter

$$\frac{d}{dt}\hat{x} = A\hat{x} + Bu + L(y - C\hat{x} - Du)$$

with inputs u (controls) and y (measurements). The noise covariance data

$$E(ww^T) = Q_n, \quad E(vv^T) = R_n, \quad E(wv^T) = N_n$$

determines the Kalman gain L through an algebraic Riccati equation.

The Kalman filter is an optimal estimator when dealing with Gaussian white noise. Specifically, it minimizes the asymptotic covariance

$$\lim_{t \to \infty} E\left((x - \hat{x})(x - \hat{x})^T\right)$$

of the estimation error $x - \hat{x}$.

For more information, see the `kalman` reference page. For a complete example of a Kalman filter implementation, see "Kalman Filtering".

Forming the LQG Servo Control

To form a two-degree-of-freedom LQG servo controller, connect the Kalman filter `kest` and LQ-optimal gain `K` by typing the following command:

```
servocontroller = lqgtrack(kest, K);
```

This command forms the LQG servo controller shown in the following figure.

LQG Servo Controller
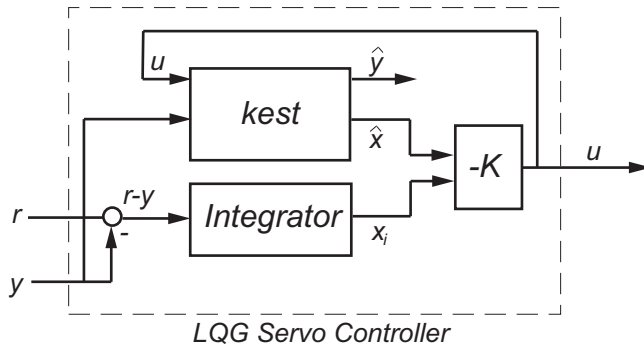
The servo controller has the following state-space equations:

$$\begin{bmatrix} \dot{\hat{x}} \\ \dot{x}_i \end{bmatrix} = \begin{bmatrix} A - BK_x - LC + LDK_x & -BK_i + LDK_i \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \hat{x} \\ x_i \end{bmatrix} + \begin{bmatrix} 0 & L \\ I & -I \end{bmatrix} \begin{bmatrix} r \\ y \end{bmatrix}$$

$$u = \begin{bmatrix} -K_x & -K_i \end{bmatrix} \begin{bmatrix} \hat{x} \\ x_i \end{bmatrix}$$

For more information on forming LQG servo controllers, including how to form a one-degree-of-freedom LQG servo controller, see the `lqgtrack` reference page.

Design an LQG Regulator

As an example of LQG design, consider the following regulation problem.

LQG Regulator
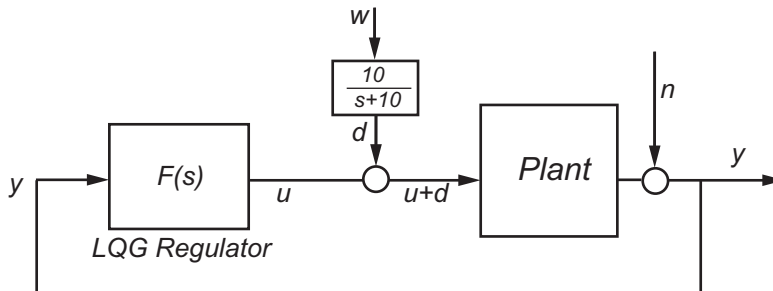
The goal is to regulate the plant output *y* around zero. The input disturbance *d* is low frequency with power spectral density (PSD) concentrated below 10 rad/s. For LQG design purposes, it is modeled as white noise driving a lowpass filter with a cutoff at 10 rad/s, shown in the following figure.
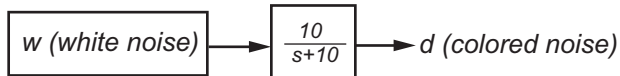
For simplicity, this noise is modeled as Gaussian white noise with variance of 1.

The following figure shows the Bode magnitude of the shaping filter.
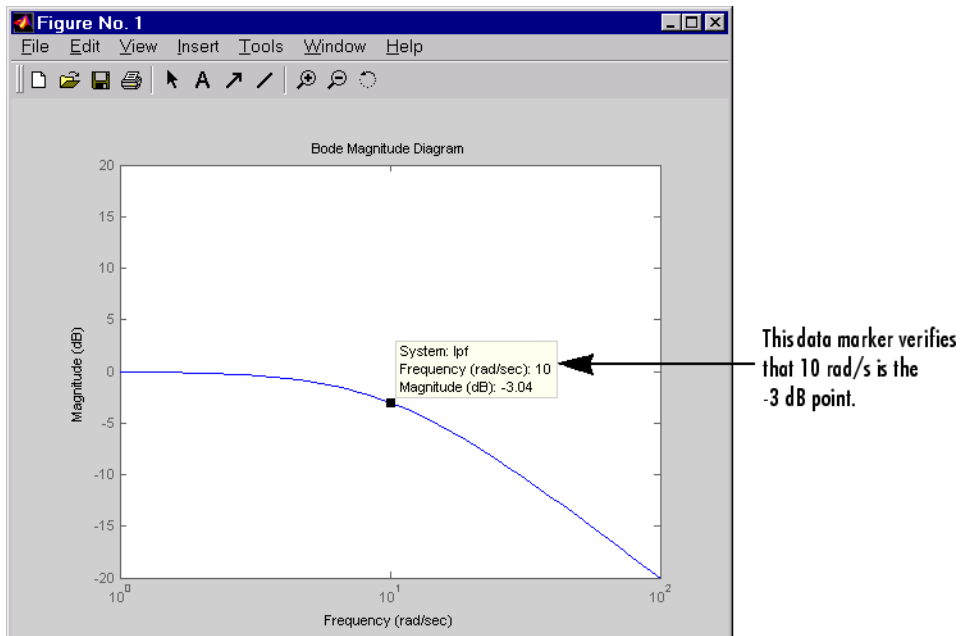
Bode Magnitude of the Lowpass Filter

There is some measurement noise *n*, with noise intensity given by

$$E(n^2) = 0.01$$

Use the cost function

$$J(u) = \int_0^\infty (10y^2 + u^2)dt$$

to specify the tradeoff between regulation performance and cost of control. The following equations represent an open-loop state-space model:

$$\dot{x} = Ax + Bu + Bd \quad (state\,equations)$$
$$y = Cx + n \qquad\quad (measurements)$$

where (A,B,C) is a state-space realization of $100/(s^2 + s + 100)$.

The following commands design the optimal LQG regulator $F(s)$ for this problem:

```
sys = ss(tf(100,[1 1 100])) % State-space plant model

% Design LQ-optimal gain K
K = lqry(sys,10,1)  % u = -Kx minimizes J(u)

% Separate control input u and disturbance input d
P = sys(:,[1 1]);
% input [u;d], output y

% Design Kalman state estimator Kest.
Kest = kalman(P,1,0.01)

% Form LQG regulator = LQ gain + Kalman filter.
F = lqgreg(Kest,K)
```

These commands returns a state-space model F of the LQG regulator $F(s)$. The `lqry`, `kalman`, and `lqgreg` functions perform discrete-time LQG design when you apply them to discrete plants.

To validate the design, close the loop with `feedback`, create and add the lowpass filter in series with the closed-loop system, and compare the open- and closed-loop impulse responses by using the `impulse` function.

```
% Close loop
```

```
clsys = feedback(sys,F,+1)
% Note positive feedback.

% Create the lowpass filter and add it in series with clsys.
s = tf('s');
lpf= 10/(s+10) ;
clsys_fin = lpf*clsys;

% Open- vs. closed-loop impulse responses
impulse(sys,'r--',clsys_fin,'b-')
```

These commands produce the following figure, which compares the open- and closed-loop impulse responses for this example.
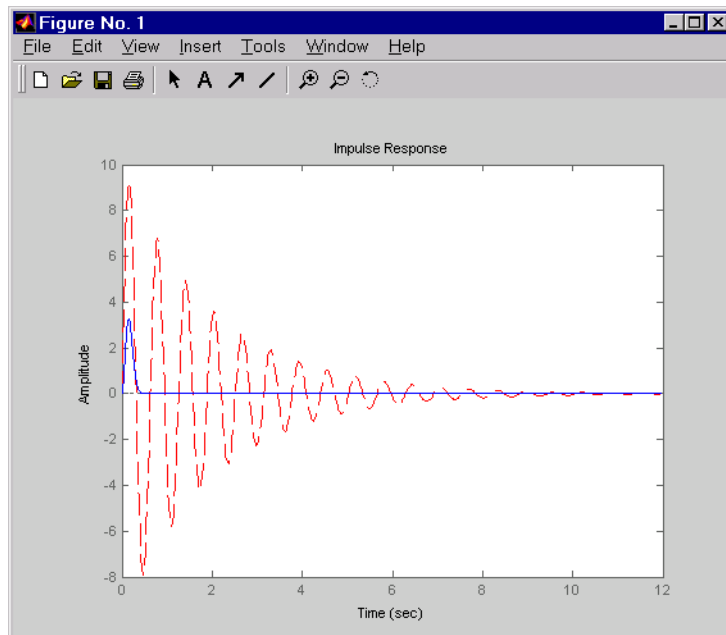
Comparison of Open- and Closed-Loop Impulse Response

Design an LQG Servo Controller

This example shows you how to design a servo controller for the following system.
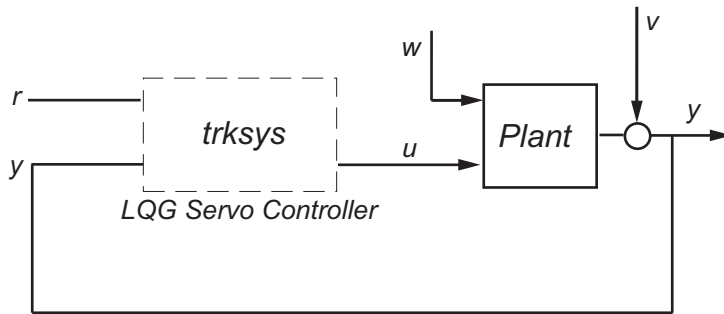
The plant has two states (x), two control inputs (u), two random inputs (w), one output (y), measurement noise for the output (v), and the following state and measurement equations:

$$\dot{x} = Ax + Bu + Gw$$
$$y = Cx + Du + Hw + v$$

where

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 0.3 & 1 \\ 0 & 1 \\ -0.3 & 0.9 \end{bmatrix} \quad G = \begin{bmatrix} -0.7 & 1.12 \\ -1.17 & 1 \\ 0.14 & 1.5 \end{bmatrix}$$

$$C = \begin{bmatrix} 1.9 & 1.3 & 1 \end{bmatrix} \quad D = \begin{bmatrix} 0.53 & -0.61 \end{bmatrix} \quad H = \begin{bmatrix} -1.2 & -0.89 \end{bmatrix}$$

The system has the following noise covariance data:

$$Q_n = E(ww^T) = \begin{bmatrix} 4 & 2 \\ 2 & 1 \end{bmatrix}$$
$$Rn = E(vv^T) = 0.7$$

Use the following cost function to define the tradeoff between tracker performance and control effort:

$$J(u) = \int_0^\infty \left(0.1 x^T x + x_i^2 + u^T \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} u \right) dt$$

To design an LQG servo controller for this system:

1 Create the state space system by typing the following in the MATLAB Command Window:

```
A = [0 1 0;0 0 1;1 0 0];
B = [0.3 1;0 1;-0.3 0.9];
G = [-0.7 1.12; -1.17 1; .14 1.5];
C = [1.9 1.3 1];
D = [0.53 -0.61];
H = [-1.2 -0.89];
sys = ss(A,[B G],C,[D H]);
```

2 Construct the optimal state-feedback gain using the given cost function by typing the following commands:

```
nx = 3;    %Number of states
ny = 1;    %Number of outputs
Q = blkdiag(0.1*eye(nx),eye(ny));
R = [1 0;0 2];
K = lqi(ss(A,B,C,D),Q,R);
```

3 Construct the Kalman state estimator using the given noise covariance data by typing the following commands:

```
Qn = [4 2;2 1];
Rn = 0.7;
kest = kalman(sys,Qn,Rn);
```

4 Connect the Kalman state estimator and the optimal state-feedback gain to form the LQG servo controller by typing the following command:

```
trksys = lqgtrack(kest,K)
```
This command returns the following LQG servo controller:

```
>> trksys = lqgtrack(kest,K)

a =
            x1_e     x2_e     x3_e      xi1
    x1_e  -2.373   -1.062   -1.649    0.772
    x2_e  -3.443   -2.876   -1.335   0.6351
    x3_e  -1.963   -2.483   -2.043   0.4049
```

```
    xi1         0        0        0        0

b =
             r1       y1
    x1_e      0   0.2849
    x2_e      0   0.7727
    x3_e      0   0.7058
    xi1       1       -1

c =
          x1_e      x2_e      x3_e       xi1
    u1  -0.5388   -0.4173   -0.2481    0.5578
    u2   -1.492    -1.388    -1.131    0.5869

d =
       r1   y1
    u1  0    0
    u2  0    0

Input groups:
        Name         Channels
      Setpoint          1
    Measurement         2

Output groups:
      Name        Channels
     Controls       1,2

Continuous-time model.
```

Design an LQR Servo Controller in Simulink

The following figure shows a Simulink block diagram shows a tracking problem in aircraft autopilot design. To open this diagram, type `lqrpilot` at the MATLAB prompt.
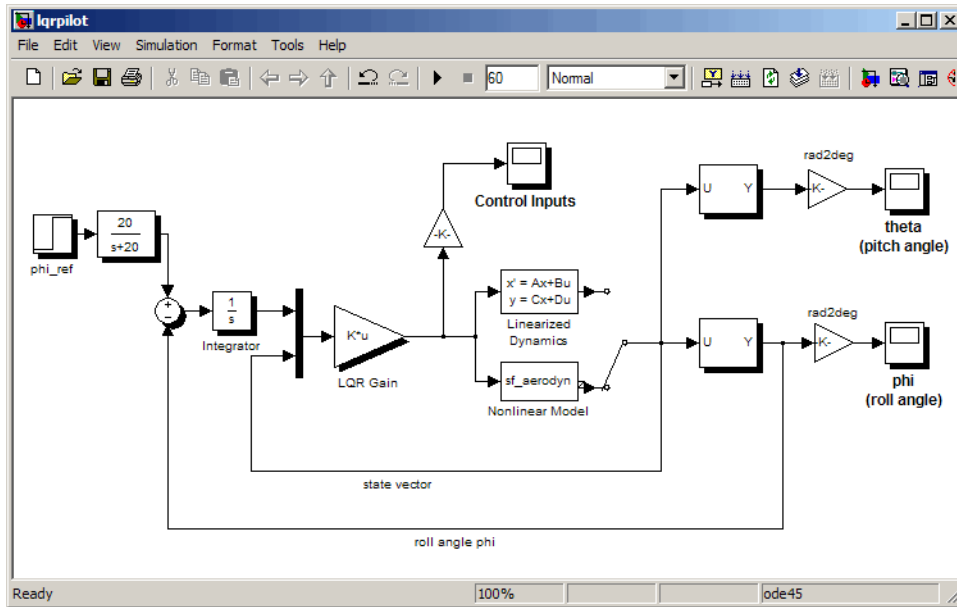
Key features of this diagram to note are the following:

- The Linearized Dynamics block contains the linearized airframe.
- `sf_aerodyn` is an S-Function block that contains the nonlinear equations for $(\theta,\phi) = (0, 15°)$.
- The error signal between ϕ and the ϕ_{ref} is passed through an integrator. This aids in driving the error to zero.

State-Space Equations for an Airframe

Beginning with the standard state-space equation

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

where

$$x = [u, v, w, p, q, r, \theta, \phi]^T$$

The variables u, v, and w are the three velocities with respect to the body frame, shown as follows.
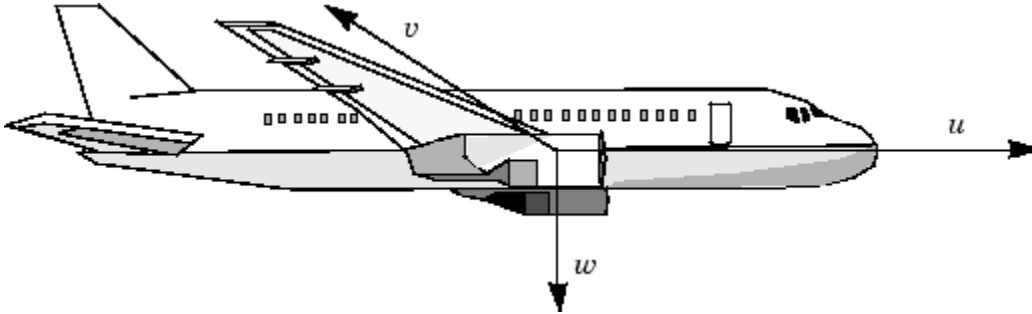
Body Coordinate Frame for an Aircraft

The variables ϕ and θ are roll and pitch, and p, q, and r are the roll, pitch, and yaw rates, respectively.

The airframe dynamics are nonlinear. The following equation shows the nonlinear components added to the state space equation.

$$\dot{x} = Ax + Bu + \begin{bmatrix} -g\sin\theta \\ g\cos\theta\sin\phi \\ g\cos\theta\cos\phi \\ 0 \\ 0 \\ 0 \\ q\cos\phi - r\sin\phi \\ (q\sin\phi + r\cos\phi)\cdot\tan\theta \end{bmatrix}$$

Nonlinear Component of the State-Space Equation

To see the numerical values for A and B, type

```
load lqrpilot
A, B
```

at the MATLAB prompt.

Trimming

For LQG design purposes, the nonlinear dynamics are trimmed at $\phi = 15°$ and p, q, r, and θ set to zero. Since u, v, and w do not enter into the nonlinear term in the preceding figure, this amounts to linearizing around $(\theta, \phi) = (0, 15°)$ with all remaining states set to zero. The resulting state matrix of the linearized model is called A15.

Problem Definition

The goal to perform a steady coordinated turn, as shown in this figure.
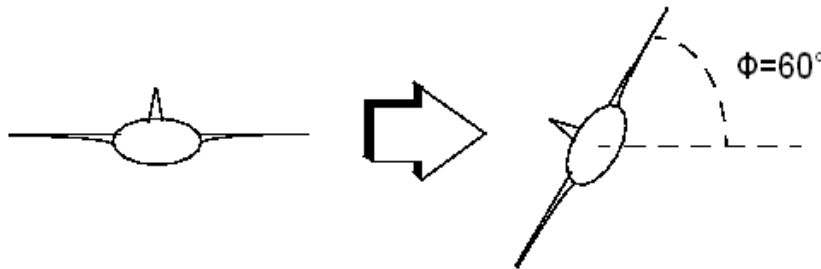
Aircraft Making a 60° Turn

To achieve this goal, you must design a controller that commands a steady turn by going through a 60° roll. In addition, assume that θ, the pitch angle, is required to stay as close to zero as possible.

Results

To calculate the LQG gain matrix, K, type

```
lqrdes
```

at the MATLAB prompt. Then, start the lqrpilot model with the nonlinear model, sf_aerodyn, selected.

This figure shows the response of ϕ to the 60° step command.
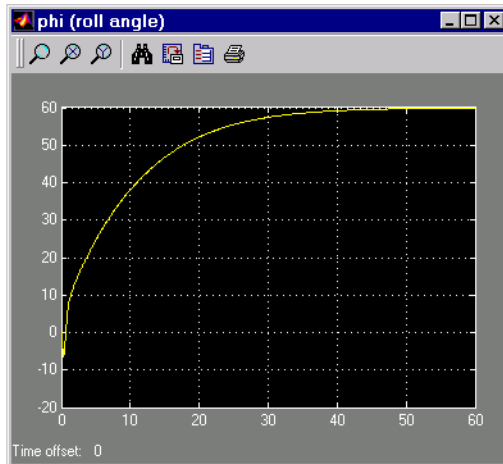
Tracking the Roll Step Command

As you can see, the system tracks the commanded 60° roll in about 60 seconds.

Another goal was to keep θ, the pitch angle, relatively small. This figure shows how well the LQG controller did.
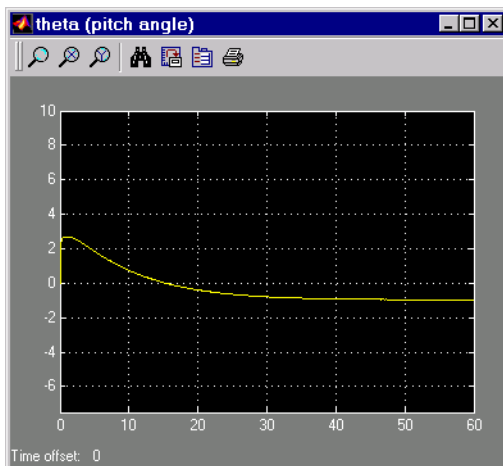
Minimizing the Displacement in the Pitch Angle, Theta

Finally, this figure shows the control inputs.
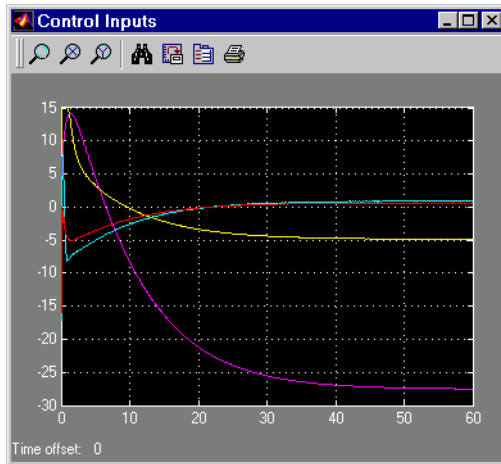
Control Inputs for the LQG Tracking Problem

Try adjusting the Q and R matrices in lqrdes.m and inspecting the control inputs and the system states, making sure to rerun lqrdes to update the LQG gain matrix K. Through trial and error, you may improve the response time of this design. Also, compare the linear and nonlinear designs to see the effects of the nonlinearities on the system performance.

State Estimation Using Time-Varying Kalman Filter

This example shows how to estimate states of linear systems using time-varying Kalman filters in Simulink. You use the Kalman Filter block from the Control System Toolbox library to estimate the position and velocity of a ground vehicle based on noisy position measurements such as GPS sensor measurements. The plant model in Kalman filter has time-varying noise characteristics.

Introduction

You want to estimate the position and velocity of a ground vehicle in the north and east directions. The vehicle can move freely in the two-dimensional space without any constraints. You design a multi-purpose navigation and tracking system that can be used for any object and not just a vehicle.
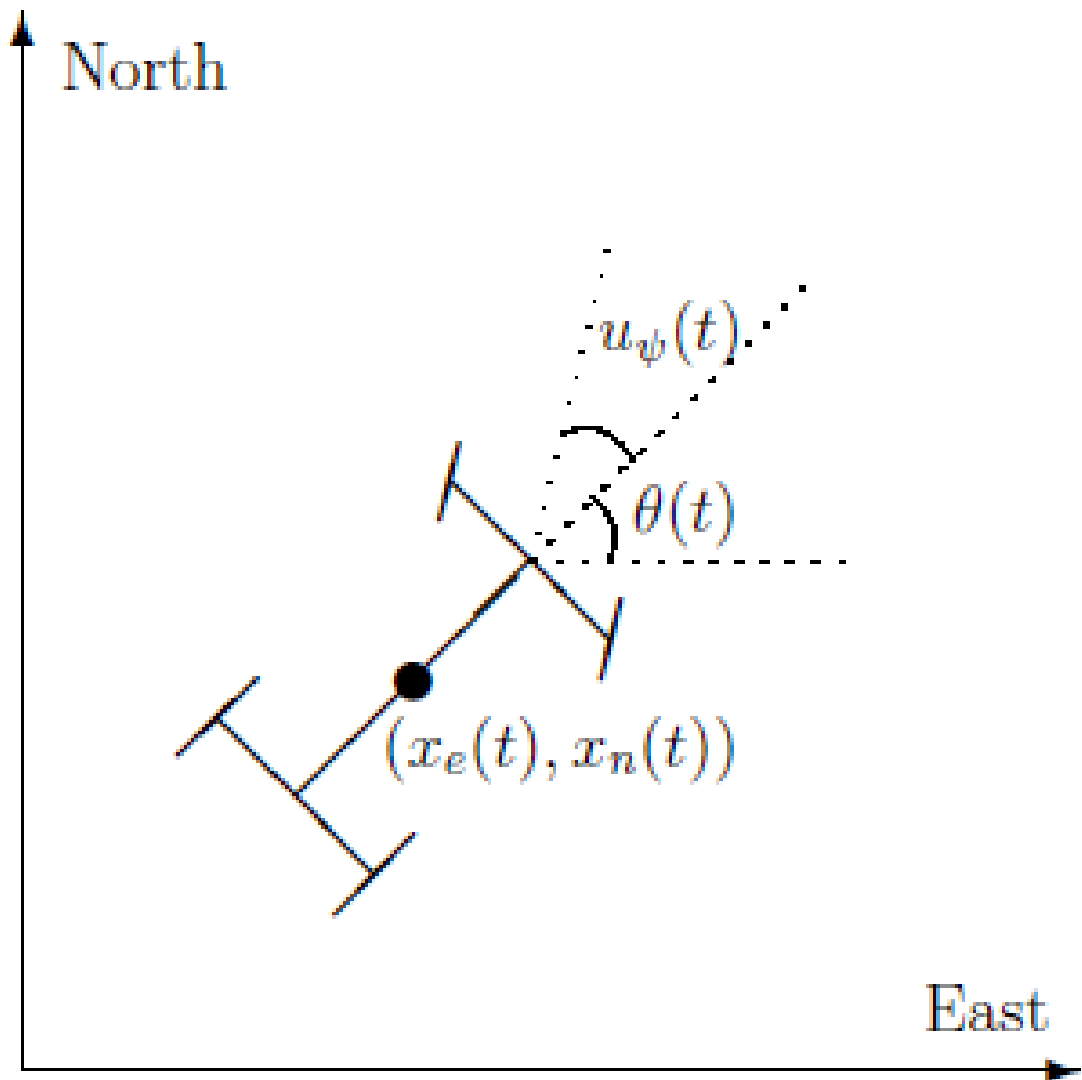
$x_e(t)$ and $x_n(t)$ are the vehicle's east and north positions from the origin, $\theta(t)$ is the vehicle orientation from east and $u_\psi(t)$ is the steering angle of the vehicle. t is the continuous-time variable.

The Simulink model consists of two main parts: Vehicle model and the Kalman filter. These are explained further in the following sections.

```
open_system('ctrlKalmanNavigationExample');
```
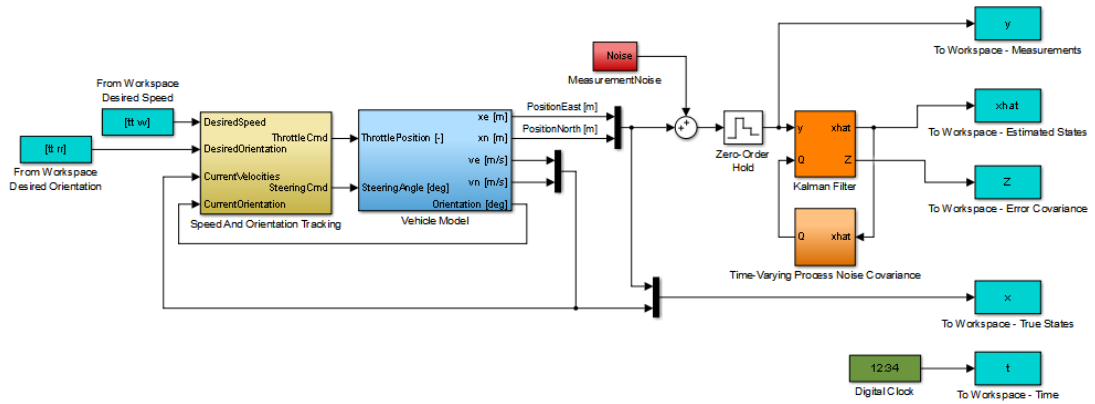

Copyright 2014 The MathWorks, Inc

Vehicle Model

The tracked vehicle is represented with a simple point-mass model:

$$
\frac{d}{dt}
\begin{bmatrix}
x_e(t) \\
x_n(t) \\
s(t) \\
\theta(t)
\end{bmatrix}
=
\begin{bmatrix}
s(t)\cos(\theta(t)) \\
s(t)\sin(\theta(t)) \\
(P\,\frac{u_T(t)}{s(t)} - A\,C_d\,s(t)^2)/m \\
s(t)\tan(u_\psi(t))/L
\end{bmatrix}
$$

where the vehicle states are:

$x_e(t)$ East position $[m]$
$x_n(t)$ North position $[m]$
$s(t)$ Speed $[m/s]$
$\theta(t)$ Orientation from east $[deg]$

the vehicle parameters are:

$P = 100000$ Peak engine power $[W]$
$A = 1$ Frontal area $[m^2]$
$C_d = 0.3$ Drag coefficient $[Unitless]$
$m = 1250$ Vehicle mass $[kg]$
$L = 2.5$ Wheelbase length $[m]$

and the control inputs are:

$u_T(t)$ Throttle position in the range of -1 and 1 $[Unitless]$
$u_\psi(t)$ Steering angle $[deg]$

The longitunidal dynamics of the model ignore tire rolling resistance. The lateral dynamics of the model assume that the desired steering angle can be achieved instantaneously and ignore the yaw moment of inertia.

The car model is implemented in the `ctrlKalmanNavigationExample/Vehicle Model` subsystem. The Simulink model contains two PI controllers for tracking the desired orientation and speed for the car in the `ctrlKalmanNavigationExample/ Speed And Orientation Tracking` subsystem. This allows you to specify various operating conditions for the car test the Kalman filter performance.

Kalman Filter Design

Kalman filter is an algorithm to estimate unknown variables of interest based on a linear model. This linear model describes the evolution of the estimated variables over time in response to model initial conditions as well as known and unknown model inputs. In this example, you estimate the following parameters/variables:

$$\hat{x}[n] = \begin{bmatrix} \hat{x}_e[n] \\ \hat{x}_n[n] \\ \hat{\dot{x}}_e[n] \\ \hat{\dot{x}}_n[n] \end{bmatrix}$$

where

$\hat{x}_e[n]$ East position estimate $[m]$
$\hat{x}_n[n]$ North position estimate $[m]$
$\hat{\dot{x}}_e[n]$ East velocity estimate $[m/s]$
$\hat{\dot{x}}_n[n]$ North velocity estimate $[m/s]$

The $\dot{x}$ terms denote velocities and not the derivative operator. n is the discrete-time index. The model used in the Kalman filter is of the form:

$$\begin{aligned} \hat{x}[n+1] &= A\hat{x}[n] + Gw[n] \\ y[n] &= C\hat{x}[n] + v[n] \end{aligned}$$

where $\hat{x}$ is the state vector, y is the measurements, w is the process noise, and v is the measurement noise. Kalman filter assumes that w and v are zero-mean, independent random variables with known variances $E[ww^T] = Q$, $E[vv^T] = R$, and $E[wv^T] = N$. Here, the A, G, and C matrices are:

$$A = \begin{bmatrix} 1 & 0 & T_s & 0 \\ 0 & 1 & 0 & T_s \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$G = \begin{bmatrix} T_s/2 & 0 \\ 0 & T_s/2 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

where $T_s = 1\,[s]$

The third row of A and G model the east velocity as a random walk:
$\hat{\dot{x}}_e[n+1] = \hat{\dot{x}}_e[n] + w_1[n]$. In reality, position is a continuous-time variable and is the integral of velocity over time $\frac{d}{dt}\hat{x}_e = \hat{\dot{x}}_e$. The first row of the A and G represent a disrete approximation to this kinematic relationship:
$(\hat{x}_e[n+1] - \hat{x}_e[n])/Ts = (\hat{\dot{x}}_e[n+1] + \hat{\dot{x}}_e[n])/2$. The second and fourth rows of the A and G represent the same relationship between the north velocity and position.

The C matrix represents that only position measurements are available. A position sensor, such as GPS, provides these measurements at the sample rate of 1Hz. The

variance of the measurment noise v, the R matrix, is specified as $R = 50$. Since R is specified as a scalar, the Kalman filter block assumes that the matrix R is diagonal, its diagonals are 50 and is of compatible dimensions with y. If the measurement noise is Gaussian, R=50 corresponds to 68% of the position measurements being within $\pm\sqrt{50}\,m$ or the actual position in the east and north directions. However, this assumption is not necessary for the Kalman filter.

The elements of w capture how much the vehicle velocity can change over one sample time Ts. The variance of the process noise w, the Q matrix, is chosen to be time-varying. It captures the intuition that typical values of $w[n]$ are smaller when velocity is large. For instance, going from 0 to 10m/s is easier than going from 10 to 20m/s. Concretely, you use the estimated north and east velocities and a saturation function to construct Q[n]:

$$f_{sat}(z) = min(max(z, 25), 625)$$

$$Q[n] = \begin{bmatrix} 1 + \dfrac{250}{f_{sat}(\hat{x}_e^2)} & 0 \\ 0 & 1 + \dfrac{250}{f_{sat}(\hat{x}_n^2)} \end{bmatrix}$$

The diagonals of Q model the variance of w inversely proportional to the square of the estimated velocities. The saturation function prevents Q from becoming too large or small. The coefficient 250 is obtained from a least squares fit to 0-5, 5-10, 10-15, 15-20, 20-25m/s acceleration time data for a generic vehicle. Note that the diagonal Q implies a naive approach that assumes that the velocity changes in north and east direction are uncorrelated.

Kalman Filter Block Inputs and Setup

The 'Kalman Filter' block is in the `Control System Toolbox` library in Simulink. It is also in `System Identification Toolbox/Estimators` library. Configure the block parameters for discrete-time state estimation. Specify the following **Filter Settings** parameters:

- **Time domain:** Discrete-time. Choose this option to estimate discrete-time states.

- Select the **Use current measurement y[n] to improve the xhat[n]** check box. This implements the "current estimator" variant of the discrete-time Kalman filter.

This option improves the estimation accuracy and is more useful for slow sample times. However, it increases the computational cost. In addition, this Kalman filter variant has direct feedthrough, which leads to an algebraic loop if the Kalman filter is used in a feedback loop that does not contain any delays (the feedback loop itself also has direct feedthrough). The algebraic loop can further impact the simulation speed.

Click the **Options** tab to set the block inport and outport options:

- Unselect the **Add input port u** check box. There are no known inputs in the plant model.

- Select the **Output state estimation error covariance Z** check box. The Z matrix provides information about the filter's confidence in the state estimates.

Kalman Filter

Estimate the states of a discrete-time or continuous-time linear system. Time-varying systems are supported.

Filter Settings

Time domain: Discrete-Time ▼

☑ Use the current measurement y[n] to improve xhat[n]

| Model Parameters | Options |

Additional Inports

☐ Add input port u

☐ Add input port Enable to control measurement updates

External reset:

None ▼

Additional Outports

☐ Output estimated model output y

☑ Output state estimation error covariance Z

Sample time (-1 for inherited): Ts

| OK | Cancel | Help | Apply |

Click **Model Parameters** to specify the plant model and noise characteristics:

- **Model source:** Individual A, B, C, D matrices.

- **A**: A. The A matrix is defined earlier in this example.

- **C**: C. The C matrix is defined earlier in this example.

- **Initial Estimate Source**: Dialog

- **Initial states x[0]**: 0. This represents an initial guess of 0 for the position and velocity estimates at t=0s.

- **State estimation error covariance P[0]**: 10. Assume that the error between your initial guess x[0] and its actual value is a random variable with a standard deviation $\sqrt{10}$.

- Select the **Use G and H matrices (defalut G=I and H=0)** check box to specify a non-default G matrix.

- **G**: G. The G matrix is defined earlier in this example.

- **H**: 0. The process noise does not impact the measurments y entering the Kalman filter block.

- Unselect the **Time-invariant Q** check box. The Q matrix is time-varying and is supplied through the block inport Q. The block uses a time-varying Kalman filter due to this setting. You can select this option to use a time-invariant Kalman filter. Time-invariant Kalman filter performs slightly worse for this problem, but is easier to design and has a lower computational cost.

- **R**: R. This is the covariance of the measurement noise $v[n]$. The R matrix is defined earlier in this example.

- **N**: 0. Assume that there is no correlation between process and measurement noises.

- **Sample time (-1 for inherited)**: Ts, which is defined earlier in this example.

Kalman Filter

Estimate the states of a discrete-time or continuous-time linear system. Time-varying systems are supported.

Filter Settings

Time domain: Discrete-Time ▼

☑ Use the current measurement y[n] to improve xhat[n]

Model Parameters   Options

System Model

Model source: Individual A, B, C, D matrices ▼

A: [1 0 Ts 0; 0 1 0 Ts; 0 0 1 0; 0 0 0 1]

C: [1 0 0 0; 0 1 0 0]

Initial Estimates

Source: Dialog ▼

Initial states x[0]: 0

State estimation error covariance P[0] 10

Noise Characteristics

☑ Use G and H matrices (default G=I and H=0)

G: [Ts/2 0; 0 Ts/2; 1 0; 0 1]   ☑ Time-invariant G

H: 0   ☑ Time-invariant H

Results

Test the performance of the Kalman filter by simulating a scenario where the vehicle makes the following maneuvers:

- At t = 0 the vehicle is at $x_e(0) = 0$, $x_n(0) = 0$ and is stationary.

- Heading east, it accelerates to 25m/s. It decelerates to 5m/s at t=50s.

- At t = 100s, it turns toward north and accelerates to 20m/s.

- At t = 200s, it makes another turn toward west. It accelerates to 25m/s.

- At t = 260s, it decelerates to 15m/s and makes a constant speed 180 degree turn.

Simulate the Simulink model. Plot the actual, measured and Kalman filter estimates of vehicle position.

```
sim('ctrlKalmanNavigationExample');

figure(1);
% Plot results and connect data points with a solid line.
plot(x(:,1),x(:,2),'bx',...
    y(:,1),y(:,2),'gd',...
    xhat(:,1),xhat(:,2),'ro',...
    'LineStyle','-');
title('Position');
xlabel('East [m]');
ylabel('North [m]');
legend('Actual','Measured','Kalman filter estimate','Location','Best');
axis tight;
```
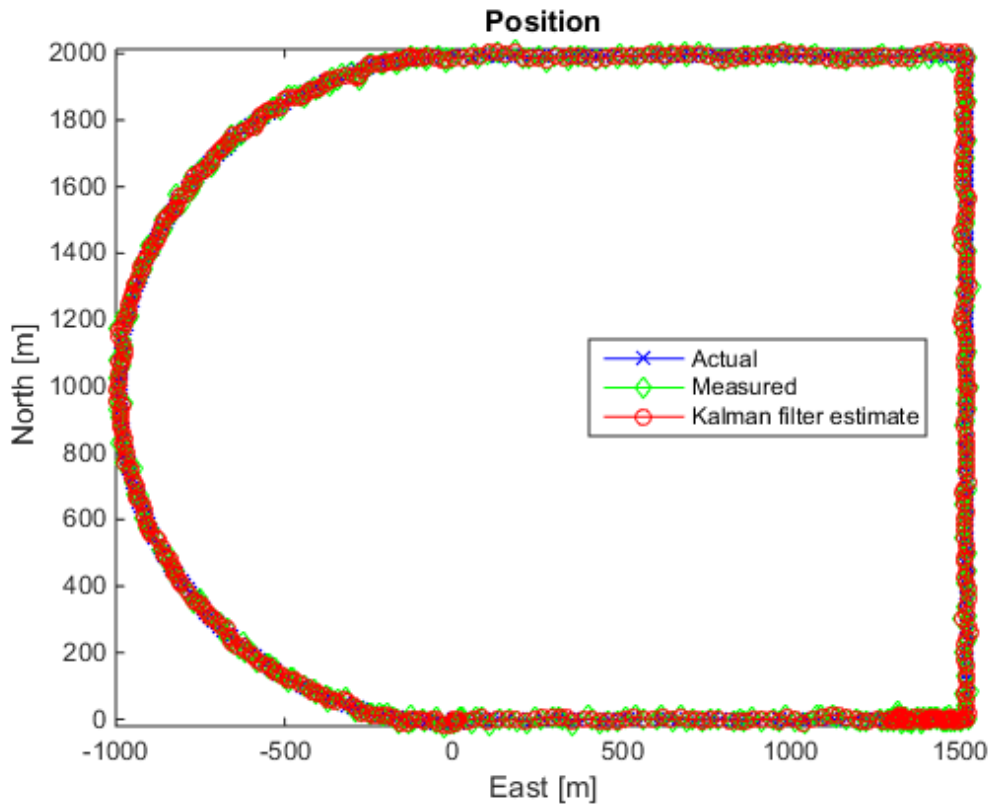
The error between the measured and actual position as well as the error between the kalman filter estimate and actual position is:

```
% East position measurement error [m]
n_xe = y(:,1)-x(:,1);
% North position measurement error [m]
n_xn = y(:,2)-x(:,2);
% Kalman filter east position error [m]
e_xe = xhat(:,1)-x(:,1);
% Kalman filter north position error [m]
e_xn = xhat(:,2)-x(:,2);

figure(2);
% East Position Errors
```

```
subplot(2,1,1);
plot(t,n_xe,'g',t,e_xe,'r');
ylabel('Position Error - East [m]');
xlabel('Time [s]');
legend(sprintf('Meas: %.3f',norm(n_xe,1)/numel(n_xe)),sprintf('Kalman f.: %.3f',norm(e_
axis tight;
% North Position Errors
subplot(2,1,2);
plot(t,y(:,2)-x(:,2),'g',t,xhat(:,2)-x(:,2),'r');
ylabel('Position Error - North [m]');
xlabel('Time [s]');
legend(sprintf('Meas: %.3f',norm(n_xn,1)/numel(n_xn)),sprintf('Kalman f: %.3f',norm(e_x
axis tight;
```
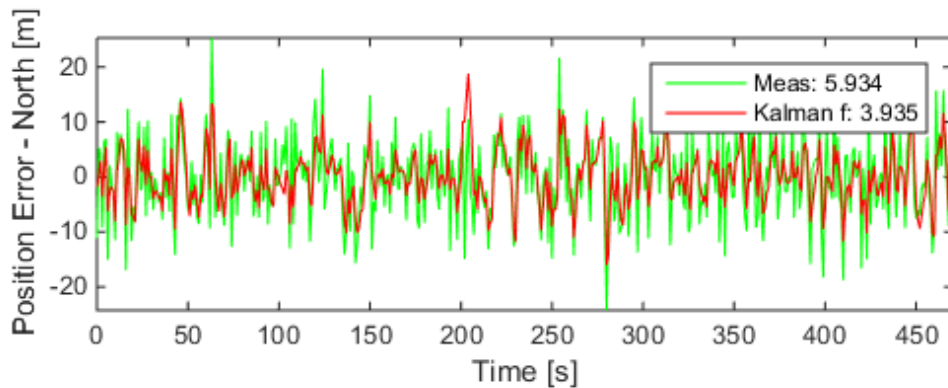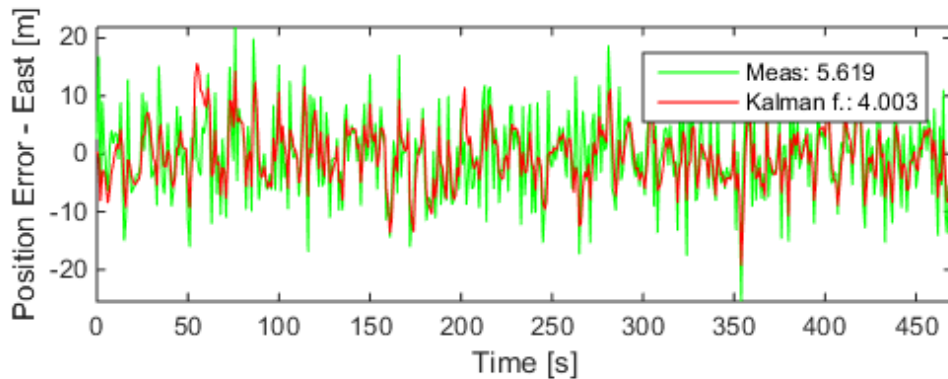
The plot legends show the position measurement and estimation error ($\|x_e - \hat{x}_e\|_1$ and $\|x_n - \hat{x}_n\|_1$) normalized by the number of data points. The Kalman filter estimates have about 25% percent less error than the raw measurements.

The actual velocity in the east direction and its Kalman filter estimate is shown below in the top plot. The bottom plot shows the estimation error.

```
e_ve = xhat(:,3)-x(:,3); % [m/s] Kalman filter east velocity error
e_vn = xhat(:,4)-x(:,4); % [m/s] Kalman filter north velocity error
figure(3);
% Velocity in east direction and its estimate
subplot(2,1,1);
plot(t,x(:,3),'b',t,xhat(:,3),'r');
ylabel('Velocity - East [m]');
xlabel('Time [s]');
legend('Actual','Kalman filter','Location','Best');
axis tight;
subplot(2,1,2);
% Estimation error
plot(t,e_ve,'r');
ylabel('Velocity Error - East [m]');
xlabel('Time [s]');
legend(sprintf('Kalman filter: %.3f',norm(e_ve,1)/numel(e_ve)));
axis tight;
```
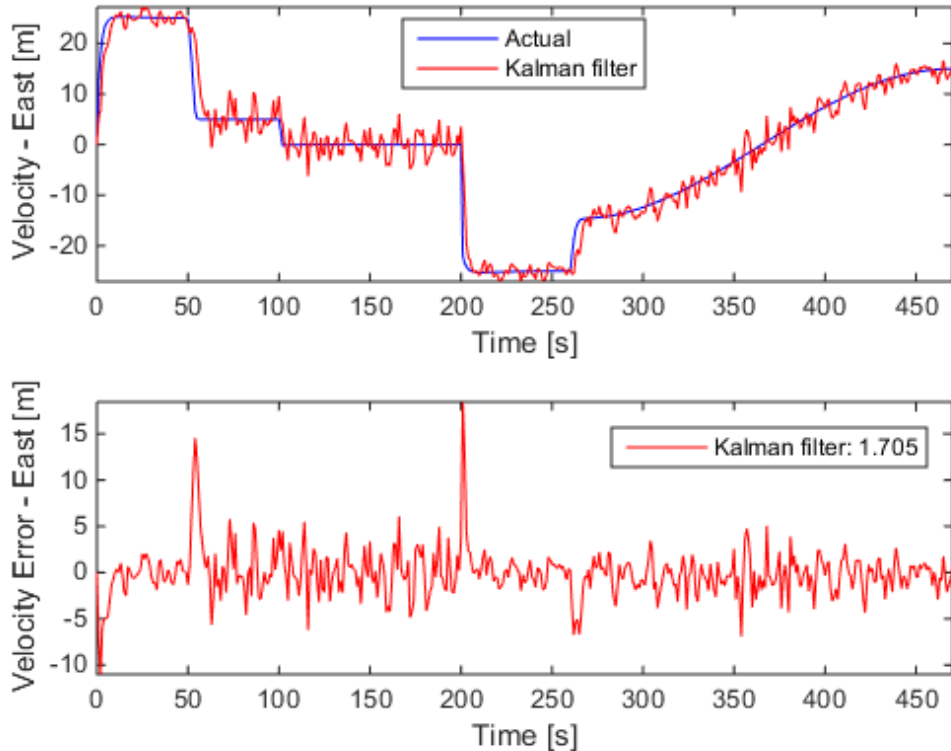
The legend on the error plot shows the east velocity estimation error $||\dot{x}_e - \hat{\dot{x}}_e||_1$ normalized by the number of data points.

The Kalman filter velocity estimates track the actual velocity trends correctly. The noise levels decrease when the vehicle is traveling at high velocities. This is in line with the design of the Q matrix. The large two spikes are at t=50s and t=200s. These are the times when the car goes through sudden decelearation and a sharp turn, respectively. The velocity changes at those instants are much larger than the predictions from the Kalman filter, which is based on its Q matrix input. After a few time-steps, the filter estimates catch up with the actual velocity.

Summary

You estimated the position and velocity of a vehicle using the Kalman filter block in Simulink. The process noise dynamics of the model were time-varying. You validated the filter performance by simulating various vehicle maneuvers and randomly generated measurement noise. The Kalman filter improved the position measurements and provided velocity estimates for the vehicle.

```
bdclose('ctrlKalmanNavigationExample');
```

Kalman Filter Design

This example shows how to perform Kalman filtering. Both a steady state filter and a time varying filter are designed and simulated below.

Problem Description

Given the following discrete plant

$$x(n + 1) = Ax(n) + Bu(n)$$

$$y(n) = Cx(n) + Du(n)$$

where

```
A = [1.1269    -0.4940     0.1129,
      1.0000          0         0,
           0     1.0000         0];

B = [-0.3832
      0.5919
      0.5191];

C = [1 0 0];

D = 0;
```

design a Kalman filter to estimate the output y based on the noisy measurements yv[n] = C x[n] + v[n]

Steady-State Kalman Filter Design

You can use the function KALMAN to design a steady-state Kalman filter. This function determines the optimal steady-state filter gain M based on the process noise covariance Q and the sensor noise covariance R.

First specify the plant + noise model. CAUTION: set the sample time to -1 to mark the plant as discrete.

```
Plant = ss(A,[B B],C,0,-1,'inputname',{'u' 'w'},'outputname','y');
```

Specify the process noise covariance (Q):

```
Q = 2.3; % A number greater than zero
```

Specify the sensor noise covariance (R):

```
R = 1; % A number greater than zero
```

Now design the steady-state Kalman filter with the equations

```
    Time update:         x[n+1|n] = Ax[n|n-1] + Bu[n]

    Measurement update:  x[n|n] = x[n|n-1] + M (yv[n] - Cx[n|n-1])

                         where M = optimal innovation gain
using the KALMAN command:
```

```
[kalmf,L,~,M,Z] = kalman(Plant,Q,R);
```

The first output of the Kalman filter KALMF is the plant output estimate y_e = Cx[n|n], and the remaining outputs are the state estimates. Keep only the first output y_e:
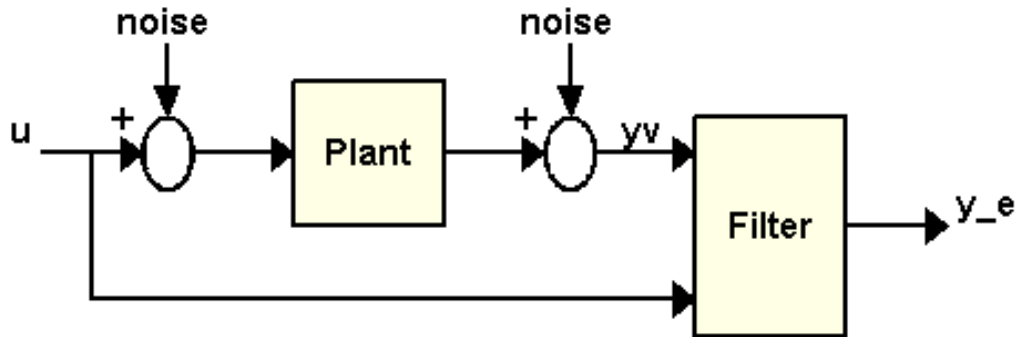
```
kalmf = kalmf(1,:);

M,    % innovation gain
```

```
M =

   0.5345
   0.0101
  -0.4776
```

To see how this filter works, generate some data and compare the filtered response with the true plant response:

To simulate the system above, you can generate the response of each part separately or generate both together. To simulate each separately, first use LSIM with the plant and then with the filter. The following example simulates both together.

```
% First, build a complete plant model with u,w,v as inputs and
% y and yv as outputs:
a = A;
b = [B B 0*B];
c = [C;C];
d = [0 0 0;0 0 1];
P = ss(a,b,c,d,-1,'inputname',{'u' 'w' 'v'},'outputname',{'y' 'yv'});
```

Next, connect the plant model and the Kalman filter in parallel by specifying u as a shared input:

```
sys = parallel(P,kalmf,1,1,[],[]);
```

Finally, connect the plant output yv to the filter input yv. Note: yv is the 4th input of SYS and also its 2nd output:

```
SimModel = feedback(sys,1,4,2,1);
SimModel = SimModel([1 3],[1 2 3]);      % Delete yv form I/O
```

The resulting simulation model has w,v,u as inputs and y,y_e as outputs:

```
SimModel.inputname

ans =

    'w'
    'v'
    'u'

SimModel.outputname

ans =

    'y'
    'y_e'
```

You are now ready to simulate the filter behavior. Generate a sinusoidal input vector (known):

```
t = (0:100)';
u = sin(t/5);
```

Generate process noise and sensor noise vectors:

```
rng(10,'twister');
w = sqrt(Q)*randn(length(t),1);
v = sqrt(R)*randn(length(t),1);
```
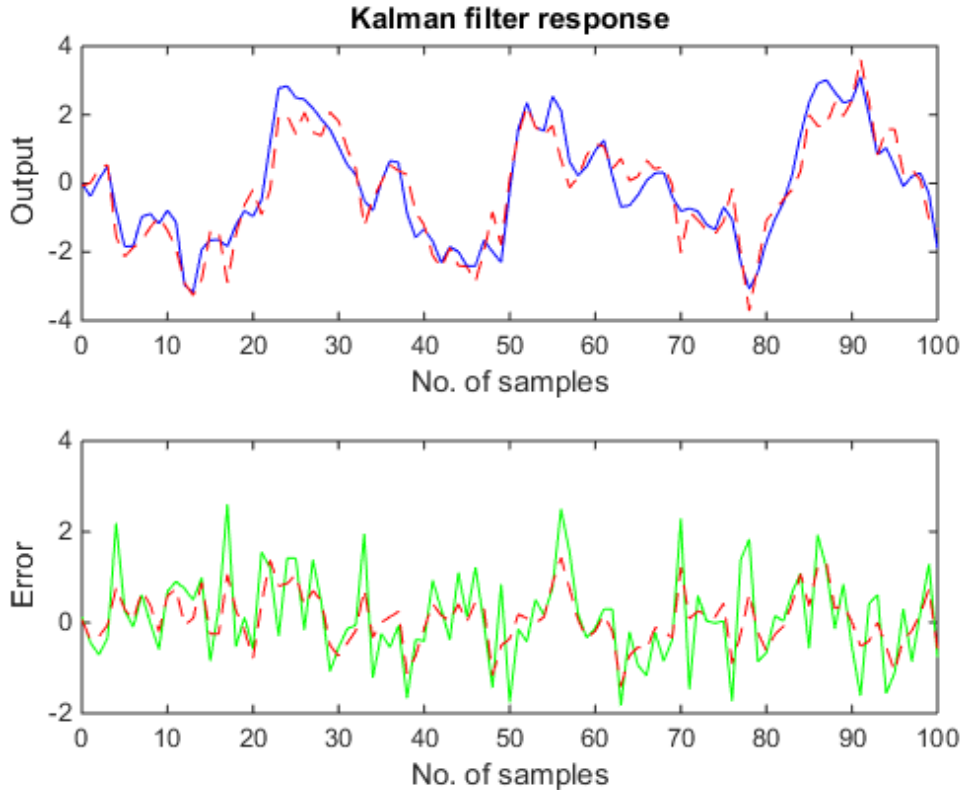
Now simulate the response using LSIM:

```
out = lsim(SimModel,[w,v,u]);

y = out(:,1);   % true response
ye = out(:,2);  % filtered response
yv = y + v;     % measured response
```

Compare the true response with the filtered response:

```
clf
subplot(211), plot(t,y,'b',t,ye,'r--'),
xlabel('No. of samples'), ylabel('Output')
title('Kalman filter response')
```

```
subplot(212), plot(t,y-yv,'g',t,y-ye,'r--'),
xlabel('No. of samples'), ylabel('Error')
```


As shown in the second plot, the Kalman filter reduces the error y-yv due to measurement noise. To confirm this, compare the error covariances:

```
MeasErr = y-yv;
MeasErrCov = sum(MeasErr.*MeasErr)/length(MeasErr);
EstErr = y-ye;
EstErrCov = sum(EstErr.*EstErr)/length(EstErr);
```

Covariance of error before filtering (measurement error):

```
MeasErrCov
```

```
MeasErrCov =

    0.9871
```

Covariance of error after filtering (estimation error):

```
EstErrCov
```

```
EstErrCov =

    0.3479
```

Time-Varying Kalman Filter Design

Now, design a time-varying Kalman filter to perform the same task. A time-varying Kalman filter can perform well even when the noise covariance is not stationary. However for this example, we will use stationary covariance.

The time varying Kalman filter has the following update equations.

```
Time update:           x[n+1|n] = Ax[n|n] + Bu[n]

                       P[n+1|n] = AP[n|n]A' + B*Q*B'

Measurement update:
                       x[n|n] = x[n|n-1] + M[n](yv[n] - Cx[n|n-1])
                                                                 -1
                       M[n] = P[n|n-1] C' (CP[n|n-1]C'+R)

                       P[n|n] = (I-M[n]C) P[n|n-1]
```

First, generate the noisy plant response:

```
sys = ss(A,B,C,D,-1);
y = lsim(sys,u+w);     % w = process noise
yv = y + v;            % v = meas. noise
```

Next, implement the filter recursions in a FOR loop:

```
P=B*Q*B';          % Initial error covariance
```

```matlab
x=zeros(3,1);      % Initial condition on the state
ye = zeros(length(t),1);
ycov = zeros(length(t),1);
errcov = zeros(length(t),1);

for i=1:length(t)
  % Measurement update
  Mn = P*C'/(C*P*C'+R);
  x = x + Mn*(yv(i)-C*x);  % x[n|n]
  P = (eye(3)-Mn*C)*P;     % P[n|n]

  ye(i) = C*x;
  errcov(i) = C*P*C';

  % Time update
  x = A*x + B*u(i);        % x[n+1|n]
  P = A*P*A' + B*Q*B';     % P[n+1|n]
end
```
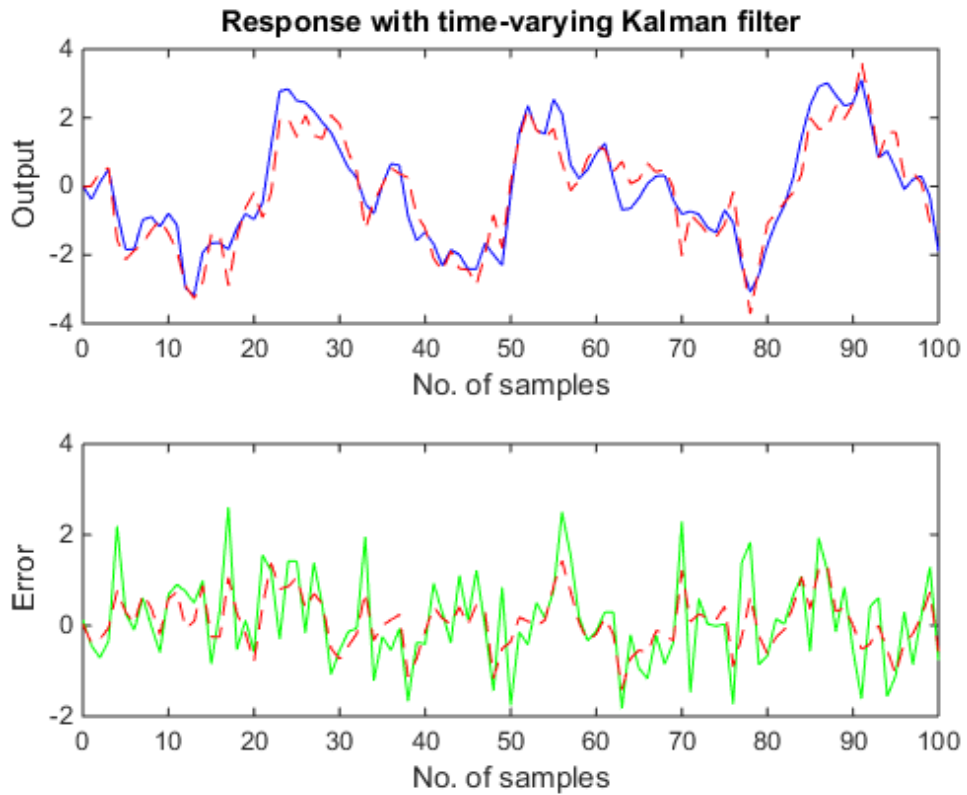
Now, compare the true response with the filtered response:

```matlab
subplot(211), plot(t,y,'b',t,ye,'r--'),
xlabel('No. of samples'), ylabel('Output')
title('Response with time-varying Kalman filter')
subplot(212), plot(t,y-yv,'g',t,y-ye,'r--'),
xlabel('No. of samples'), ylabel('Error')
```

The time varying filter also estimates the output covariance during the estimation. Plot the output covariance to see if the filter has reached steady state (as we would expect with stationary input noise):

```
subplot(211)
plot(t,errcov), ylabel('Error Covar'),
```
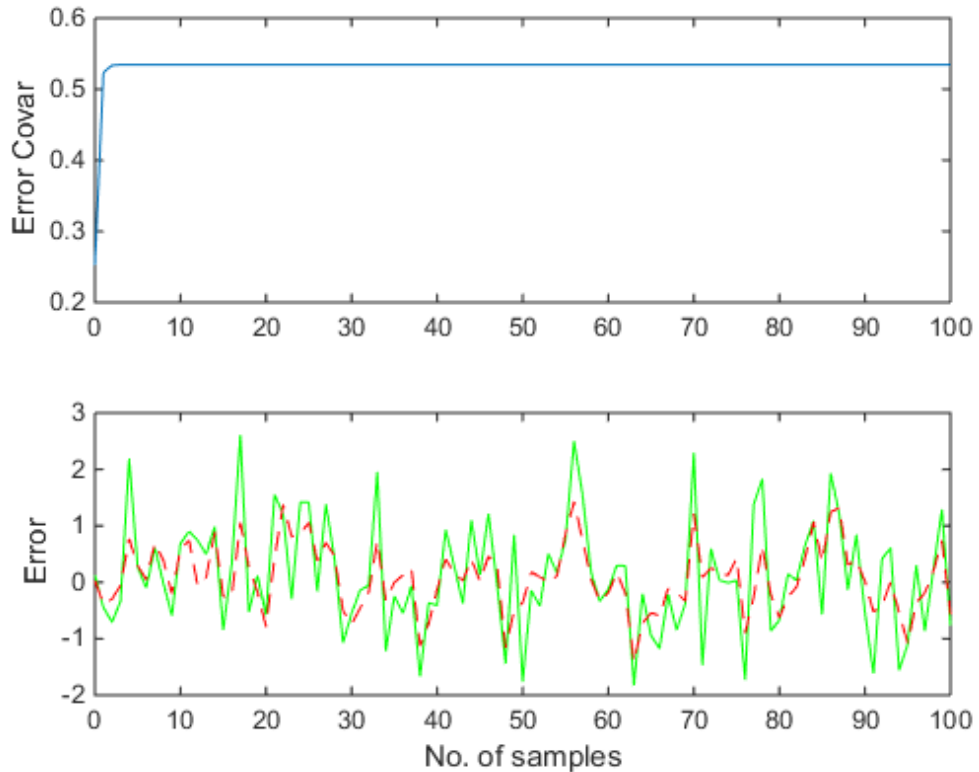
From the covariance plot you can see that the output covariance did reach a steady state in about 5 samples. From then on, the time varying filter has the same performance as the steady state version.

Compare covariance errors:

```
MeasErr = y-yv;
MeasErrCov = sum(MeasErr.*MeasErr)/length(MeasErr);
EstErr = y-ye;
EstErrCov = sum(EstErr.*EstErr)/length(EstErr);
```

Covariance of error before filtering (measurement error):

```
MeasErrCov
```

```
MeasErrCov =

    0.9871
```

Covariance of error after filtering (estimation error):

```
EstErrCov
```

```
EstErrCov =

    0.3479
```

Verify that the steady-state and final values of the Kalman gain matrices coincide:

```
M,Mn
```

```
M =

    0.5345
    0.0101
   -0.4776
```

```
Mn =

    0.5345
    0.0101
   -0.4776
```