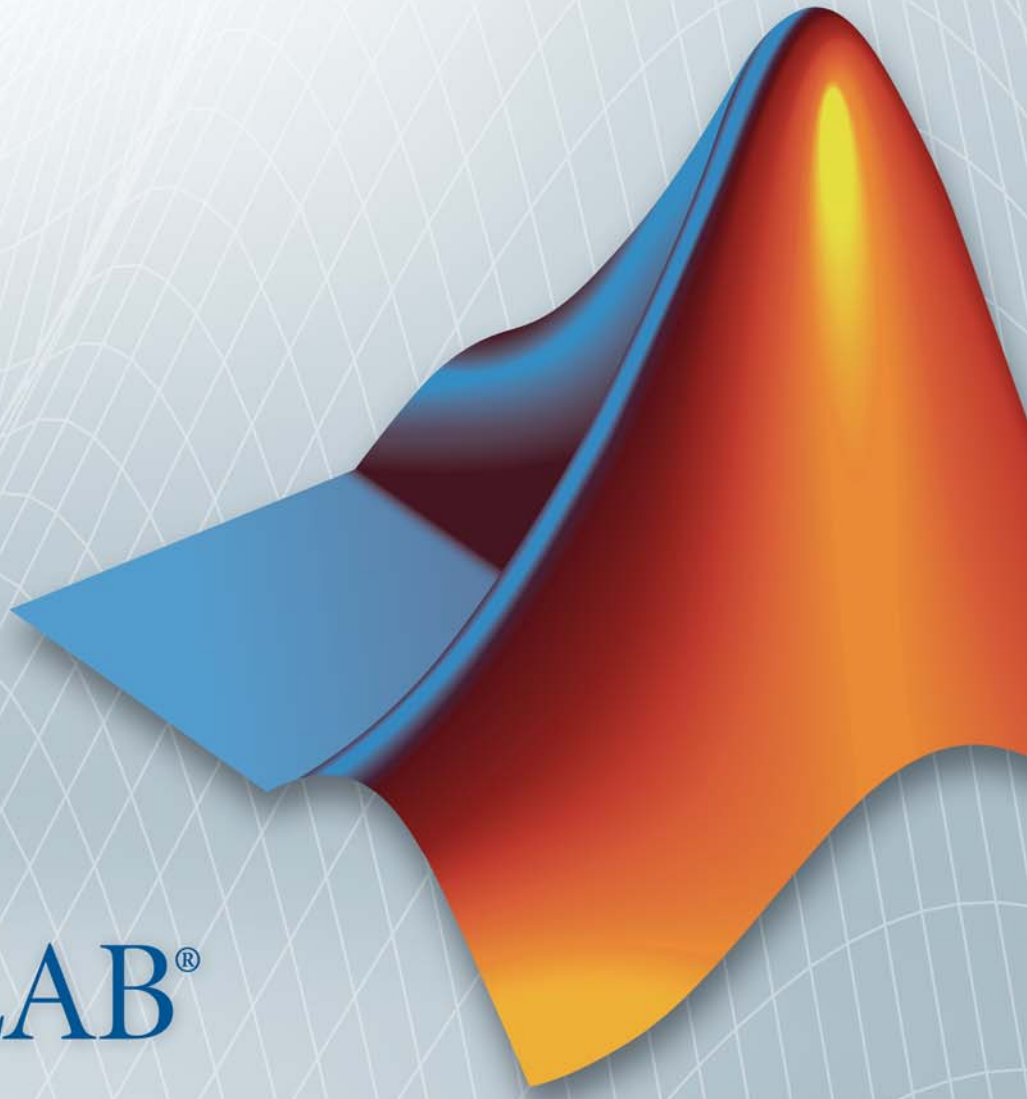
Control System Toolbox™

Getting Started Guide

R2011b



MATLAB®

MathWorks®

How to Contact MathWorks

`www.mathworks.com`	Web
`comp.soft-sys.matlab`	Newsgroup
`www.mathworks.com/contact_TS.html`	Technical Support

`suggest@mathworks.com`	Product enhancement suggestions
`bugs@mathworks.com`	Bug reports
`doc@mathworks.com`	Documentation error reports
`service@mathworks.com`	Order status, license renewals, passcodes
`info@mathworks.com`	Sales, pricing, and general information

508-647-7000 (Phone)

508-647-7001 (Fax)

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

For contact information about worldwide offices, see the MathWorks Web site.

Control System Toolbox™ Getting Started Guide

© COPYRIGHT 2000–2011 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to 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)

Contents

1 Product Overview

Why Use This Toolbox? 1-2

Using the Documentation 1-3

Related Products 1-4

2 Building Models

Linear (LTI) Models 2-2
- What Is a Plant? 2-2
- Linear Model Representations 2-2
- SISO Example: The DC Motor 2-3
- Building SISO Models 2-6
- Constructing Discrete Time Systems 2-9
- Adding Delays to Linear Models 2-10
- LTI Objects 2-10

MIMO Models 2-13
- MIMO Example: State-Space Model of Jet Transport Aircraft 2-13
- Constructing MIMO Transfer Functions 2-15
- Accessing I/O Pairs in MIMO Systems 2-17

Arrays of Linear Models 2-18

Model Characteristics 2-21

Interconnecting Linear Models 2-23

Arithmetic Operations for Interconnecting Models 2-23
Feedback Interconnections 2-24

Converting Between Continuous- and Discrete- Time Systems 2-26
Available Commands for Continuous/Discrete Conversion 2-26
Available Methods for Continuous/Discrete Conversion .. 2-26
Digitizing the Discrete DC Motor Model 2-26

Reducing Model Order 2-29
Model Order Reduction Commands 2-29
Techniques for Reducing Model Order 2-29
Example: Gasifier Model 2-30

3 Analyzing Models

Quick Start for Performing Linear Analysis Using the LTI Viewer 3-2

LTI Viewer 3-7
Plot Types Available in the LTI Viewer 3-7
Example: Time and Frequency Responses of the DC Motor 3-8
Right-Click Menus 3-10
Displaying Response Characteristics on a Plot 3-11
Changing Plot Type 3-14
Showing Multiple Response Types 3-16
Comparing Multiple Models 3-17

Simulating Models with Arbitrary Inputs and Initial Conditions 3-22
What is the Linear Simulation Tool? 3-22
Opening the Linear Simulation Tool 3-22
Working with the Linear Simulation Tool 3-23
Importing Input Signals 3-26
Example: Loading Inputs from a Microsoft® Excel Spreadsheet 3-28

Example: Importing Inputs from the Workspace 3-29
Designing Input Signals 3-33
Specifying Initial Conditions 3-35

Functions for Time and Frequency Response 3-37
When to Use Functions for Time and Frequency Response 3-37
Time and Frequency Response Functions 3-37
Plotting MIMO Model Responses 3-40
Data Markers 3-42
Plotting and Comparing Multiple Systems 3-44
Creating Custom Plots 3-47

4 Designing Compensators

Designing PID Controllers 4-2
Designing PID Controllers with the PID Tuner GUI 4-2
Designing PID Controllers at the Command Line 4-14

SISO Design Tool 4-15
Quick Start for Tuning Compensator Parameters Using the SISO Design Tool 4-15
Components of the SISO Tool 4-25
Design Options in the SISO Tool 4-25
Opening the SISO Design Tool 4-26
Using the SISO Design Task Node in the Control and Estimation Tools Manager 4-28
Importing Models into the SISO Design Tool 4-29
Feedback Structure 4-33
Analysis Plots for Loop Responses 4-35
Using the Graphical Tuning Window 4-41
Exporting the Compensator and Models 4-48
Storing and Retrieving Intermediate Designs 4-50

Bode Diagram Design 4-54
What Is Bode Diagram Design? 4-54
Example: DC Motor 4-54
Adjusting the Compensator Gain 4-55

Adjusting the Bandwidth 4-56
Adding an Integrator 4-58
Adding a Lead Network 4-62
Moving Compensator Poles and Zeros 4-67
Adding a Notch Filter 4-70
Modifying a Prefilter 4-75

Root Locus Design 4-80
What Is Root Locus Design? 4-80
Example: Electrohydraulic Servomechanism 4-81
Changing the Compensator Gain 4-87
Adding Poles and Zeros to the Compensator 4-90
Editing Compensator Pole and Zero Locations 4-95
Viewing Damping Ratios 4-99

Nichols Plot Design 4-102
What Is Nichols Plot Design? 4-102
DC Motor Example 4-102
Opening a Nichols Plot 4-103
Adjusting the Compensator Gain 4-104
Adding an Integrator 4-107
Adding a Lead Network 4-109

Automated Tuning Design 4-113
Supported Automated Tuning Methods 4-113
Loading and Displaying the DC Motor Example for Automated Tuning 4-113
Applying Automated PID Tuning 4-115

Multi-Loop Compensator Design 4-119
When to Use Multi-Loop Compensator Design 4-119
Workflow for Multi-Loop Compensator Design 4-119
Example: Position Control of a DC Motor 4-119

Control Design Analysis of Multiple Models 4-131
Multiple Models Represent System Variations 4-131
Control Design Analysis Using the SISO Design Tool 4-131
Specifying a Nominal Model 4-132
Frequency Grid for Multimodel Computations 4-135
How to Analyze the Controller Design for Multiple Models 4-136

Functions for Compensator Design **4-146**
When to Use Functions for Compensator Design **4-146**
Root Locus Design **4-146**
Pole Placement **4-147**
Linear-Quadratic-Gaussian (LQG) Design **4-150**
Example — Designing an LQG Regulator **4-161**
Example — Designing an LQG Servo Controller **4-164**
Example — Designing an LQR Servo Controller in Simulink **4-167**

Index

1

Product Overview

- "Why Use This Toolbox?" on page 1-2
- "Using the Documentation" on page 1-3
- "Related Products" on page 1-4

Why Use This Toolbox?

Control System Toolbox™ provides industry-standard algorithms and tools for systematically analyzing, designing, and tuning linear control systems. You can specify your system as a transfer function, state-space, pole-zero-gain, or frequency-response model. Interactive tools and command-line 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.

Using the Documentation

If you are new to Control System Toolbox, review the *Control System Toolbox Getting Started Guide* to learn how to:

- Build and manipulate linear time-invariant models of dynamical systems
- Analyze such models and plot their time and frequency responses
- Design compensators using root locus and pole placement techniques

This guide also discusses model order reduction and linear quadratic Gaussian (LQG) control design techniques with examples.

Other documentation available for this toolbox includes:

- Release Notes — Highlights new and changed features in the latest product release.
- "Linear System Modeling" — In-depth information about creating linear models and model arrays.
- "Working with Linear Models" — Information about operations you can perform on linear models and techniques for analyzing linear models.
- "Customization" — Information about setting plot properties, including setting preferences that persist from session to session.
- "Design Case Studies" — Worked examples of control design problems, including Kalman filtering and Linear-Quadratic-Gaussian design. The examples illustrate design techniques in both single-input/single-output (SISO) and multiple-input/multiple-output (MIMO) systems.
- "Reliable Computations" — Discussion of numerical stability and accuracy in computation
- "Using the SISO Design Tool and the LTI Viewer" — Complete descriptions of the LTI Viewer and SISO Design Tool. These tools help you analyze systems and design SISO compensators.
- Function Reference — Describes the purpose, syntax, and options for each function.

Related Products

The following table summarizes MathWorks® products that extend and complement the Control System Toolbox software. For current information about these products and other MathWorks products, point your Web browser to:

```
www.mathworks.com
```

Product	Description
Simulink®	Provides blocks for simulating models you create and analyze using the Control System Toolbox software.
Fuzzy Logic Toolbox™	Complements Control System Toolbox software with tools for modeling complex systems based on fuzzy logic.
Model Predictive Control Toolbox™	Helps you design and simulate model predictive controllers for linear plant models created using Control System Toolbox functions.
Robust Control Toolbox™	Helps you design robust control systems based on plant models created using the Control System Toolbox software. Enables you to assess robustness based on confidence bounds for your plant models.
Simulink® Control Design™	Lets you use Control System Toolbox linear control tools to analyze and tune Simulink models.

Product	Description
Simulink® Design Optimization™	Helps you improve controller designs by estimating and tuning model parameters using numerical optimization.
"System Identification Toolbox™"	Helps you estimate linear and nonlinear mathematical models of dynamic systems from measured data. Use the Control System Toolbox software to analyze and design controllers for resulting linear models.

2

Building Models

- "Linear (LTI) Models" on page 2-2
- "MIMO Models" on page 2-13
- "Arrays of Linear Models" on page 2-18
- "Model Characteristics" on page 2-21
- "Interconnecting Linear Models" on page 2-23
- "Converting Between Continuous- and Discrete- Time Systems" on page 2-26
- "Reducing Model Order" on page 2-29

Linear (LTI) Models

In this section...

"What Is a Plant?" on page 2-2
"Linear Model Representations" on page 2-2
"SISO Example: The DC Motor" on page 2-3
"Building SISO Models" on page 2-6
"Constructing Discrete Time Systems" on page 2-9
"Adding Delays to Linear Models" on page 2-10
"LTI Objects" on page 2-10

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,

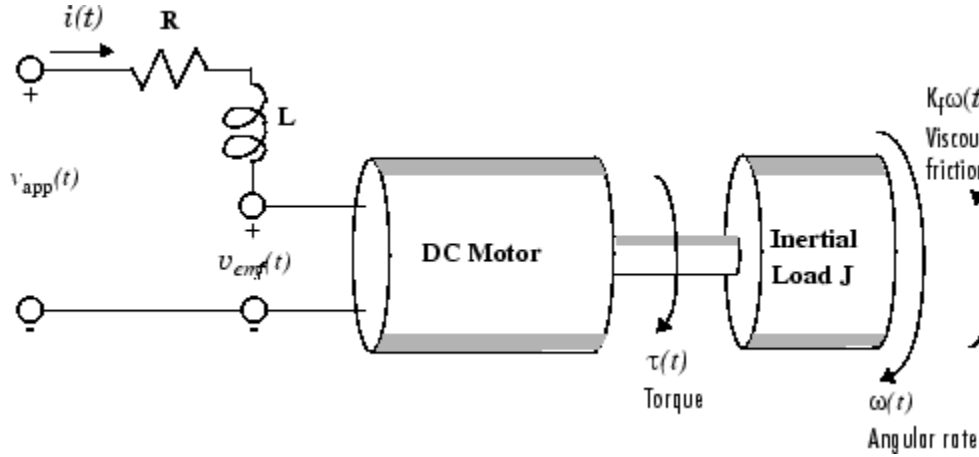
$$H(z) = 3\frac{(z+1+j)(z+1-j)}{(z+0.2)(z+0.1)}$$

- 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,

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

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.

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

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

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. See Reliable Computations for more information on numerical issues.

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;
```

For more information on LTI properties, type `ltiprops` at the MATLAB prompt. For a complete description of LTI objects, see "Linear System Modeling".

MIMO Models

In this section...

"MIMO Example: State-Space Model of Jet Transport Aircraft" on page 2-13

"Constructing MIMO Transfer Functions" on page 2-15

"Accessing I/O Pairs in MIMO Systems" on page 2-17

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. See "Model Interconnections" for more details on concatenation operations for LTI systems.

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

$$N(s) = \left[\frac{s-1}{s+2} \right] \qquad D(s) = \left[\frac{s+1}{s^2+4s+5} \right]$$

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 ijth 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.

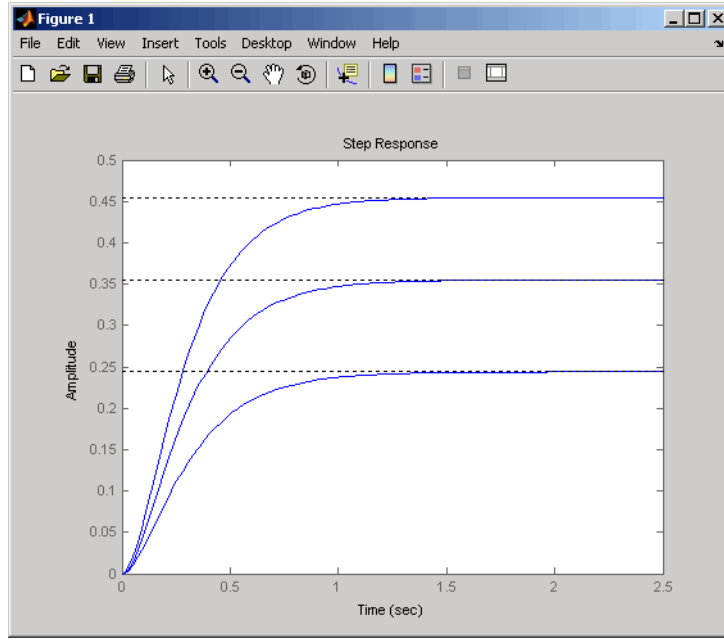
```
Model sys_lti(:,:,1,1)
======================

  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
`lti/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

Commands to Query Model Characteristics (Continued)

Command	Description
`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)
`lti/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 2-23

"Feedback Interconnections" on page 2-24

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 "Precedence 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

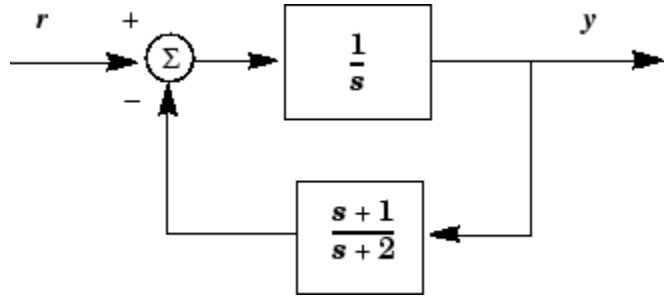
You can use the `feedback` and `lft` functions to derive closed-loop models. For example,

```
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" on page 2-26

"Available Methods for Continuous/Discrete Conversion" on page 2-26

"Digitizing the Discrete DC Motor Model" on page 2-26

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.
```

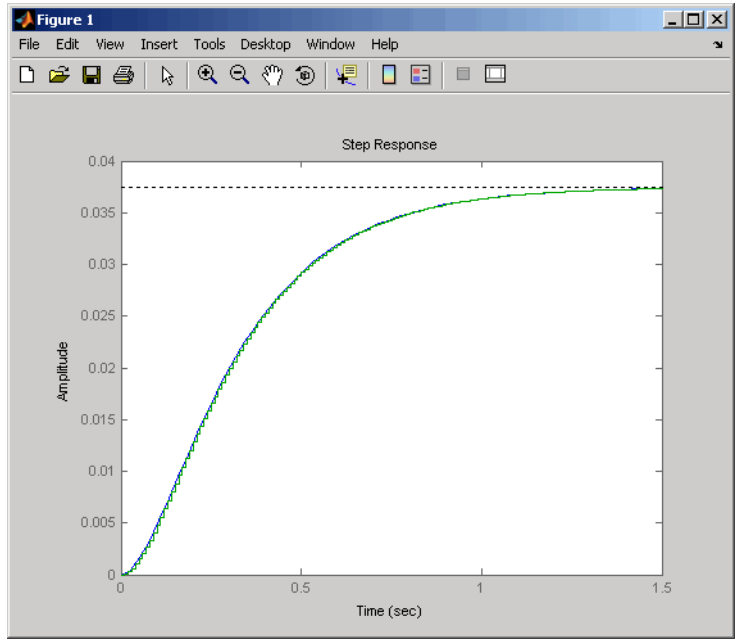
To see the discrete-time zero-pole gain for the digital DC motor, use `zpk` to convert the model.

```
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" on page 2-29
"Techniques for Reducing Model Order" on page 2-29
"Example: Gasifier Model" on page 2-30

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 2-34.

This example focuses on a single input/output pair of the gasifier, input `5` to output `3`.

```
sys35 = gasf(3,5);
```

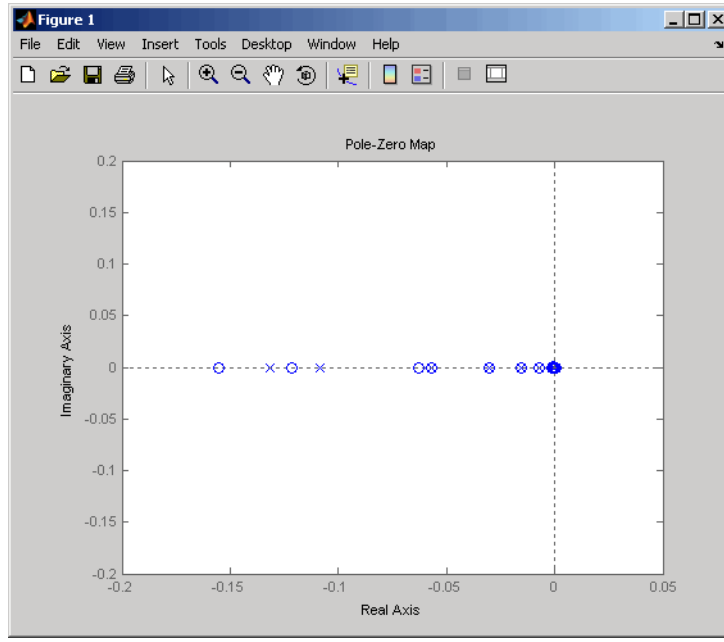
Before attempting model order reduction, inspect the pole and zero locations by typing

```
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.

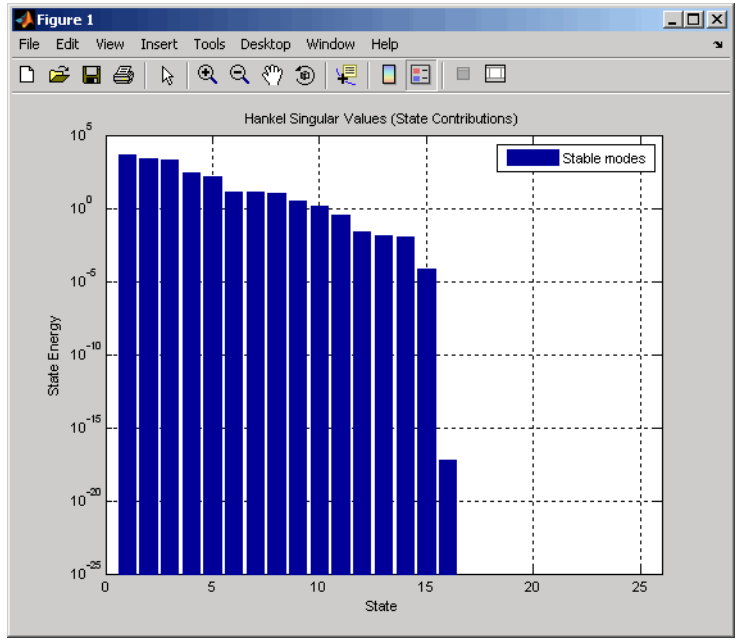
To find a low-order reduction of this SISO model, perform the following steps:

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

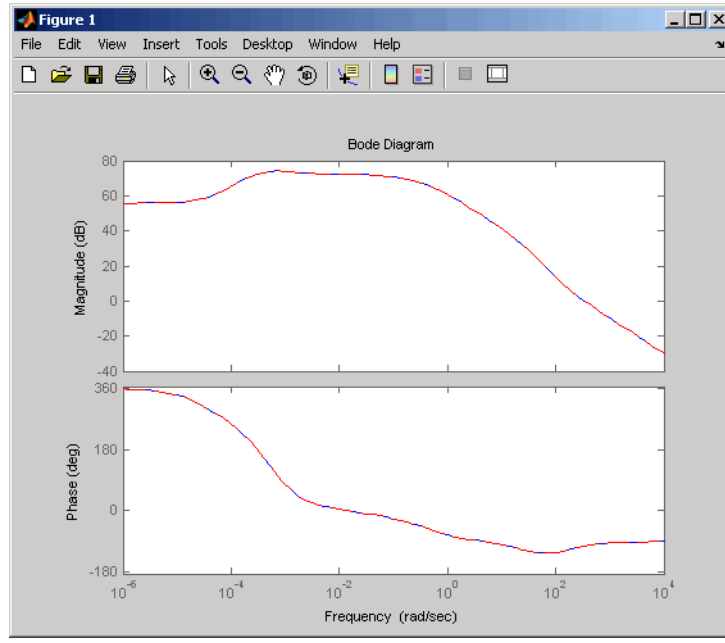
```
rsys35 = balred(sys35,15);
```

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.



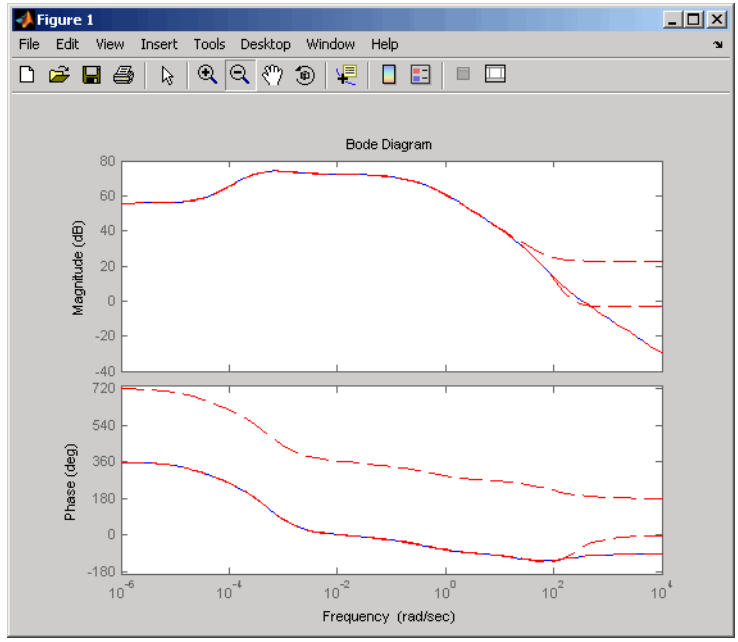
As the overlap of the curves in the figure shows, the reduced model provides a good approximation of the original system.

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

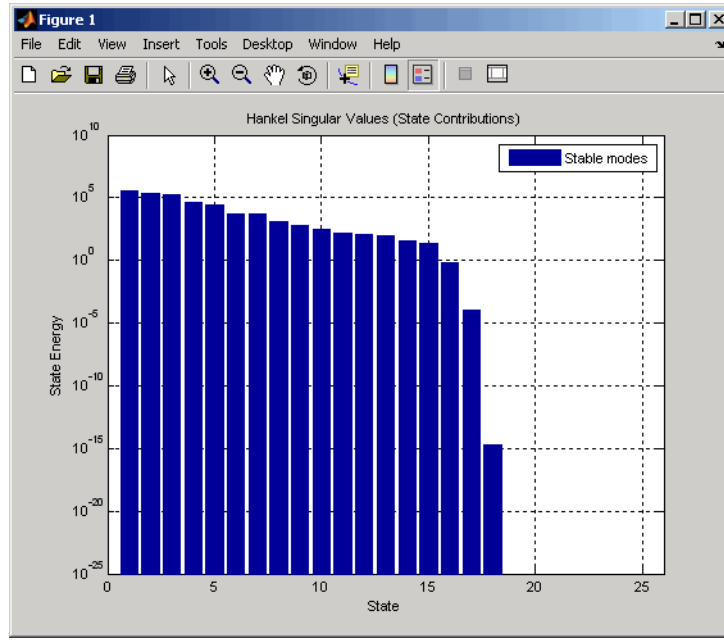
You can approximate MIMO models using the same steps as SISO models as follows:

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

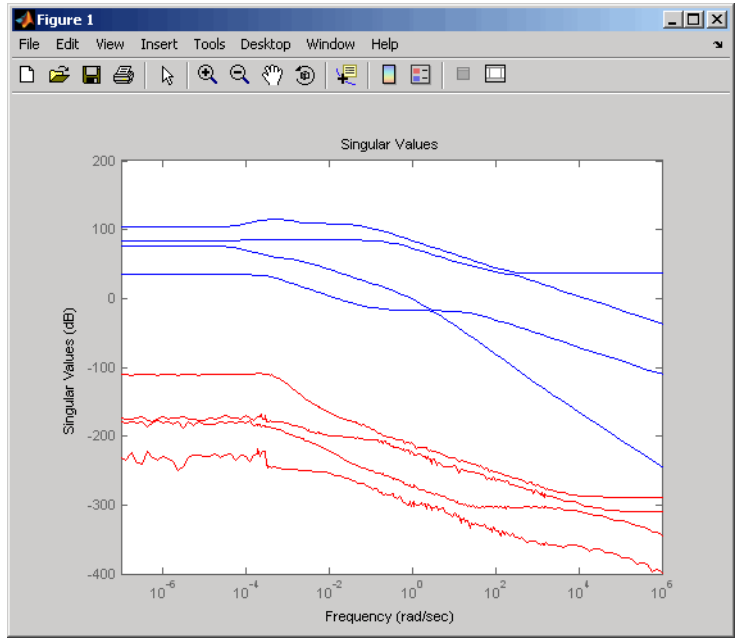
```
rsys=balred(gasf,17);
```

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

MathWorks thanks ALSTOM® Power UK for permitting use of their gasifier model for this example. This model was issued as part of the ALSTOM Benchmark Challenge on Gasifier Control. For more details see Dixon, R., (1999), "Advanced Gasifier Control," *Computing & Control Engineering Journal*, IEE, Vol. 10, No. 3, pp. 92–96.

3

Analyzing Models

- "Quick Start for Performing Linear Analysis Using the LTI Viewer" on page 3-2
- "LTI Viewer" on page 3-7
- "Simulating Models with Arbitrary Inputs and Initial Conditions" on page 3-22
- "Functions for Time and Frequency Response" on page 3-37

Quick Start for Performing Linear Analysis Using the LTI Viewer

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

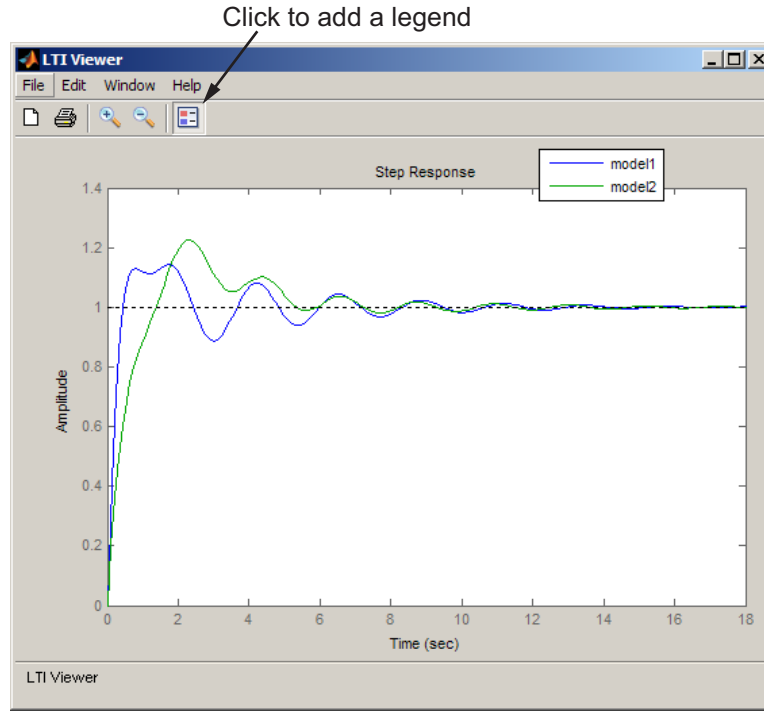
Before you can perform the analysis, you must have already created linear models in the MATLAB workspace. For information on how to create a model, see Chapter 2, "Building Models".

To perform linear analysis:

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

   ```
   ltiview(model1,model2,...,modelN)
   ```

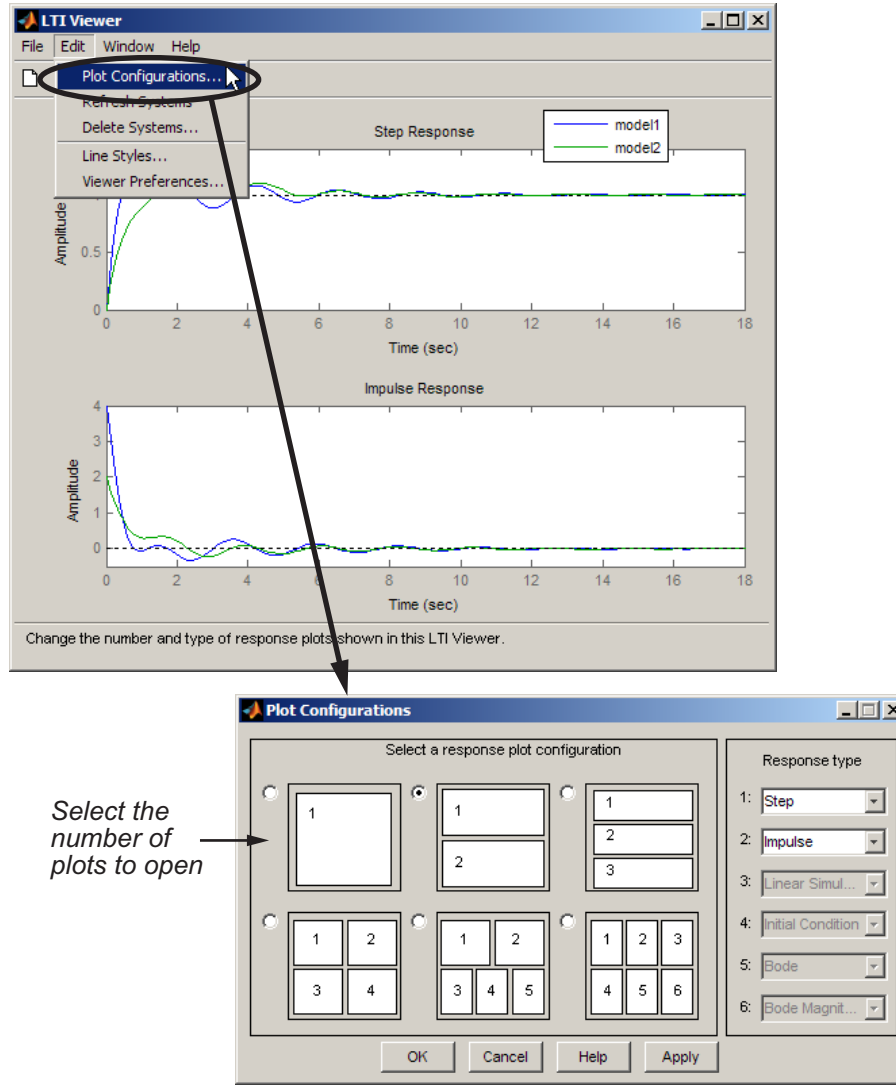
   By default, this syntax opens a step response plot of your models, as shown in the following figure.


Click to add a legend
LTI Viewer
File
Edit
Window
Help
Step Response
model1
model2
Amplitude
Time (sec)
0
0.2
0.4
0.6
0.8
1
1.2
1.4
0
2
4
6
8
10
12
14
16
18
LTI Viewer

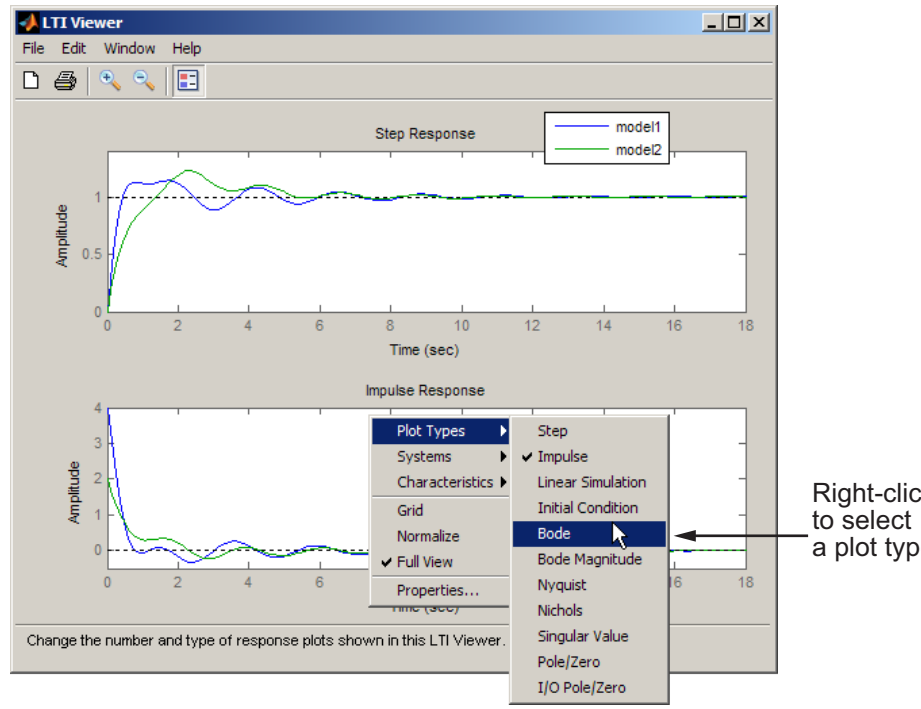
2 Add more plots to the LTI Viewer.

a Select **Edit > Plot Configurations**.

b In the Plot Configurations dialog box, select the number of plots to open.



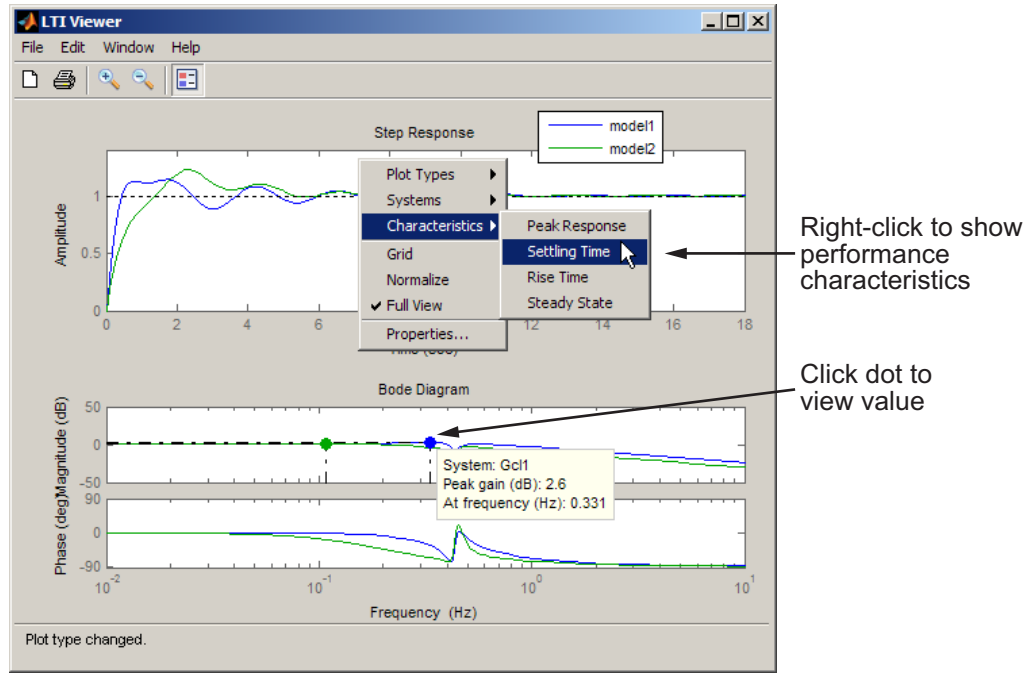
3 To view a different type of response on a plot, right-click and select a plot type.



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.



LTI Viewer

In this section...

"Plot Types Available in the LTI Viewer" on page 3-7
"Example: Time and Frequency Responses of the DC Motor" on page 3-8
"Right-Click Menus" on page 3-10
"Displaying Response Characteristics on a Plot" on page 3-11
"Changing Plot Type" on page 3-14
"Showing Multiple Response Types" on page 3-16
"Comparing Multiple Models" on page 3-17

Plot Types Available in the LTI Viewer

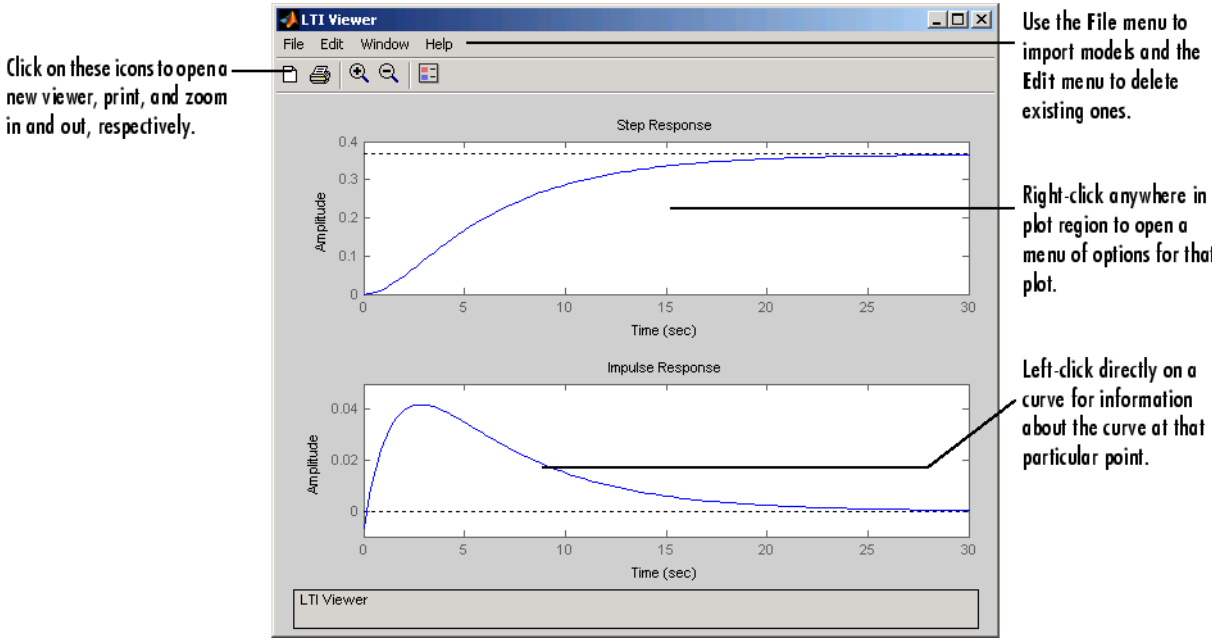
The LTI Viewer is a GUI for viewing and manipulating the response plots of linear models. You can display the following plot types for linear models using the LTI Viewer:

- 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.



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

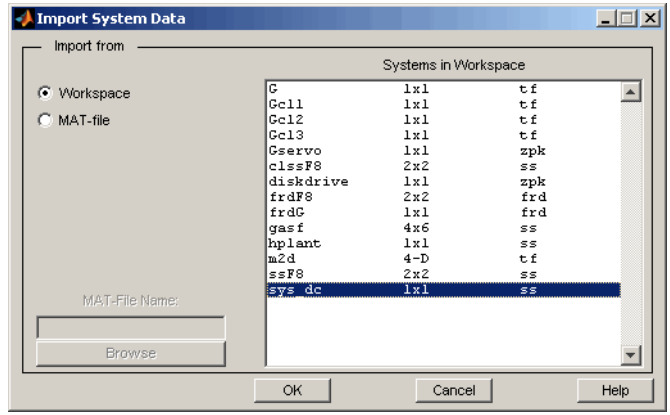
To open the LTI Viewer, type

```
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.



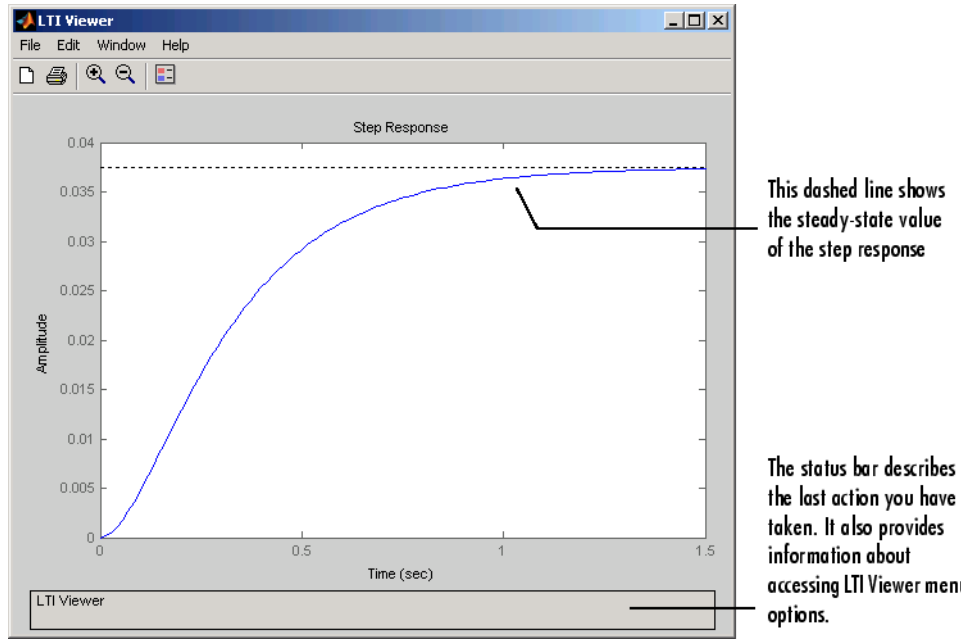
Import System Data Dialog Box with the DC Motor Model Selected

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.



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)
```

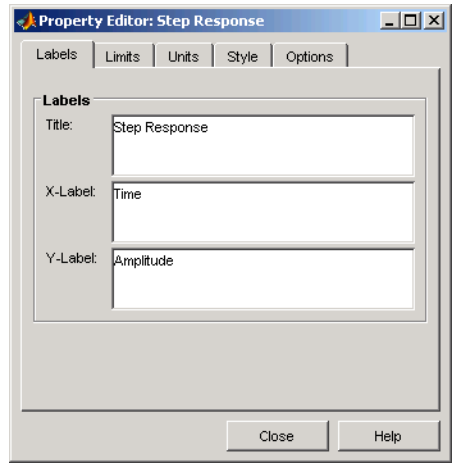
See the `ltiview` reference page for a complete list of options.

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.

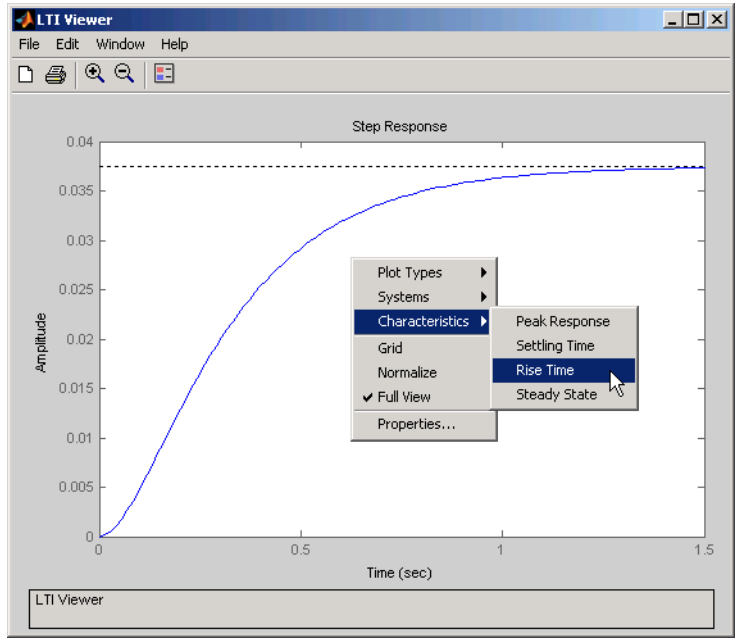


You can use this editor to customize various attributes of your plot. See Customizing Plot Properties and Preferences for a full description of the Property Editor.

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

Displaying Response Characteristics on a Plot

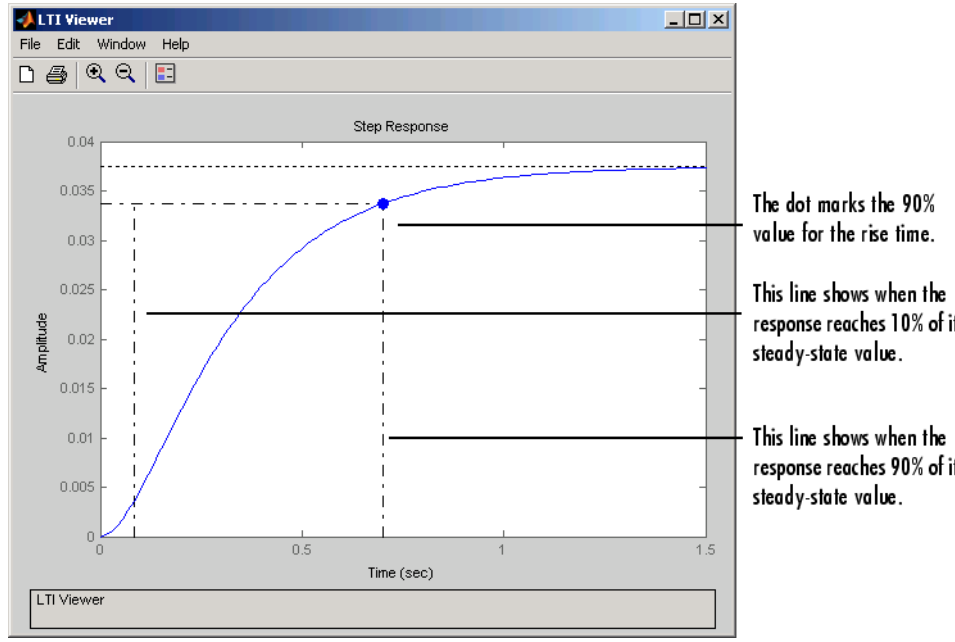
For example, to see the rise time for the DC motor step response, right-click and select **Characteristics > Rise Time**.



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.

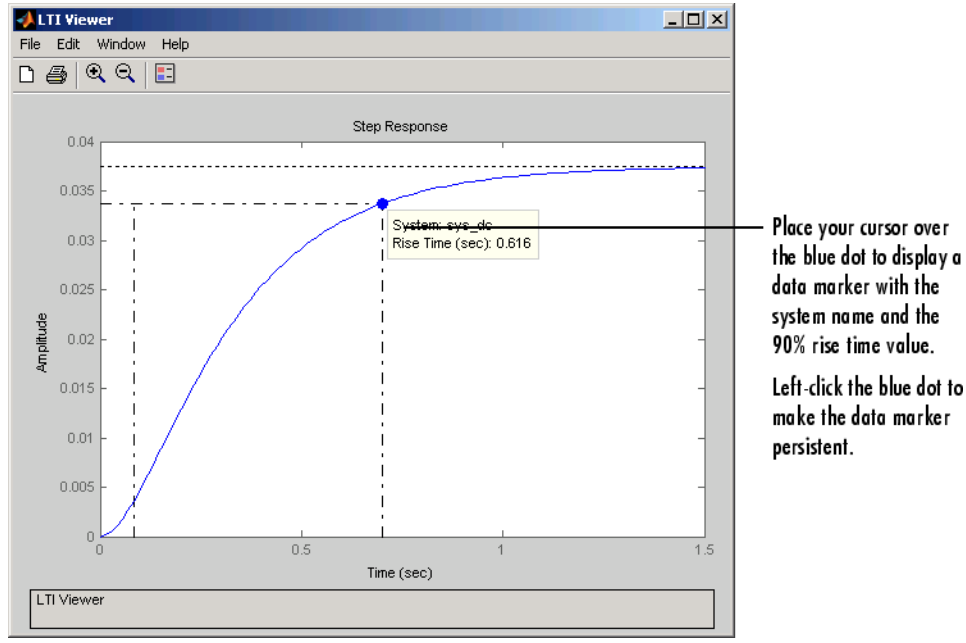
The LTI Viewer calculates and displays the rise time for the step response.



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.



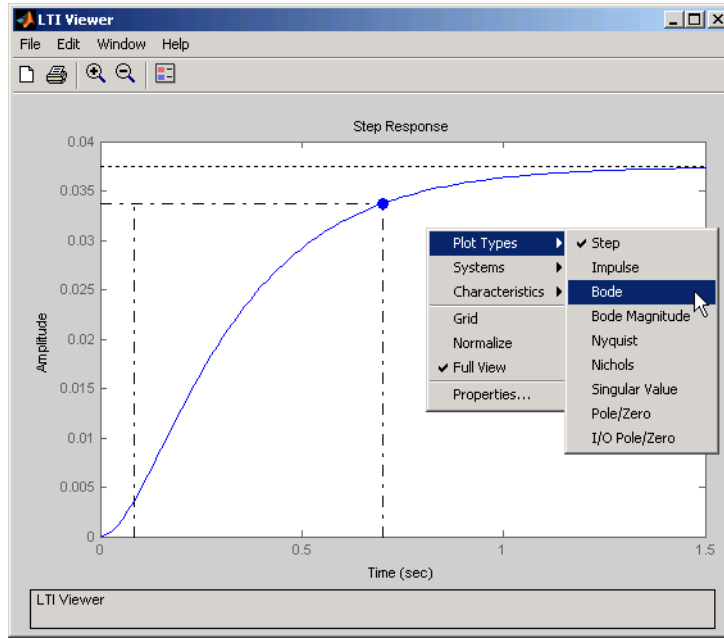
Using Your Mouse to Get the Rise Time Values

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 3-42.

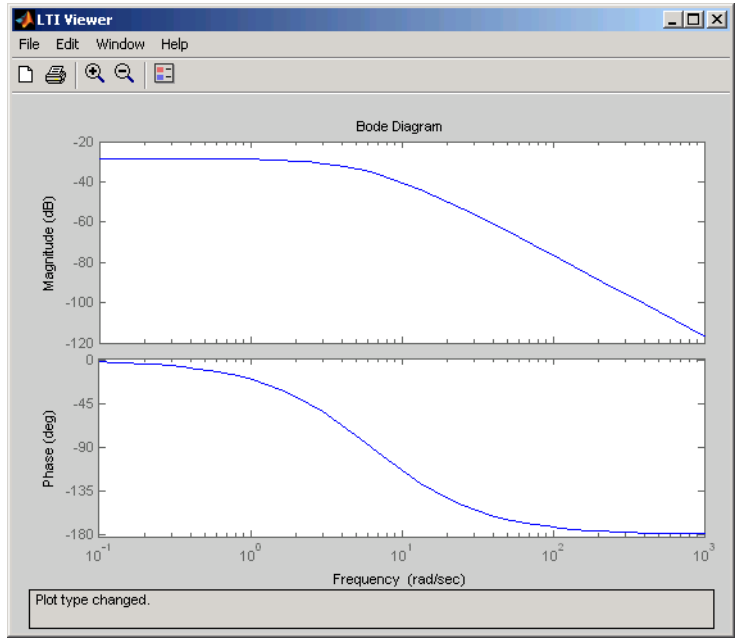
Changing Plot Type

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

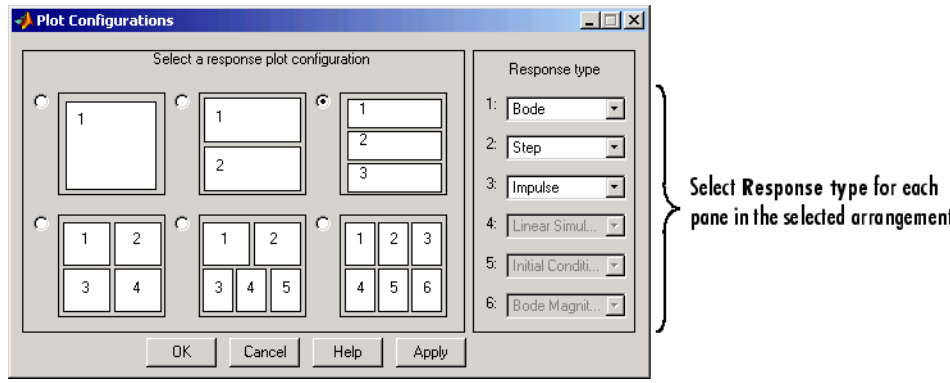
Selecting **Bode** changes the step response to a Bode plot for the DC motor model.



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 Configurations** 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.

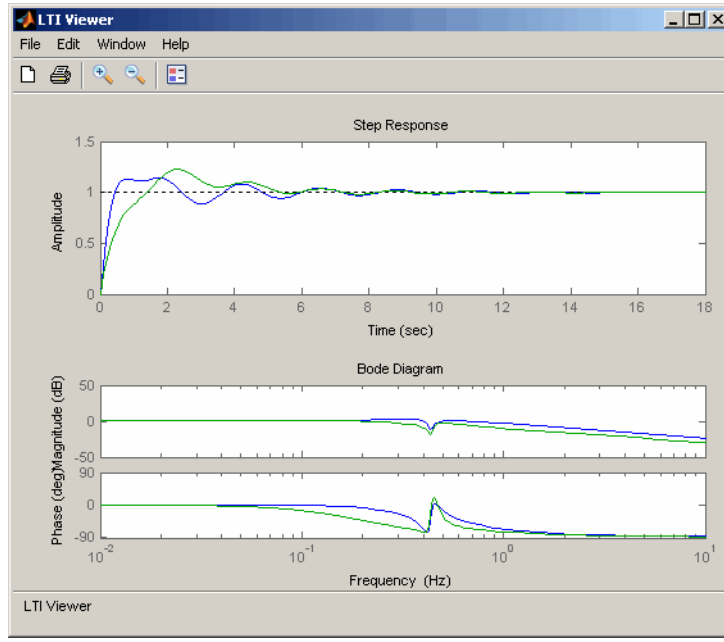
Initializing the LTI Viewer with Multiple Plots

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 3-9 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 3-16 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

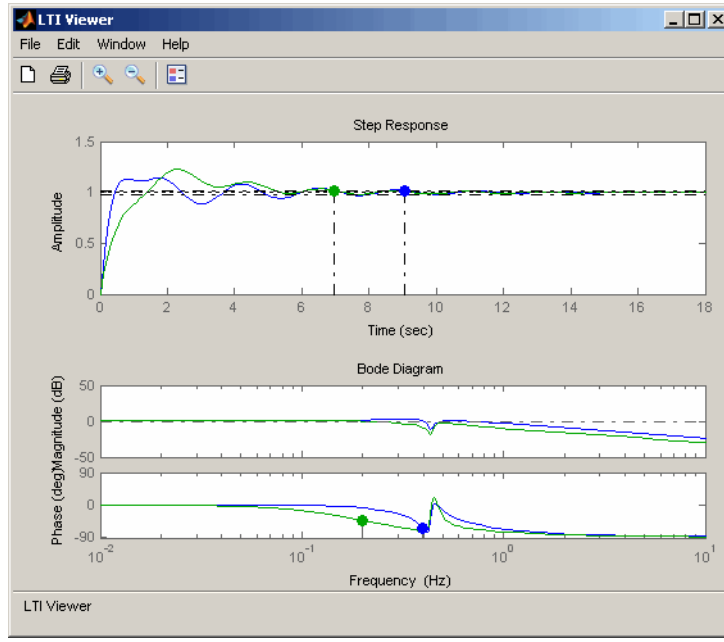
Inspecting Response Characteristics

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

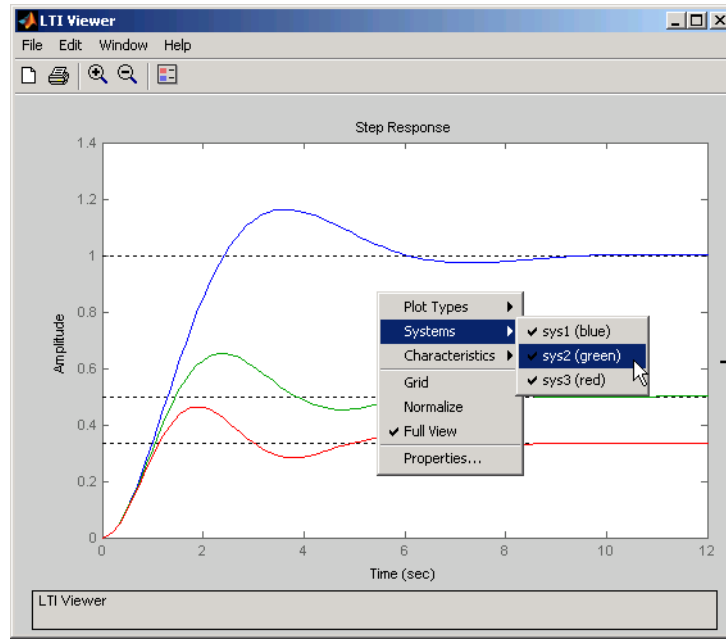
The minimum stability margins, meaning the smallest magnitude phase and gain margins, are displayed as green and blue markers on the Bode phase diagram. If you want to see all the gain and phase margins of a system, right-click and select **Characteristics > Minimum Stability Margins**.

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.

Simulating Models with Arbitrary Inputs and Initial Conditions

In this section...

"What is the Linear Simulation Tool?" on page 3-22

"Opening the Linear Simulation Tool" on page 3-22

"Working with the Linear Simulation Tool" on page 3-23

"Importing Input Signals" on page 3-26

"Example: Loading Inputs from a Microsoft® Excel Spreadsheet" on page 3-28

"Example: Importing Inputs from the Workspace" on page 3-29

"Designing Input Signals" on page 3-33

"Specifying Initial Conditions" on page 3-35

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)
```

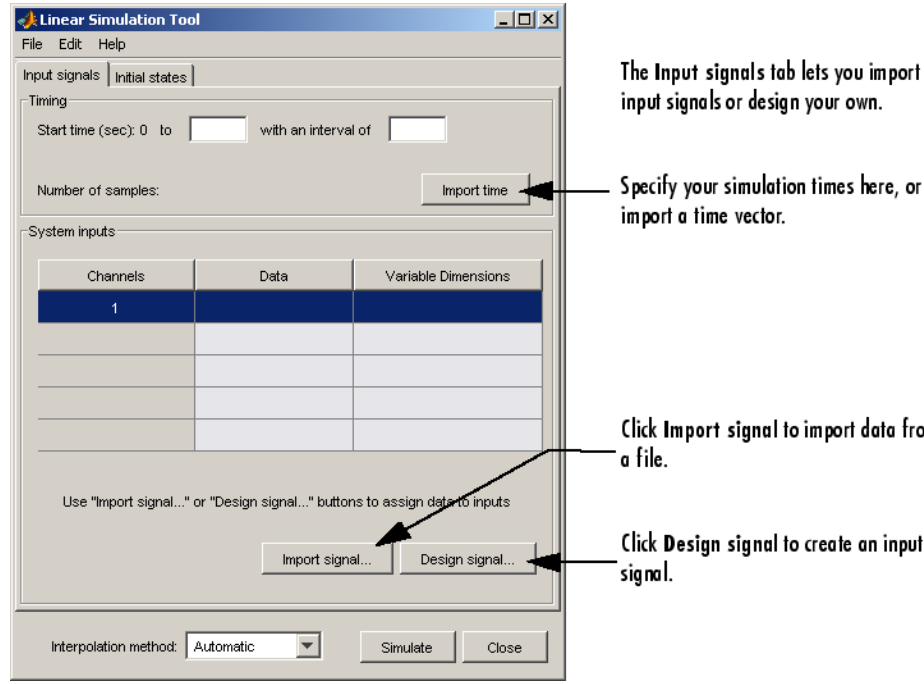
- In the MATLAB Figure window, right-click a response plot and select **Input data**.

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 3-22), follow these steps to simulate your model:

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



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

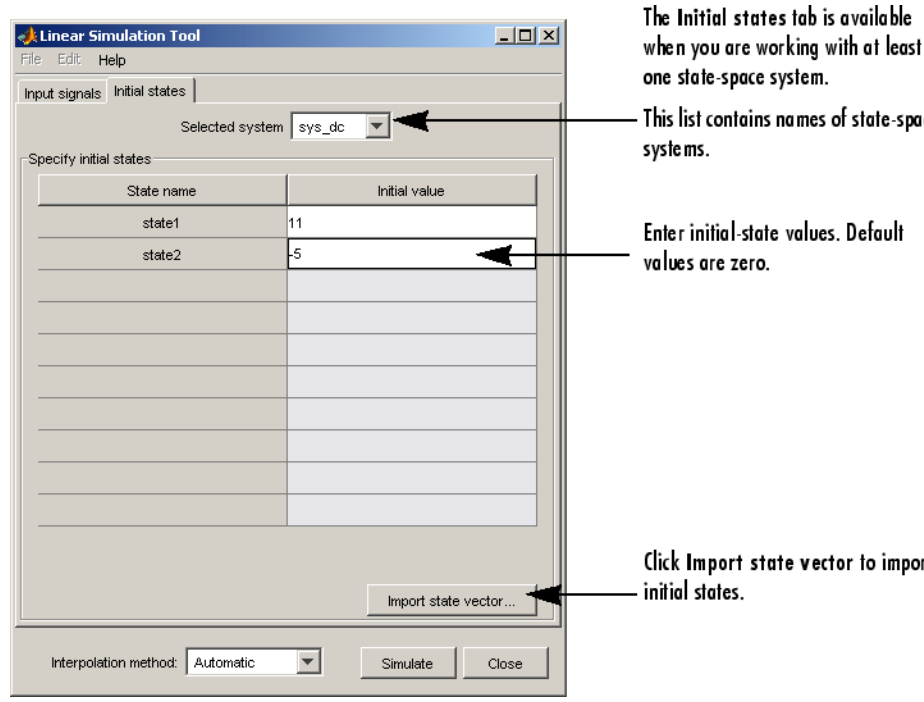
- Import the time vector by clicking **Import time**.
- Enter the end time and the time interval in seconds. The start time is set to `0` seconds.

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 3-26.
- Click **Design signal** to create your own inputs. For more information, see "Designing Input Signals" on page 3-33.

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 3-35.



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**.

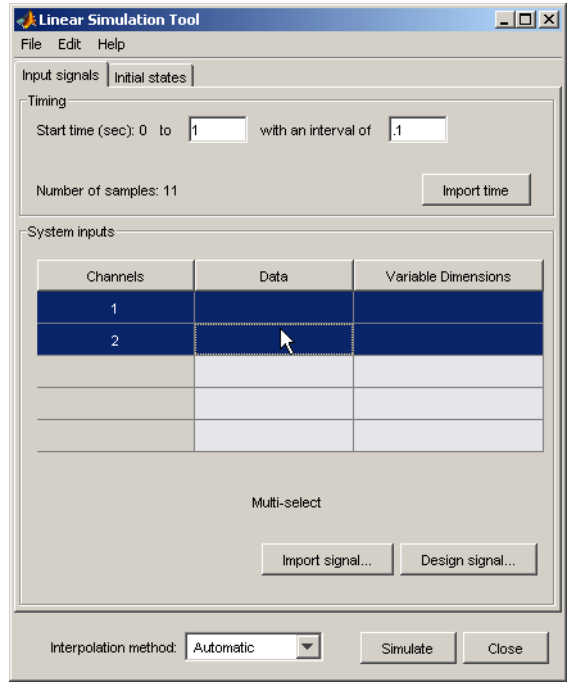
Importing Input Signals

You can import input signals from the MATLAB workspace after opening the Linear Simulation Tool (see “Opening the Linear Simulation Tool” on page 3-22). You can also import inputs from a MAT-file, Microsoft Excel spreadsheet, ASCII flat-file, comma-separated variable file (CSV), or text file.

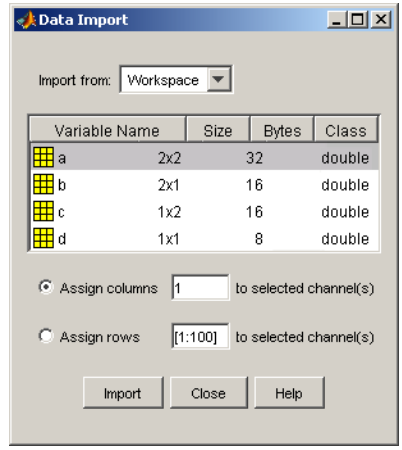
For information about creating your own inputs, see “Designing Input Signals” on page 3-33. For an overview of working with the Linear Simulation Tool, see “Working with the Linear Simulation Tool” on page 3-23.

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 3-28
- "Example: Importing Inputs from the Workspace" on page 3-29

Example: Loading Inputs from a Microsoft Excel Spreadsheet

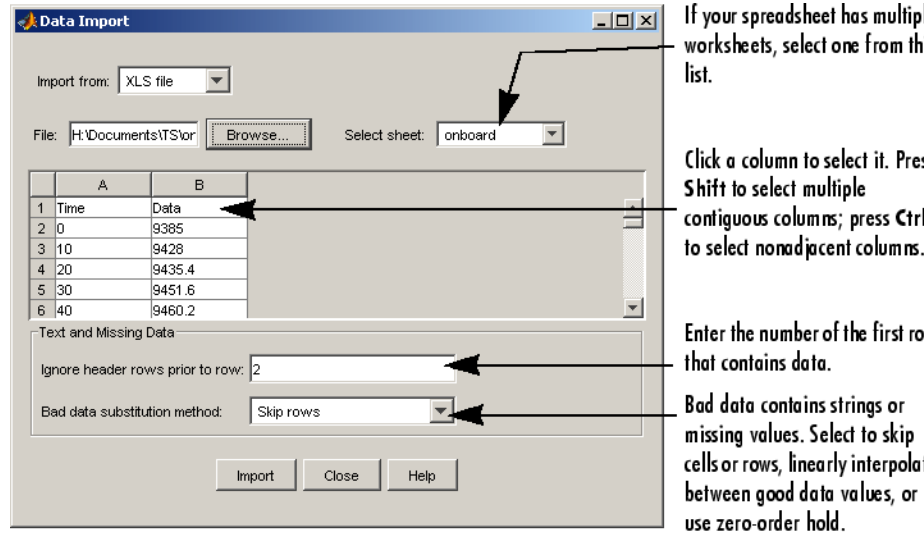
To load inputs from a Microsoft Excel (XLS) spreadsheet:

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.



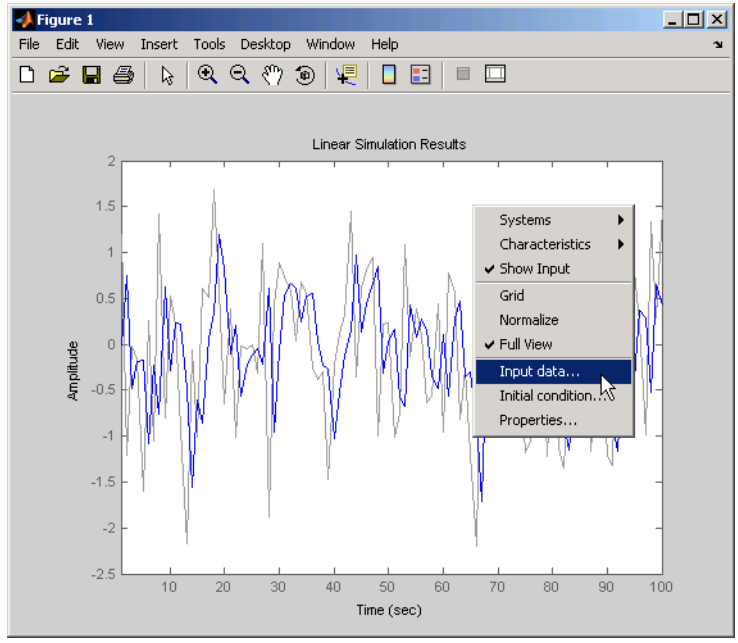
Example: Importing Inputs from the Workspace

To load an input signal from the MATLAB workspace:

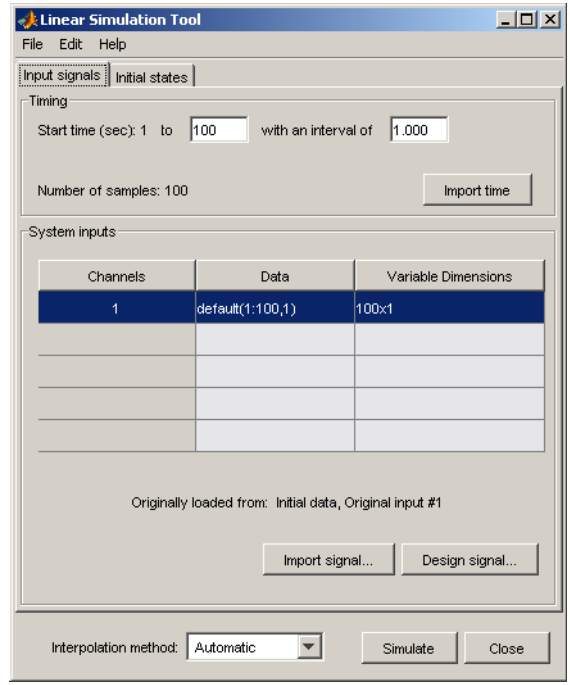
1. Enter this code to open a response plot with a second-order system:

```
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.

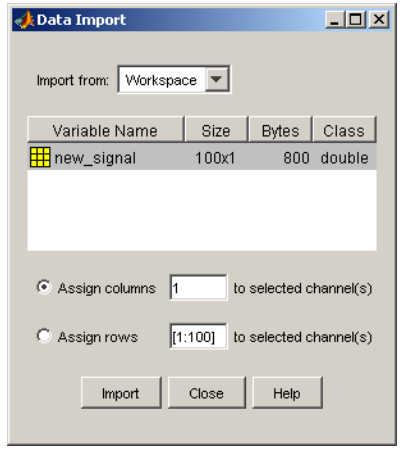


3 Create an input signal for your system in the MATLAB Command Window, such as the following:

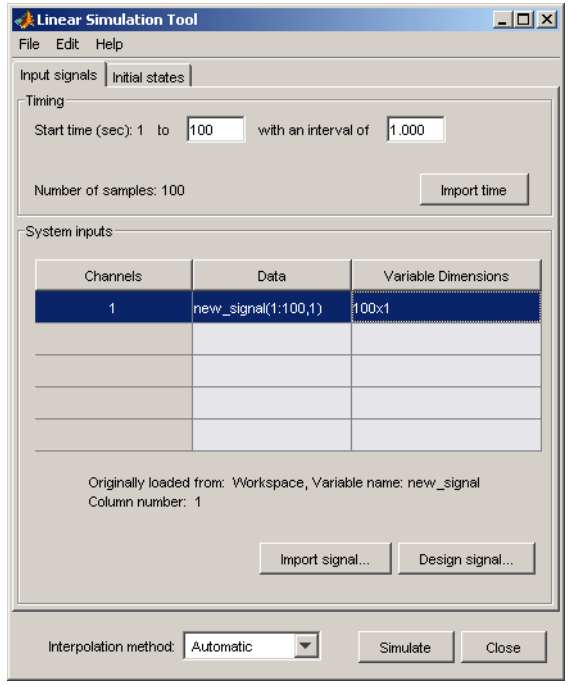
```
new_signal=[-3*ones(1,20) 2*ones(1,30) 0.5*ones(1,50)]';
```

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

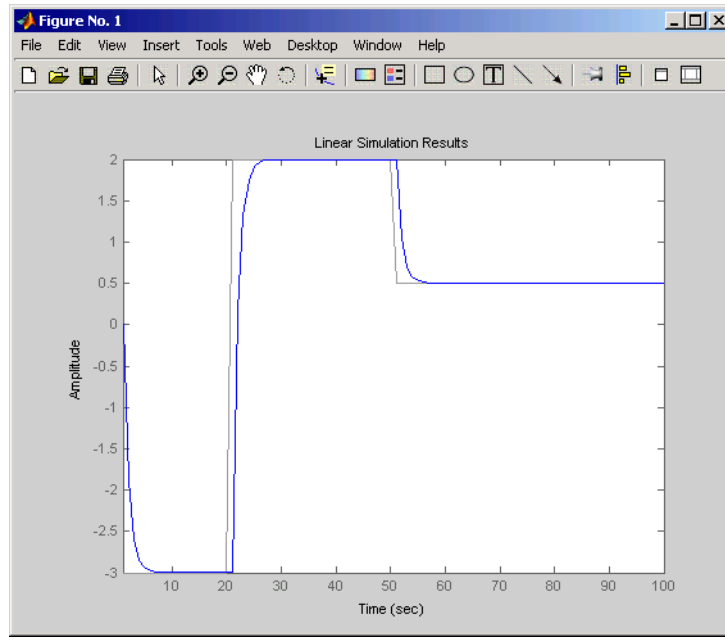
5 In the Data Import dialog box, click, **Assign columns** to assign the first column of the input signal to the selected channel.



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 3-22).

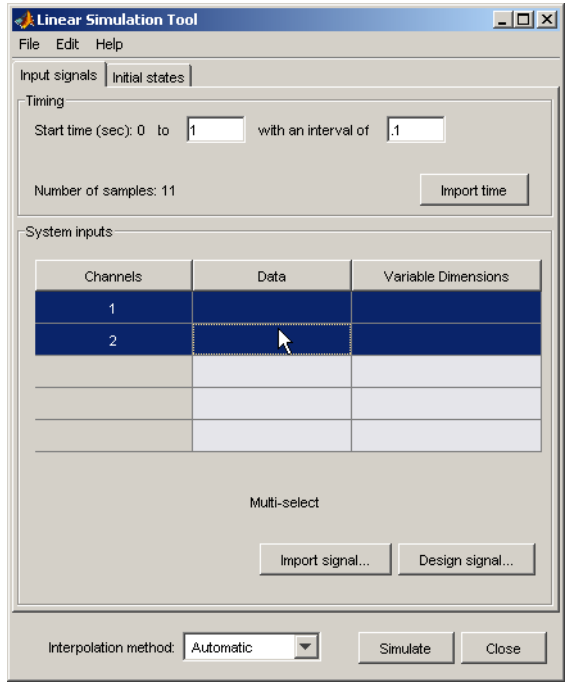
For information about importing inputs from the MATLAB workspace or from a file, see “Importing Input Signals” on page 3-26. For an overview of working with the Linear Simulation Tool, see “Working with the Linear Simulation Tool” on page 3-23.

To design one or more input signals:

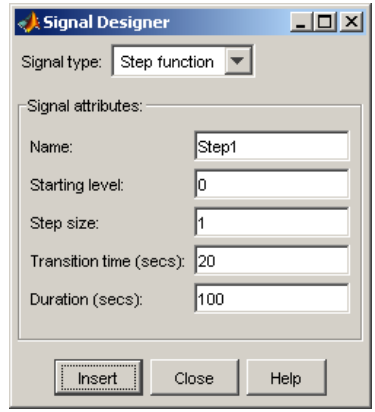
1 In the Linear Simulation Tool, click the **Input signals** tab (if it is not displayed).

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 3-22).

For an overview of working with the Linear Simulation Tool, see “Working with the Linear Simulation Tool” on page 3-23.

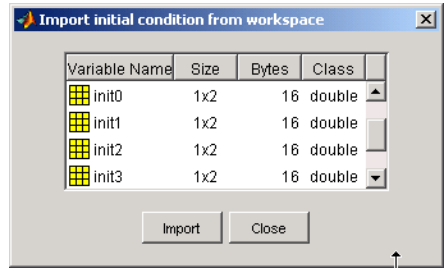
You can also import initial states from the MATLAB workspace.

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" on page 3-37

"Time and Frequency Response Functions" on page 3-37

"Plotting MIMO Model Responses" on page 3-40

"Data Markers" on page 3-42

"Plotting and Comparing Multiple Systems" on page 3-44

"Creating Custom Plots" on page 3-47

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

Functions for Frequency and Time Response (Continued)

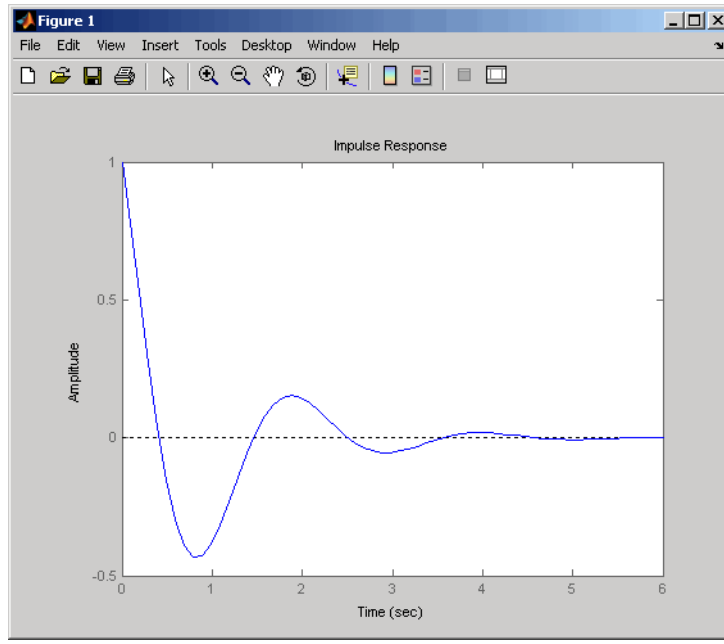
Functions	Description
`hsvd`	Compute Hankel singular values of LTI model
`bodemag`	Bode magnitude response of LTI models

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

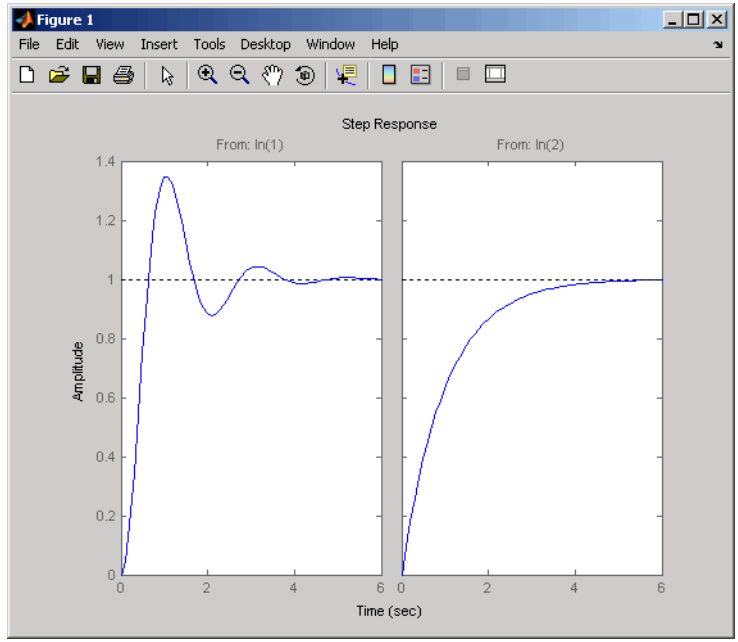
Frequency-domain plots automatically generate an appropriate frequency range as well.

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,

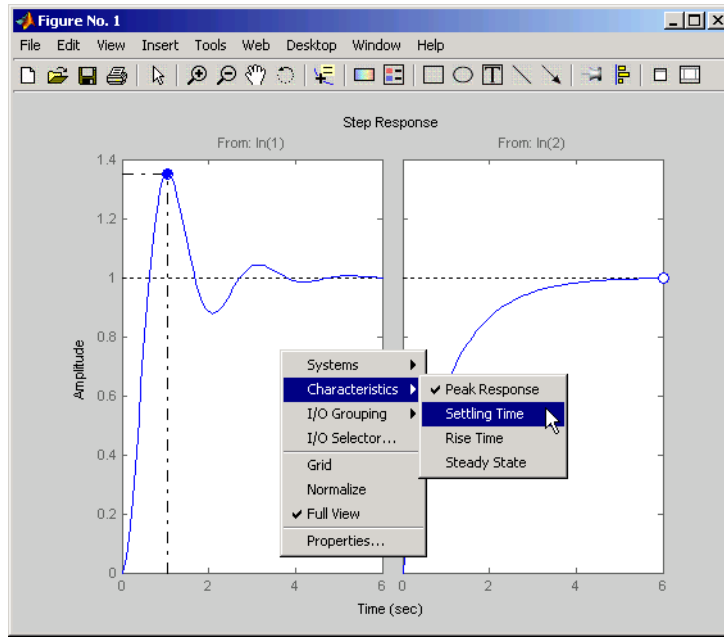
```
step(h,10) % Simulates from 0 to 10 seconds
```

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

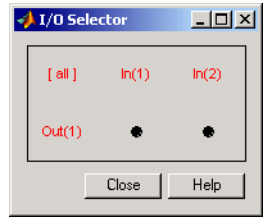
All the time and frequency response functions provide right-click menus that allow you to customize your plots. For more information on using the LTI Viewer right-click menus, see "Using the Right-Click Menu in the LTI Viewer". This figure shows the plots from Step Responses for a MIMO Model on page 3-40, 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.

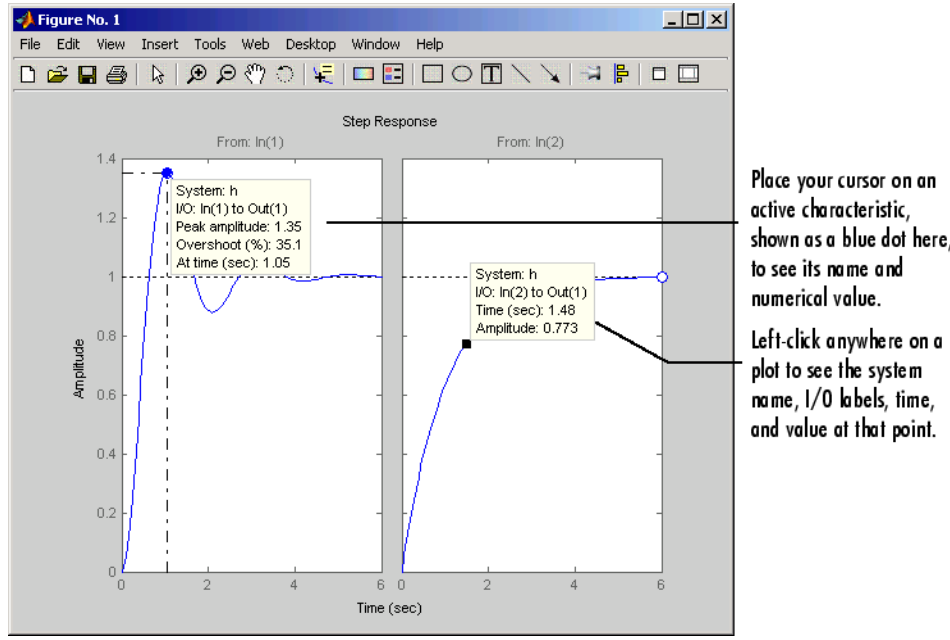


  Use this dialog box to select/clear which inputs and outputs to plot.
- **Normalize** — Scale responses to fit the view (only available for time-domain plot types).
- **Full View** — Use automatic limits to make the entire curve visible.
- **Grid** — Add grids to your plots.
- **Properties** — Open the 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 on page 3-40, shows markers on the plots.



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.

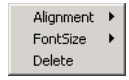
Right-Click Menus

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.

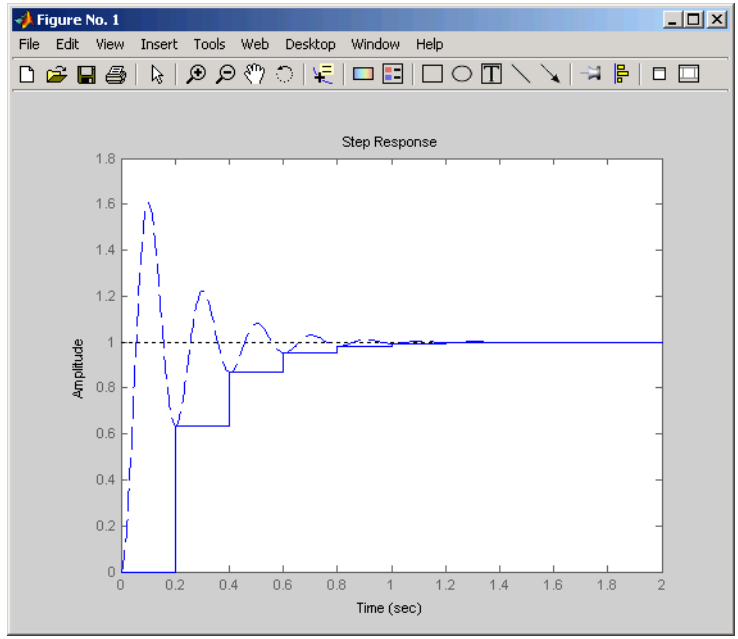
You can plot responses of multiple models on the same plot. These models do not need to be all continuous-time or all discrete-time.

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.

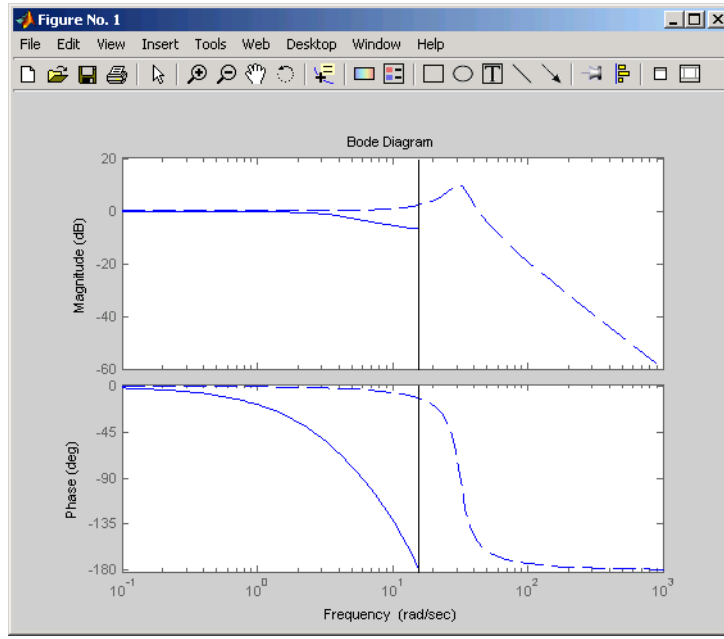


Comparison of a Continuous Model to Its Discretized Version

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 `plot`, `subplot`, and `hold`.

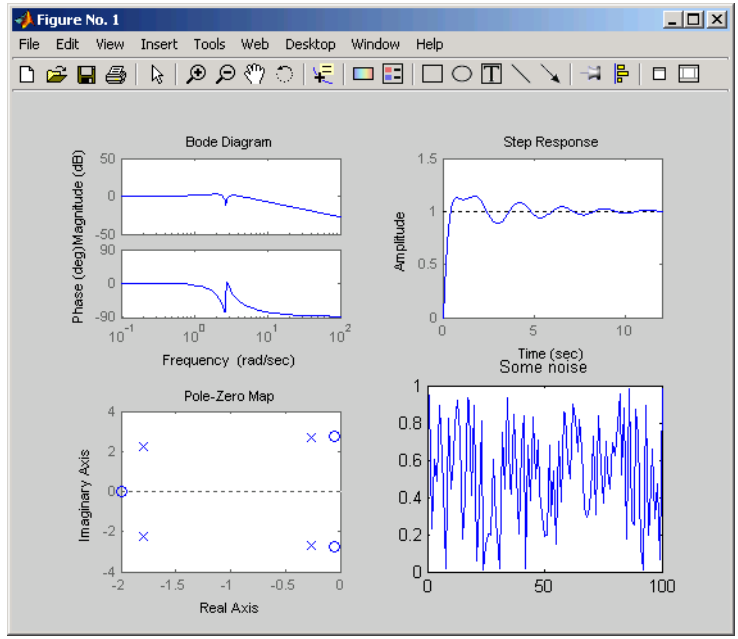
Example: Custom Plots

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

For information about `plot`, `subplot`, `hold`, and other options for plotting general data, see Basic Plots and Graphs in the MATLAB Function Reference. These documents are available in the MATLAB online help.

Note Each of the plots generated by response analysis functions in Example of Creating a Custom Plot on page 3-48 (`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.

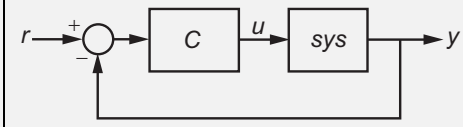
4

Designing Compensators

- "Quick Start for Tuning Compensator Parameters Using the SISO Design Tool" on page 4-15
- "Designing PID Controllers" on page 4-2
- "SISO Design Tool" on page 4-15
- "Bode Diagram Design" on page 4-54
- "Root Locus Design" on page 4-80
- "Nichols Plot Design" on page 4-102
- "Automated Tuning Design" on page 4-113
- "Multi-Loop Compensator Design" on page 4-119
- "Control Design Analysis of Multiple Models" on page 4-131
- "Functions for Compensator Design" on page 4-146

Designing PID Controllers

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

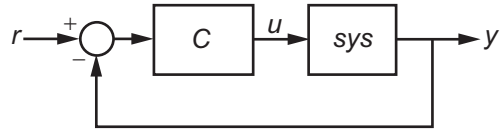
Tool	When to use
• PID Tuner GUI • Command-line PID tuning	Use to design a SISO PID controller in the feed-forward path of single-loop, unity-feedback control configuration. 
SISO Design Tool	Use to design a SISO PID controller in any other loop configuration.

Designing PID Controllers with the PID Tuner GUI

- "PID Tuner Overview" on page 4-2
- "PID Controller Type" on page 4-3
- "PID Controller Form" on page 4-5
- "Analyzing Your Design Using Response Plots" on page 4-5
- "Example: Designing a Controller for Fast Reference Tracking" on page 4-7

PID Tuner Overview

Use the PID Tuner GUI 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 `pidtool` command:

```
pidtool(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

```
pidtool(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 `pidtool` reference page.

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 `pidtool`.
- Provide a baseline controller `Cbase` to the launch command `pidtool`. The PID Tuner designs a controller of the same type as `Cbase`.
- Use the **Type** menu to change controller types after launching the PID Tuner.

`type` input to `pidtool`	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	$\frac{K_i}{s}$	$K_i \frac{T_s}{z-1}$
`'pi'`	`PI`	Proportional and integral	$K_p + \frac{K_i}{s}$	$K_p + K_i \frac{T_s}{z-1}$
`'pd'`	`PD`	Proportional and derivative	$K_p + K_d s$	$K_p + K_d \frac{z-1}{T_s}$
`'pdf'`	`PDF`	Proportional and derivative with first-order filter on derivative term	$K_p + \frac{K_d s}{T_f s + 1}$	$K_p + K_d \frac{1}{T_f + \frac{T_s}{z-1}}$
`'pid'`	`PID`	Proportional, integral, and derivative	$K_p + \frac{K_i}{s} + K_d s$	$K_p + K_i \frac{T_s}{z-1} + K_d \frac{z-1}{T_s}$
`'pidf'`	`PIDF`	Proportional, integral, and derivative with first-order filter on derivative term	$K_p + \frac{K_i}{s} + \frac{K_d s}{T_f s + 1}$	$K_p + K_i \frac{T_s}{z-1} + K_d \frac{1}{T_f + \frac{T_s}{z-1}}$

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 `pidtool`. The PID Tuner designs a controller that has the same discrete integrator formulas as `Cbase`.
- After launching the PID Tuner, click the button to open the **PID Tuner Settings** 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

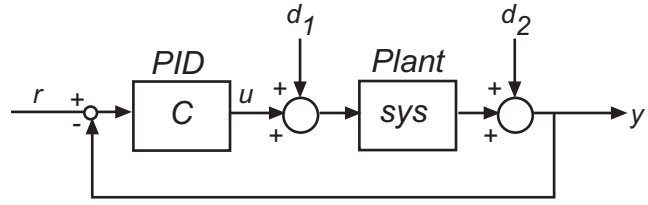
When you use the `type` input to launch the PID Tuner, the PID Tuner designs a controller in parallel form. To design a controller in standard form, use one of the following methods:

- Provide a standard-form baseline controller `Cbase` to the launch command `pidtool`. 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.

Analyzing Your Design Using Response Plots

Select response plots from the **Response** menu to analyze the controller's performance. The PID Tuner computes the system responses assuming the following architecture:



The following table summarizes the available response plots.

Response	Plotted System	Description
`Reference tracking`	$\frac{C*sys}{1+C*sys}$ (from r to y)	Shows the closed-loop system response to a change in setpoint. Use when your design specifications include setpoint tracking.
`Controller effort`	$\frac{C}{1+C*sys}$ (from r to u)	Shows the closed-loop controller output response to a change in setpoint. Use when your design is limited by practical constraints, such as controller saturation.
`Input disturbance Rejection`	$\frac{sys}{1+C*sys}$ (from d_1 to y)	Shows the closed-loop system response to load disturbance (a disturbance at the plant input). Use when your design specifications include input disturbance rejection.
`Output disturbance Rejection`	$\frac{1}{1+C*sys}$ (from d_2 to y)	Shows the closed-loop system response to measurement noise.
`Open-loop`	*C*sys*	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.
`Plant`	*sys*	Shows the plant response. Use to examine plant dynamics.

Choose time-domain (`Step`) or frequency-domain (`Bode`) responses from the **Plot** menu.

If you provide `Cbase`, check **Show baseline** to display the response of the baseline controller.

Example: Designing a Controller for Fast Reference Tracking

In this example, you use the PID Tuner to design a controller for the plant:

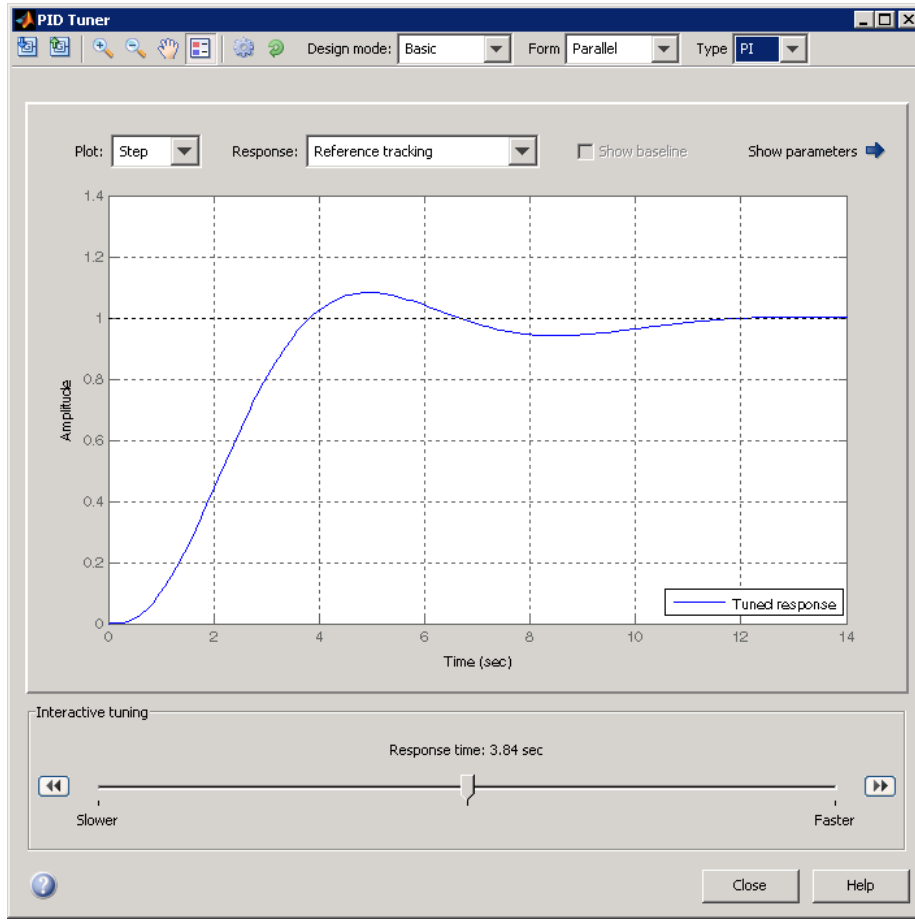
$$sys = \frac{1}{(s+1)^3}.$$

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. Launch the PID Tuner to design a PI controller for a first pass design.

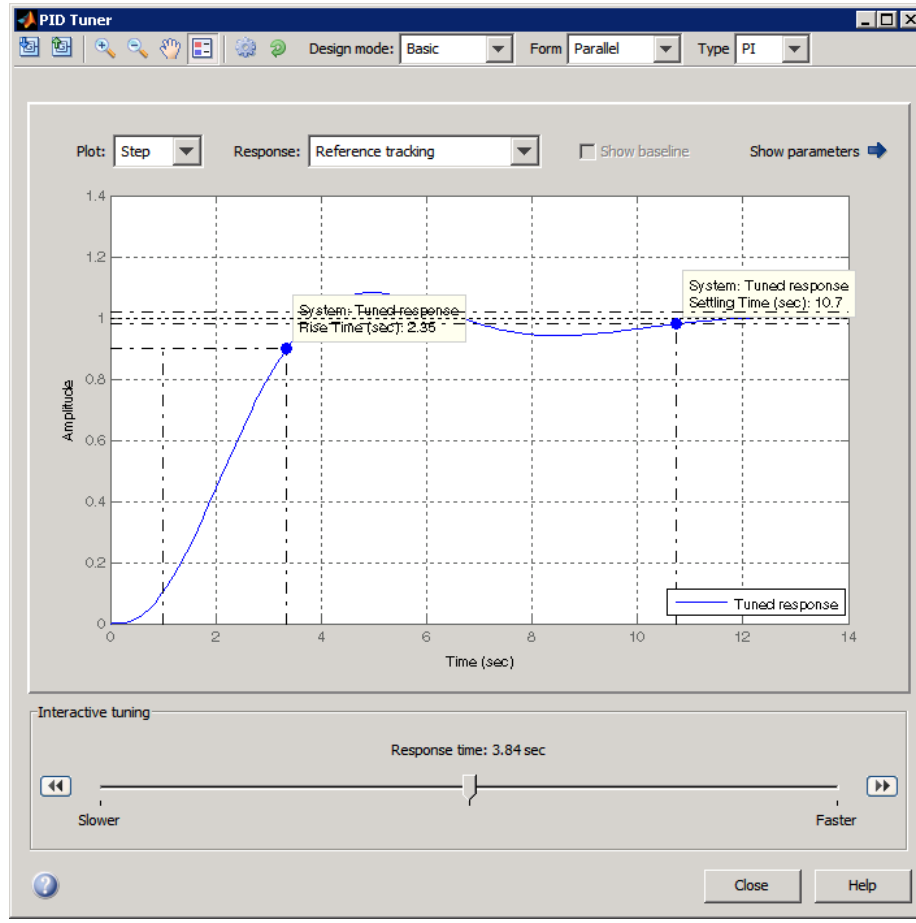
   ```
   sys = zpk([],[-1 -1 -1],1);  % create the plant model
   pidtool(sys,'pi')
   ```

   Upon launch, the PID Tuner displays the closed-loop step response of the system with a PI controller designed for a balance between performance (response time) and robustness (stability margins).



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.



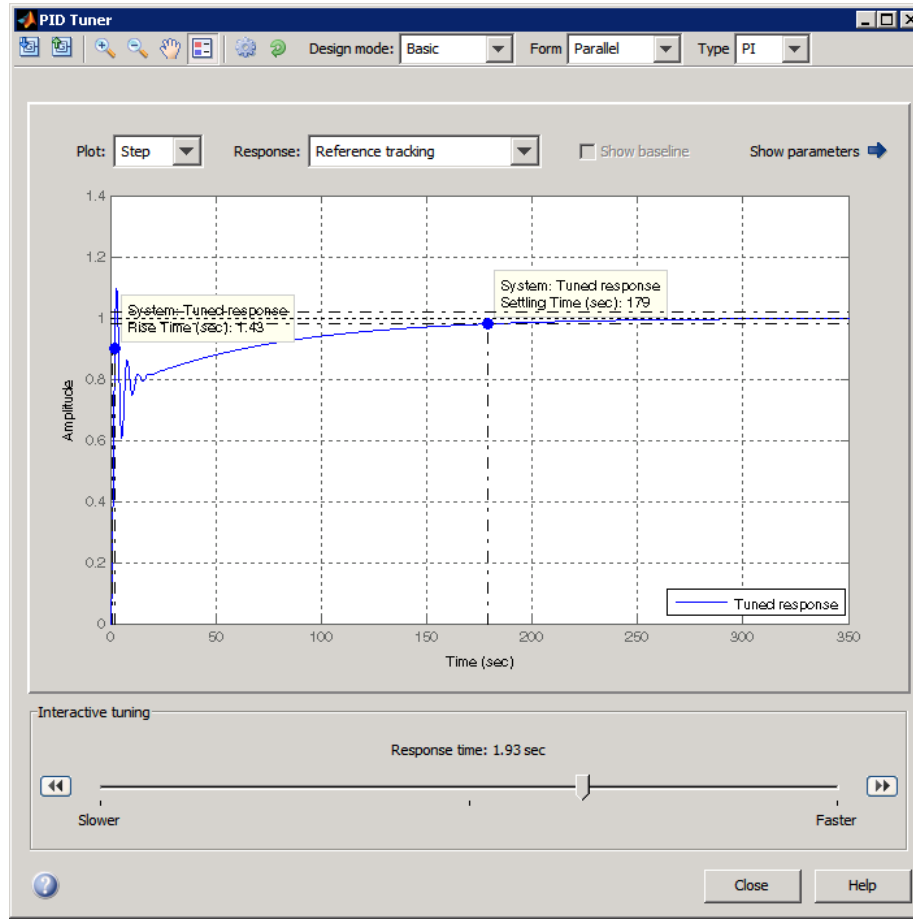
The PID Tuner default PI controller provides a rise time of 2.35 s and settling time of 10.7 s. Both results are slower than the design requirements.

Note To display the performance metrics in a table instead of in tool-tips on the plot, click the **Show parameters** button . This action expands the GUI to show performance and robustness metrics. It also displays the controller gains.

3 Use the **Response time** slider to try to improve the loop performance.

Slide the slider to the right to make the loop faster. The response plot automatically updates with the new design.

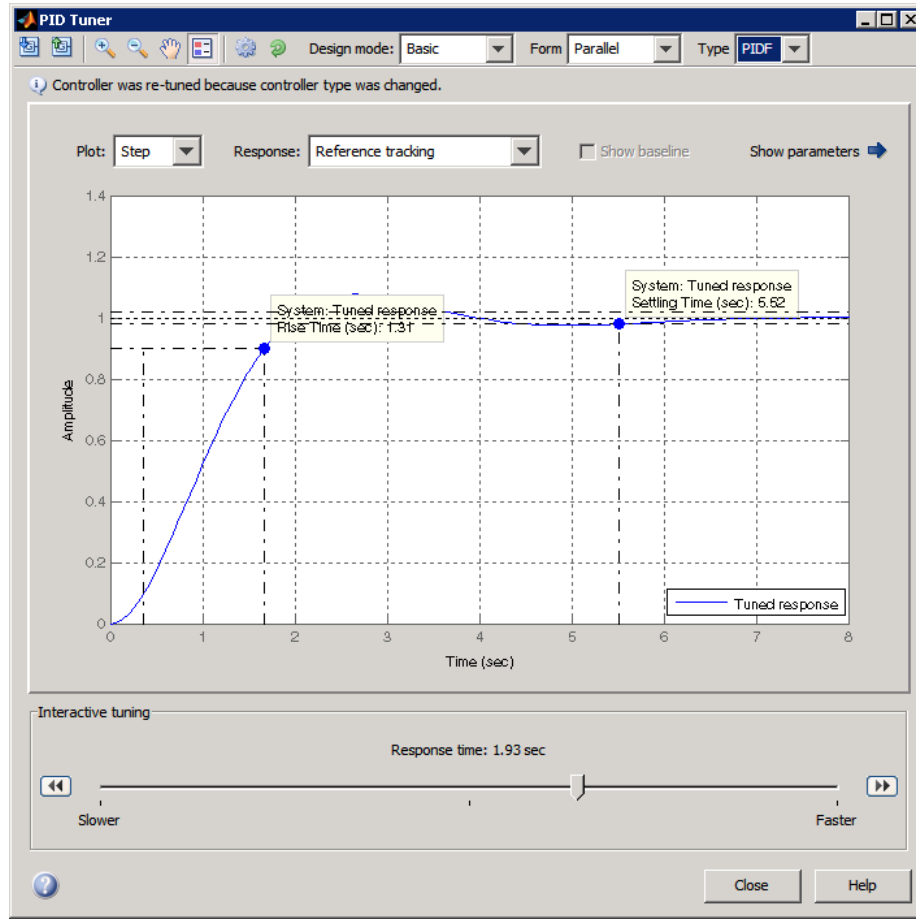
Moving the slider far enough to meet the rise time requirement of less than 1.5 s, however, results in unacceptably long settling time. There is also more oscillation. To achieve the faster response speed, the algorithm must sacrifice stability.



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.

Select `PIDF` from the **Type** menu. The PID Tuner designs a new PIDF controller. (See "PID Controller Type" on page 4-3 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 the button.

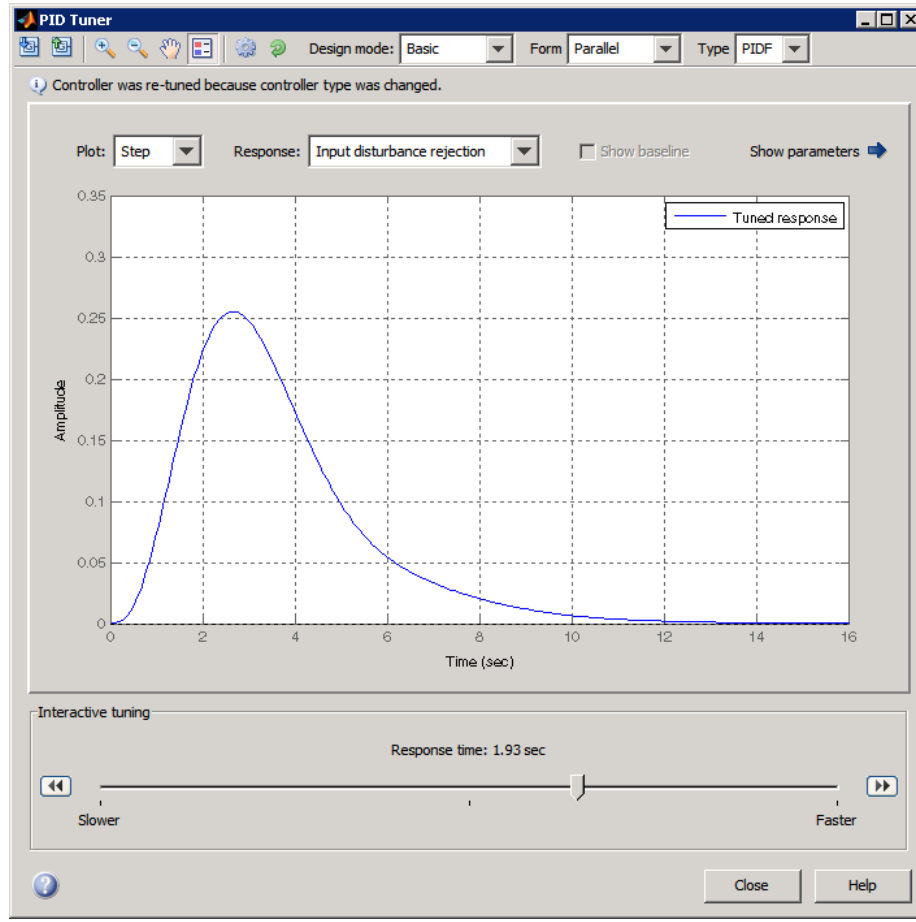
Note To adjust the closed-loop bandwidth and phase margin separately, select `Extended` from the **Design mode** menu . The bandwidth is inversely proportional to the response time.

---

5 Analyze other system responses, if appropriate.

To analyze other system responses, use the **Response** menu to choose a response plot and the **Plot** menu to choose time-domain (`Step`) or frequency-domain (`Bode`) responses.

For example, to observe the closed-loop system response to disturbance at the plant input, select `Input disturbance rejection` from the **Response** menu.



See “Analyzing Your Design Using Response Plots” on page 4-5 for more information about available response plots.

6 Export your controller design to the MATLAB workspace.

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

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

For more examples of controller design using the PID Tuner, see the Control System Toolbox demo, Designing PID for Disturbance Rejection with PID Tuner

Designing PID Controllers at the Command Line

To design PID Controllers at the command line, use the `pidtune` command. For more information and examples, see:

- `pidtune` reference page
- Control System Toolbox demo: Designing Cascade Control System with PI Controllers

SISO Design Tool

In this section...

"Quick Start for Tuning Compensator Parameters Using the SISO Design Tool" on page 4-15

"Components of the SISO Tool" on page 4-25

"Design Options in the SISO Tool" on page 4-25

"Opening the SISO Design Tool" on page 4-26

"Using the SISO Design Task Node in the Control and Estimation Tools Manager" on page 4-28

"Importing Models into the SISO Design Tool" on page 4-29

"Feedback Structure" on page 4-33

"Analysis Plots for Loop Responses" on page 4-35

"Using the Graphical Tuning Window" on page 4-41

"Using the Graphical Tuning Window" on page 4-41

"Exporting the Compensator and Models" on page 4-48

"Storing and Retrieving Intermediate Designs" on page 4-50

Quick Start for Tuning Compensator Parameters Using the SISO Design Tool

In this Quick Start, 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

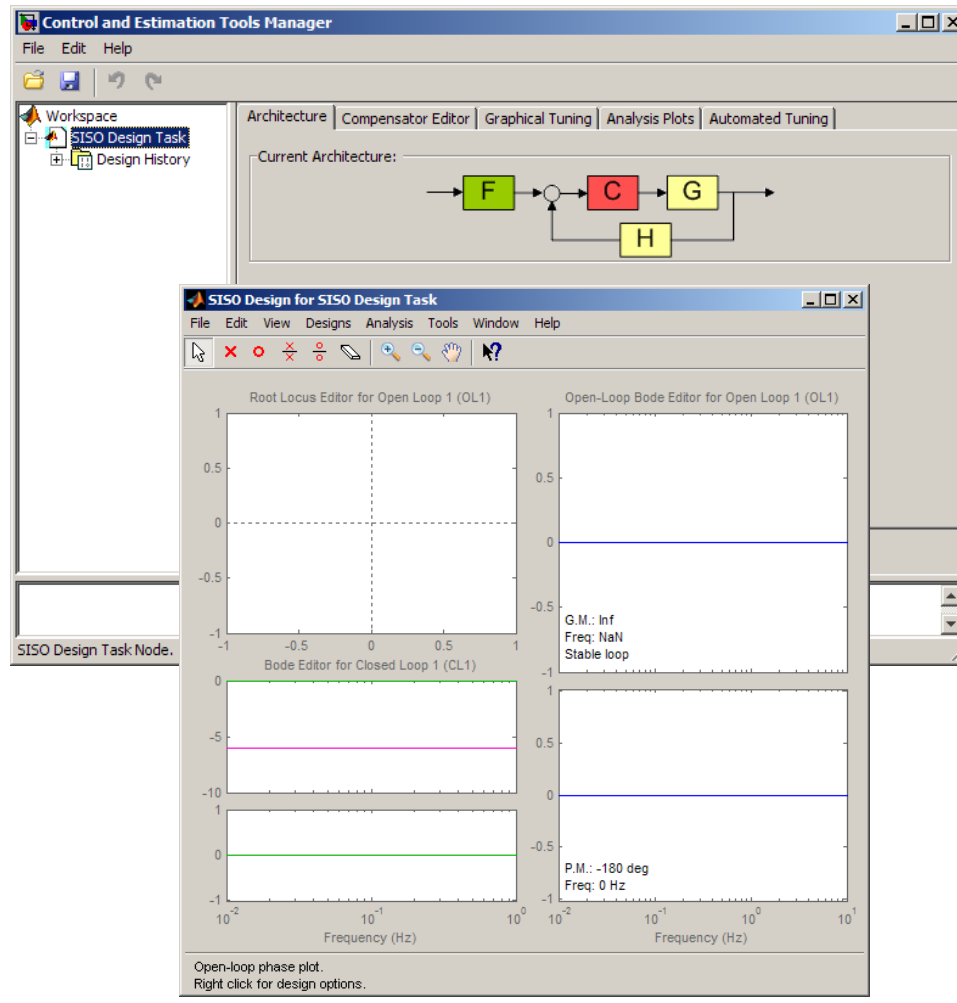
  For information on how to create a model, see Chapter 2, "Building Models".
- Design requirements for your system

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.

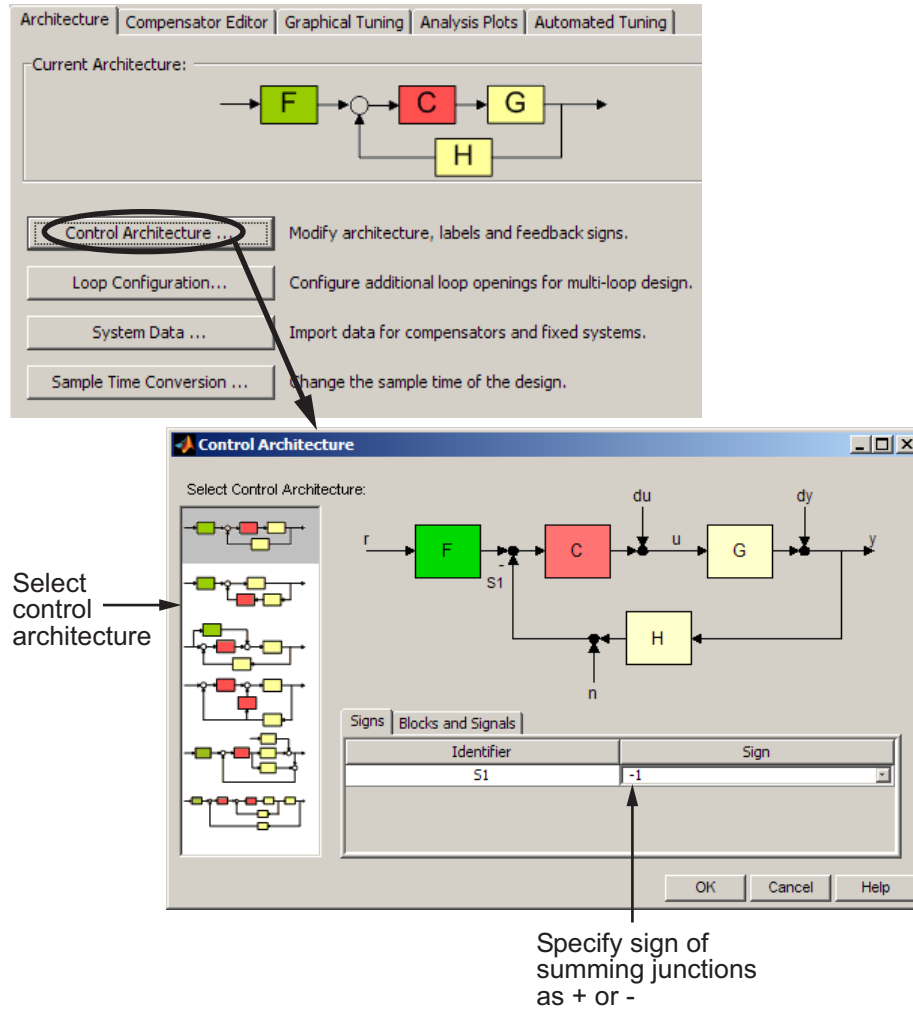


2 Define the control architecture for your system.

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.

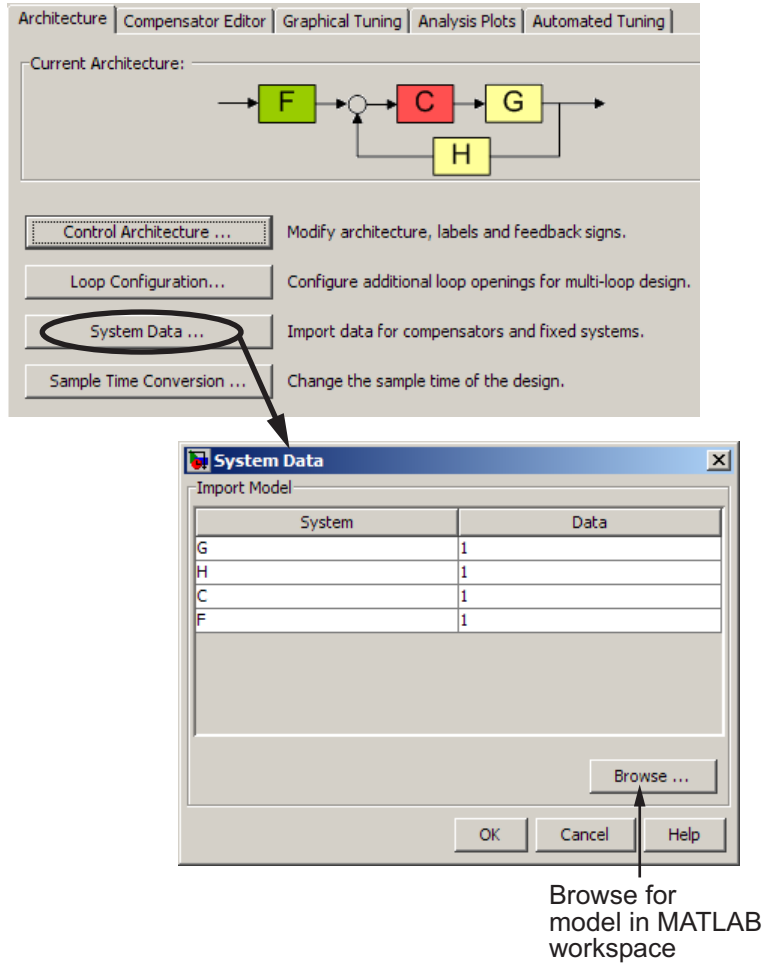
c Specify the sign of summing junctions as + or -.



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.

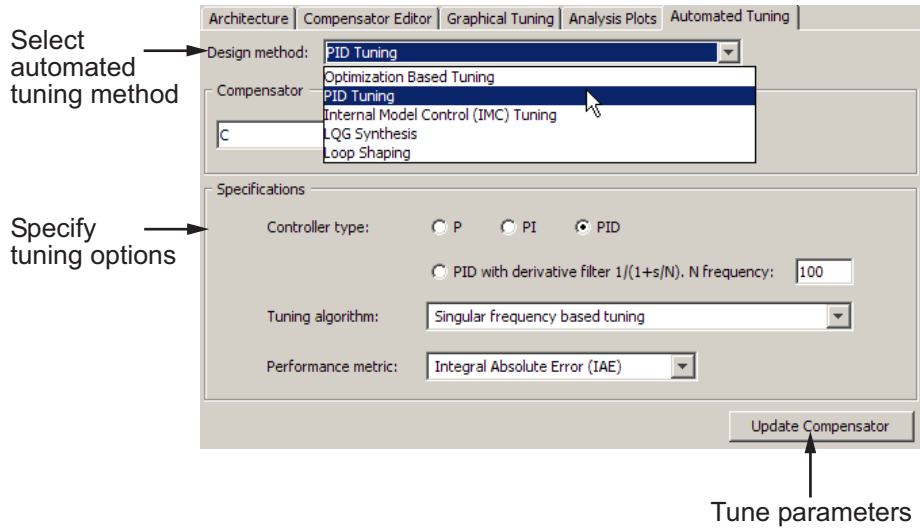


4 Design a compensator using automated tuning, for example PID Tuning.

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.

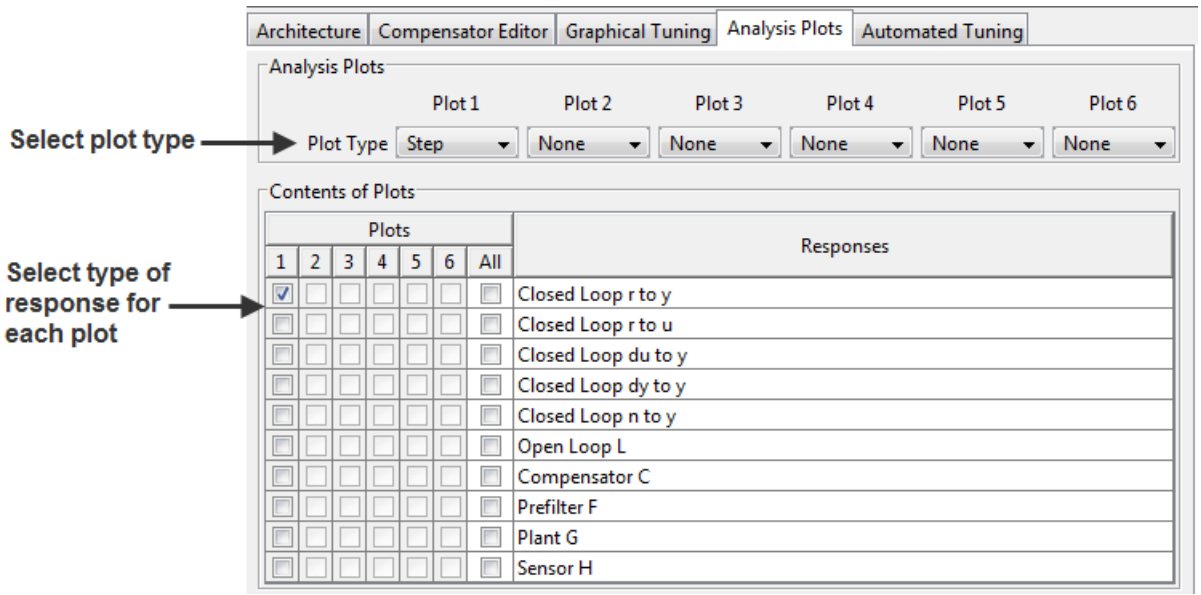


5 Evaluate the system's closed-loop performance.

a Plot the system response.

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

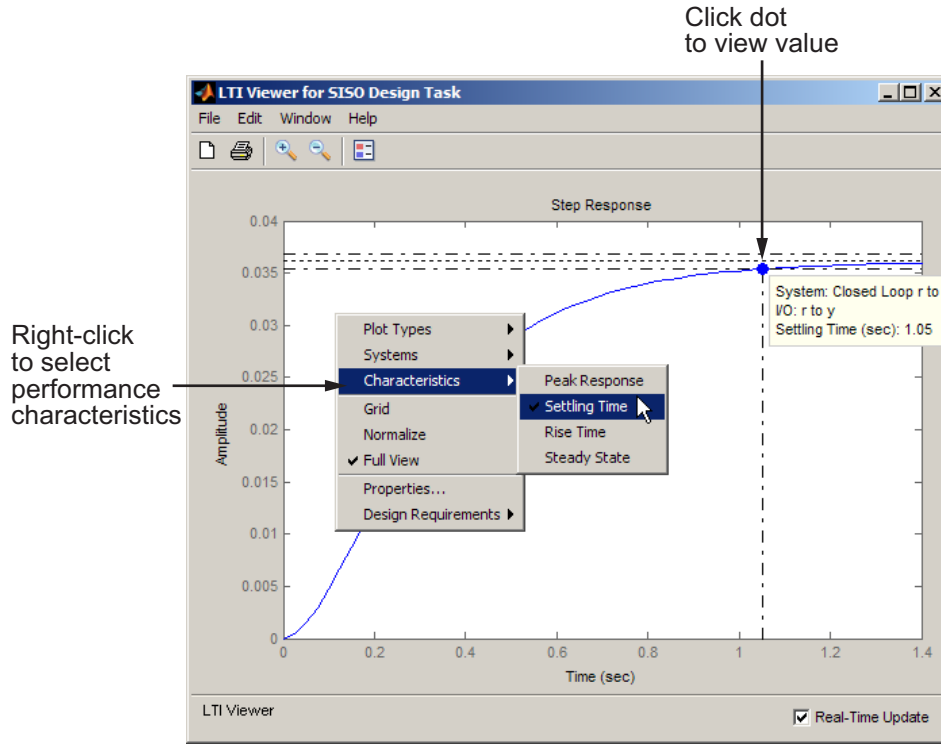
ii Select the type of response for each plot.



b Display specific performance characteristic for your system. Compare values to design requirements.

iii Right-click to select performance characteristics.

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

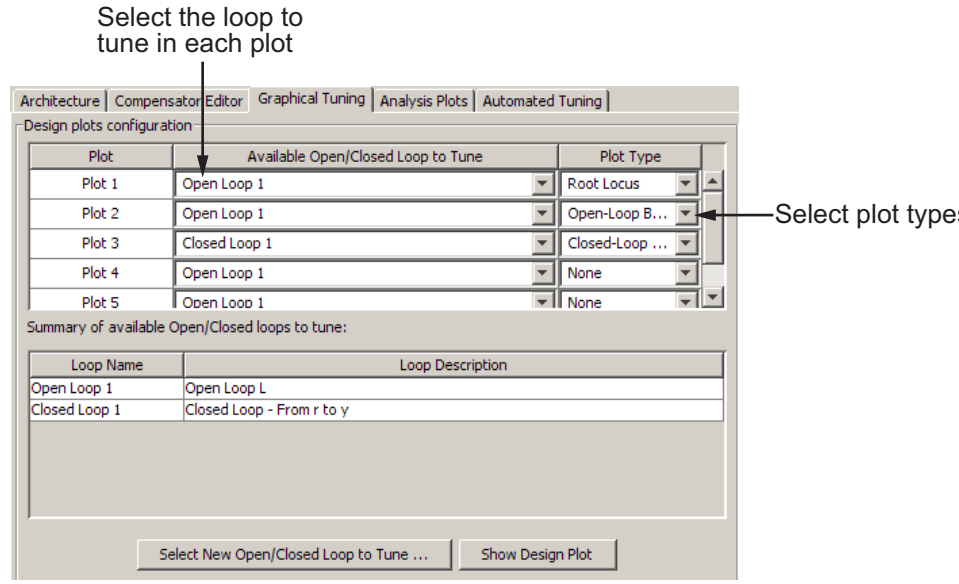


6 Refine compensator design using graphical tuning.

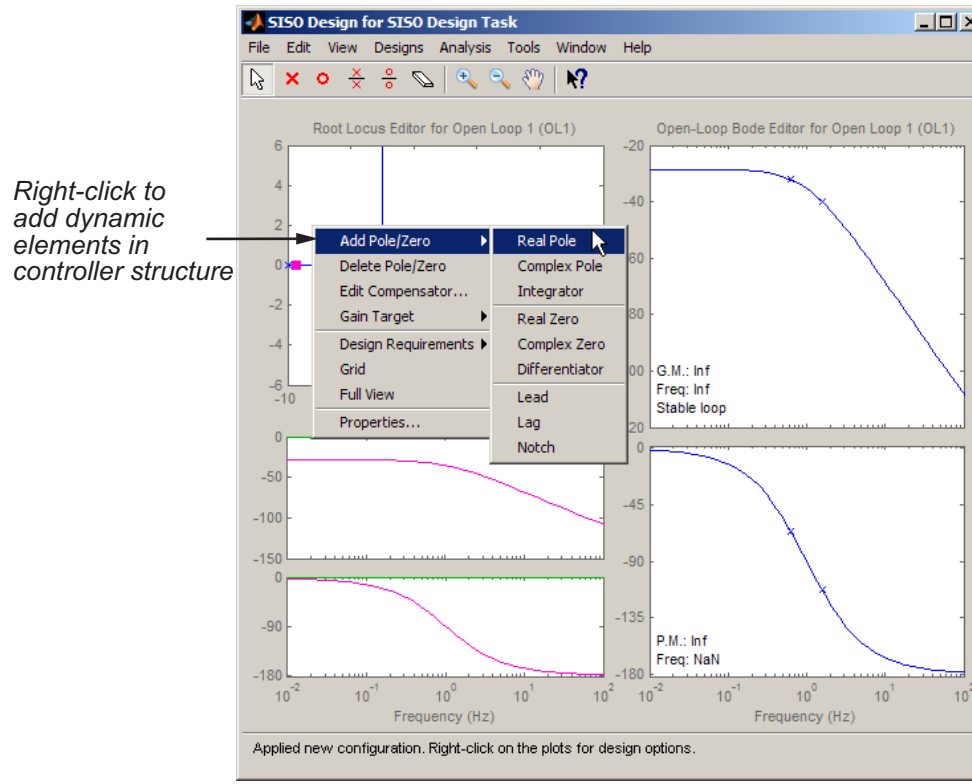
a Create plots for graphical tuning.

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

ii Select plot types.

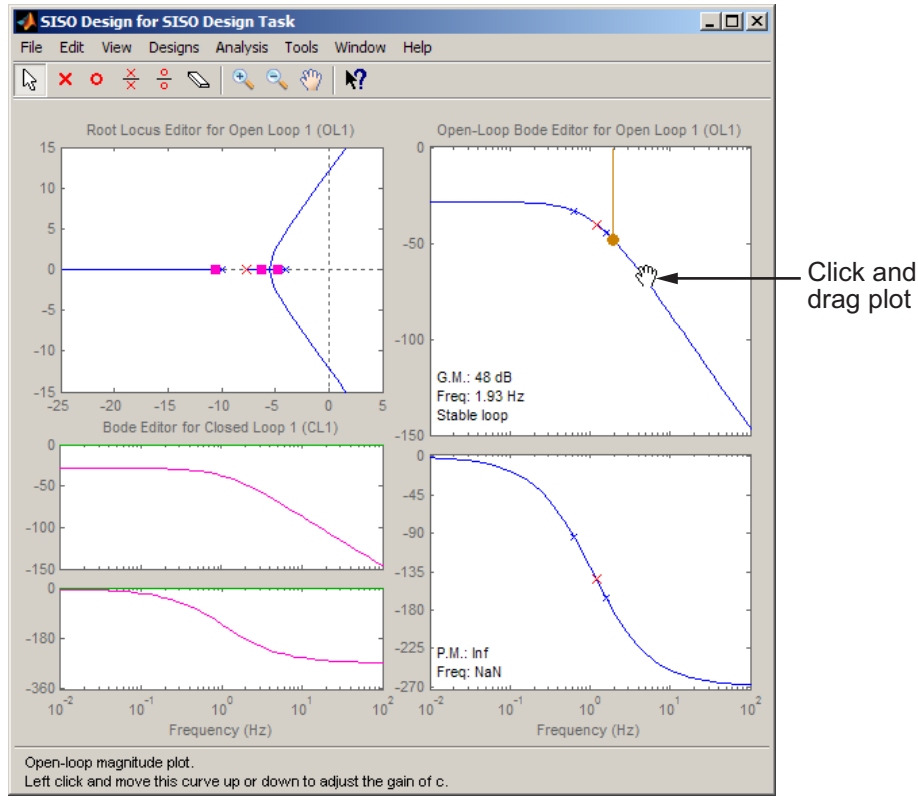


b In a design plot, modify the compensator by adding poles, zeros, lead, and lag, for example. To do this, right-click to add dynamic elements in the controller structure.



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.



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 created in Chapter 2, “Building Models”

If you have not built the DC motor model, type

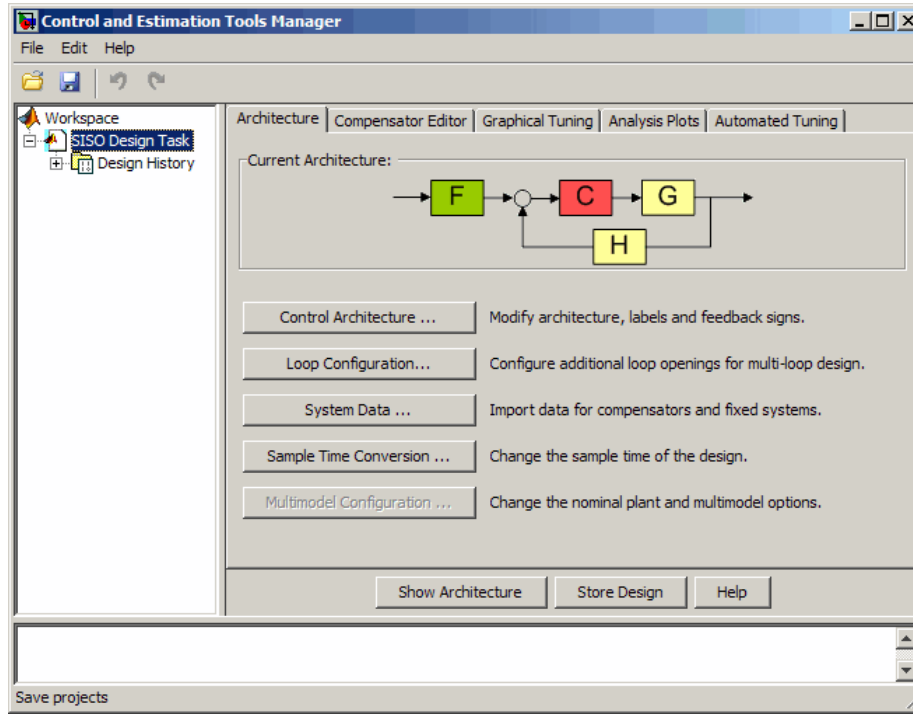
```
load ltiexamples
```

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

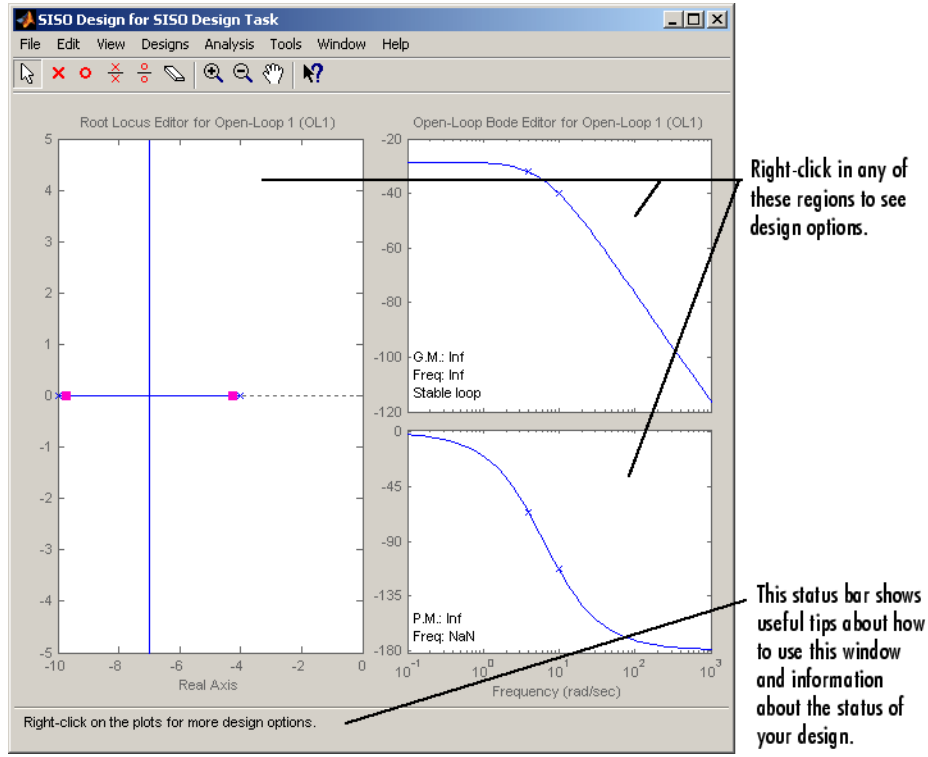
```
sisotool(sys_dc)
```

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)



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:

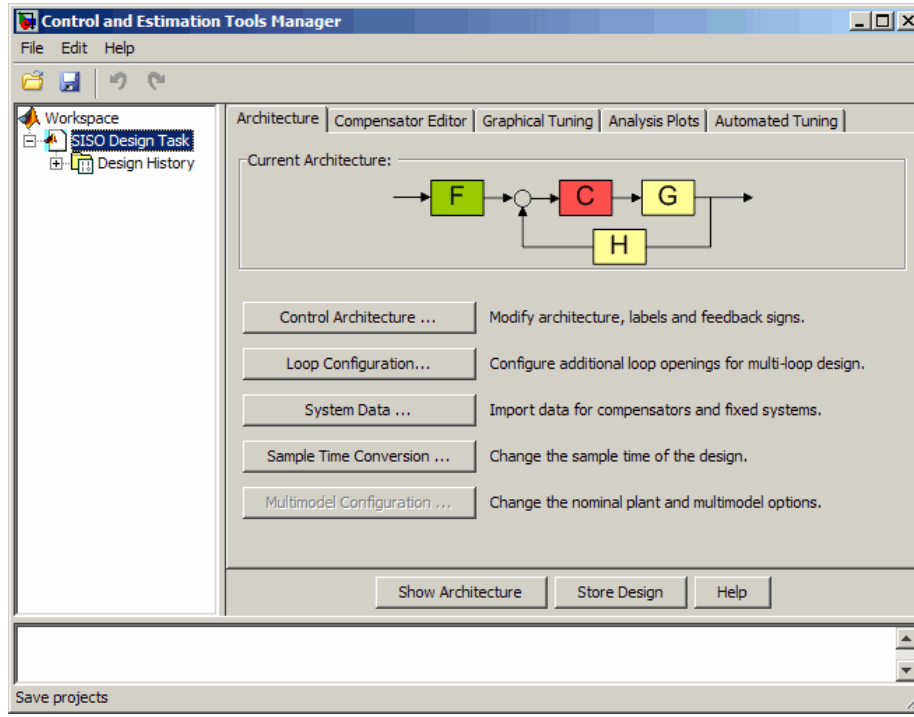
- **Architecture**:
  - Change the feedback structure and label signals and blocks.
  - Configure loops for multi-loop design by opening signals to remove the effects of other feedback loops.
  - Import LTI models or row or column arrays of LTI models into your system.

  - 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).

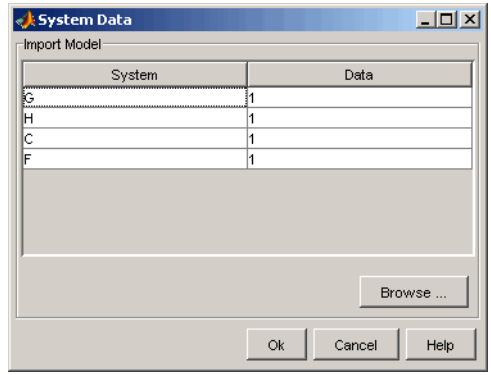
Importing Models into the SISO Design Tool

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.

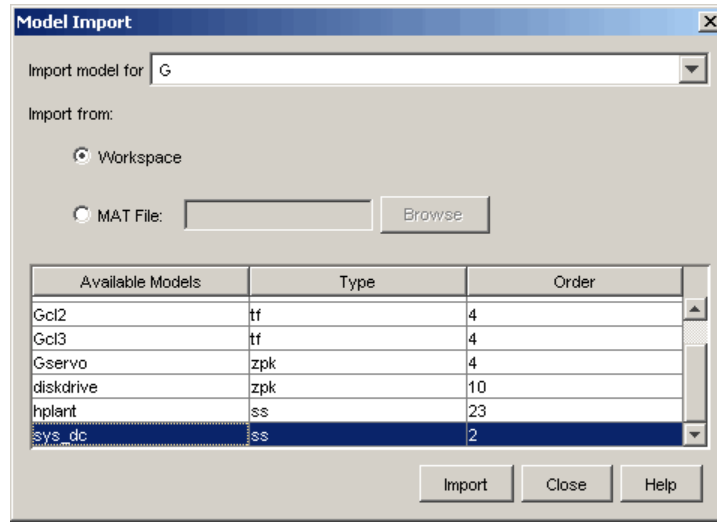
- **H** is the sensor modeled as an LTI model or multiple sensors modeled as a row or column array of LTI models. If **G** is also an array, their sizes must match.
- **C** is a controller and is an LTI model.
- **F** is a prefilter and is an LTI model.

The default values of **G**, **H**, **C**, and **F** are 1. See "Feedback Structure" on page 4-33 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 4-131.

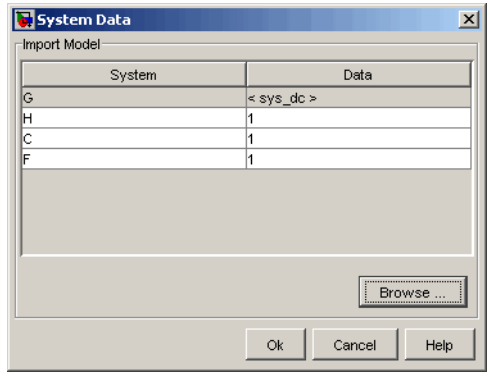
To import the DC motor model:

1. Select **G** and click **Browse**. The Model Import dialog box opens, as shown in the following figure.



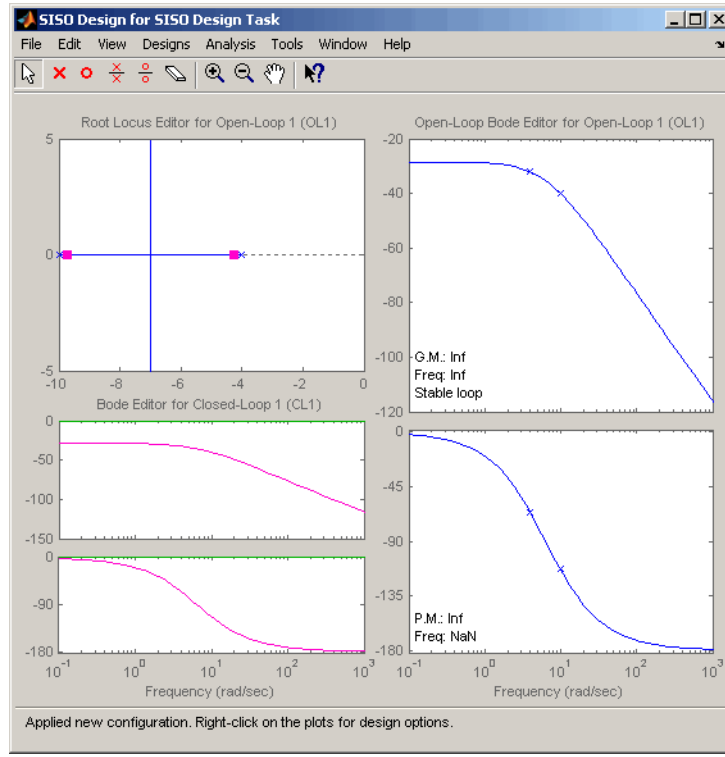
2 Select `sys_dc` from the **Available Models** list. Click **Import**, and then click **Close**.

You can now see `sys_dc` in the System Data dialog box.



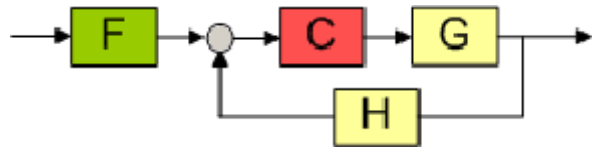
3 Click **OK**.

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

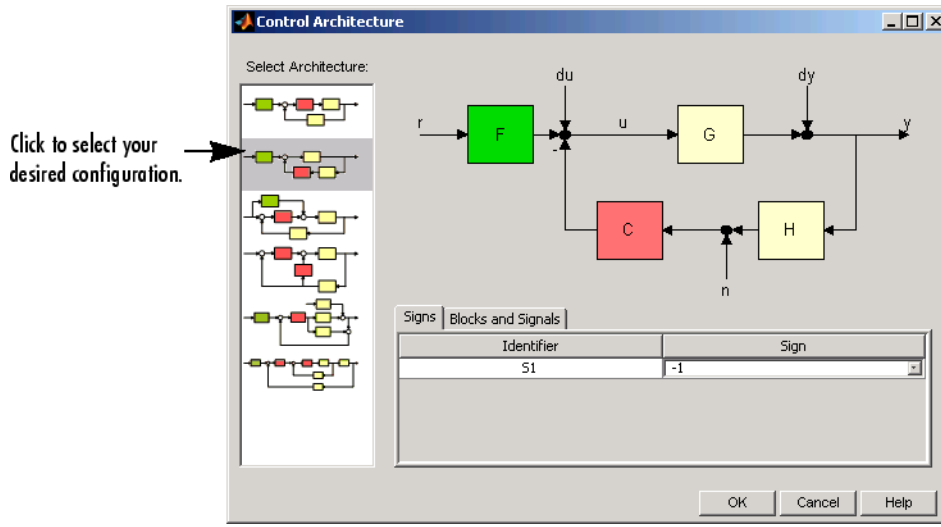
- **F** represents the prefilter
- **C** represents the compensator

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`.

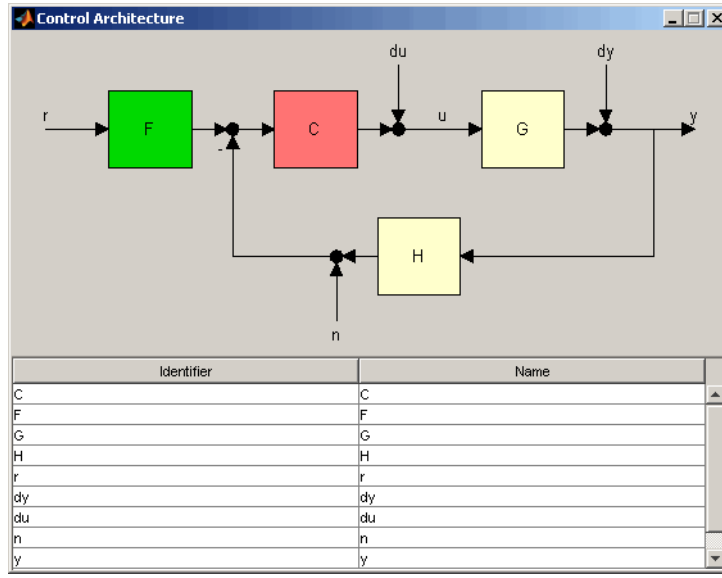
Alternative Feedback Structures

In the **Architecture** page, click **Control Architecture** to open the Control Architecture dialog box.



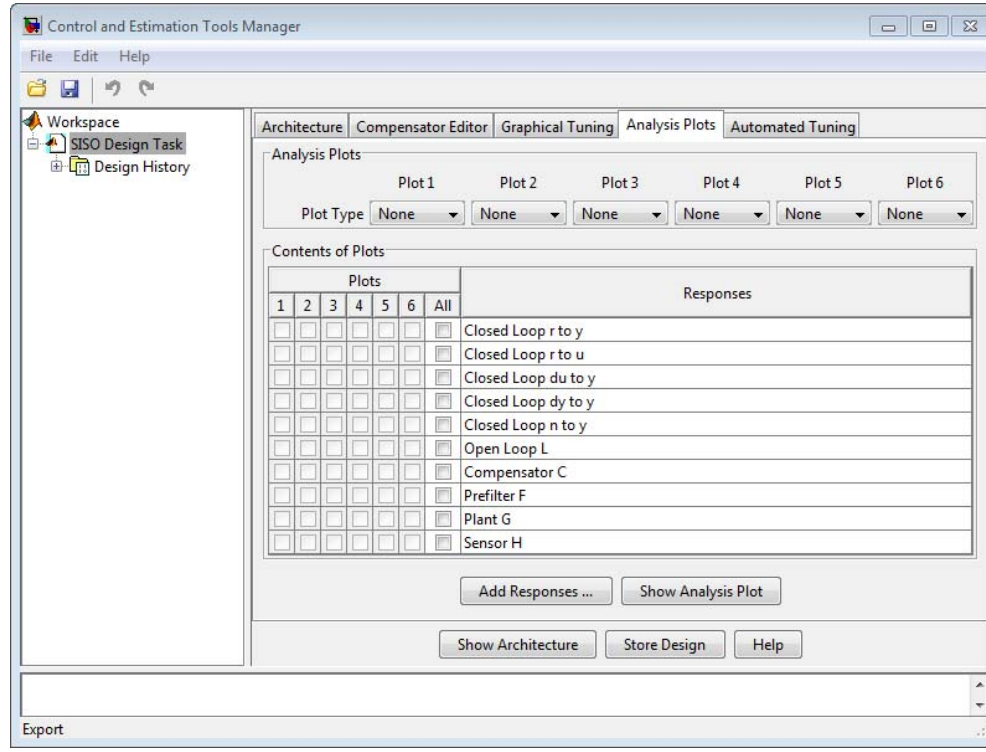
You can use the **Signs** and **Blocks and Signals** panes to change the sign of the feedback signal into a summing junction and rename blocks and signals in the diagram respectively. See “Block Diagram Structure Modifications” for more details.

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.

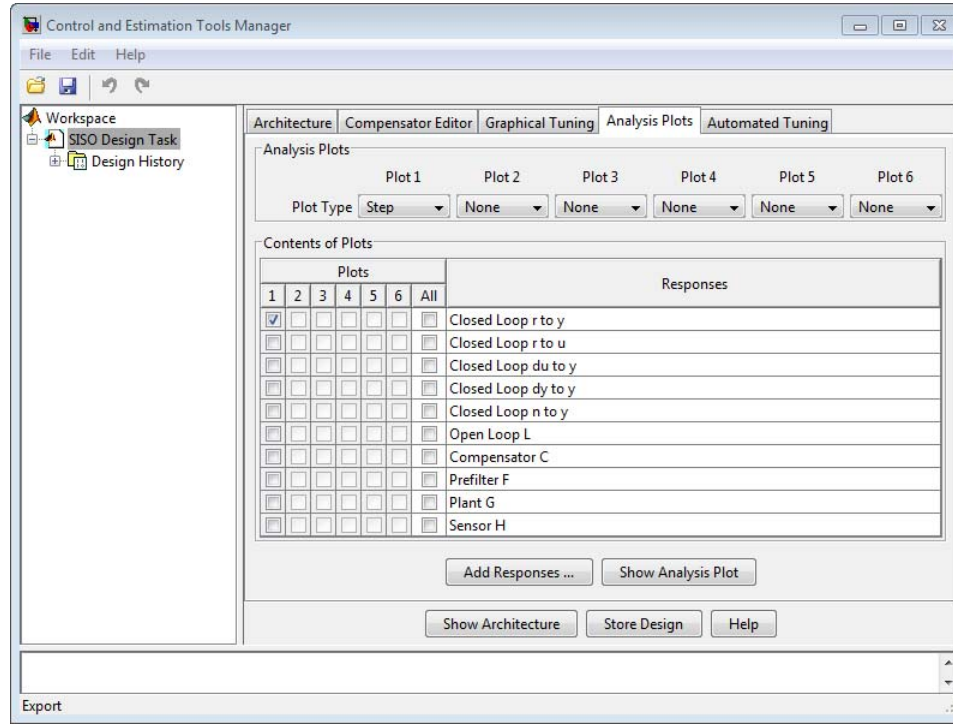


Analysis Plots for Loop Responses

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.



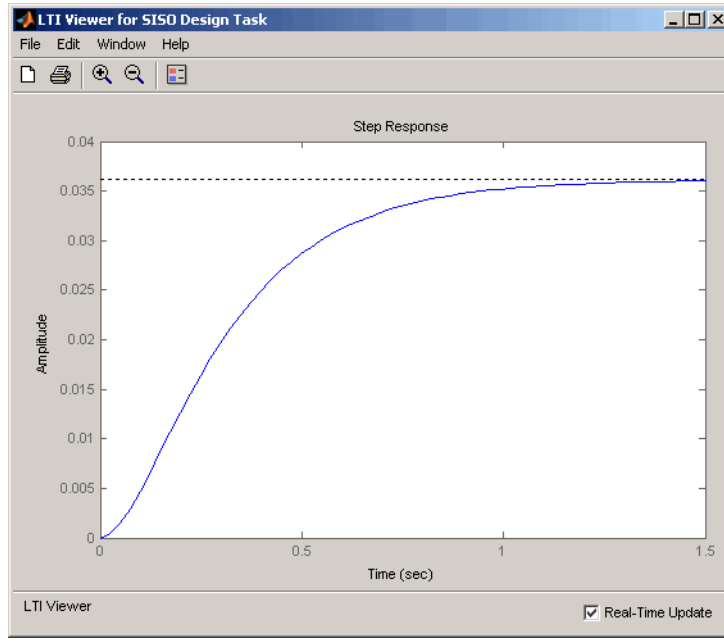
Select the plot types for each plot in the **Analysis Plots** area. Then select the plots to appear in the **Plots** list in the **Contents of Plots** table, as shown in the following figure.



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.

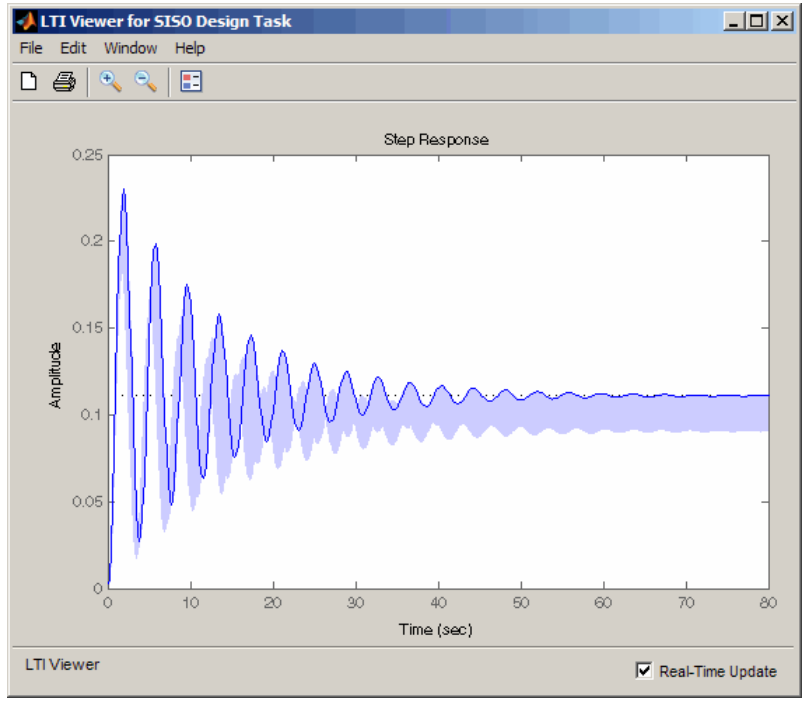


LTI Viewer Showing the Step Response for the DC Motor

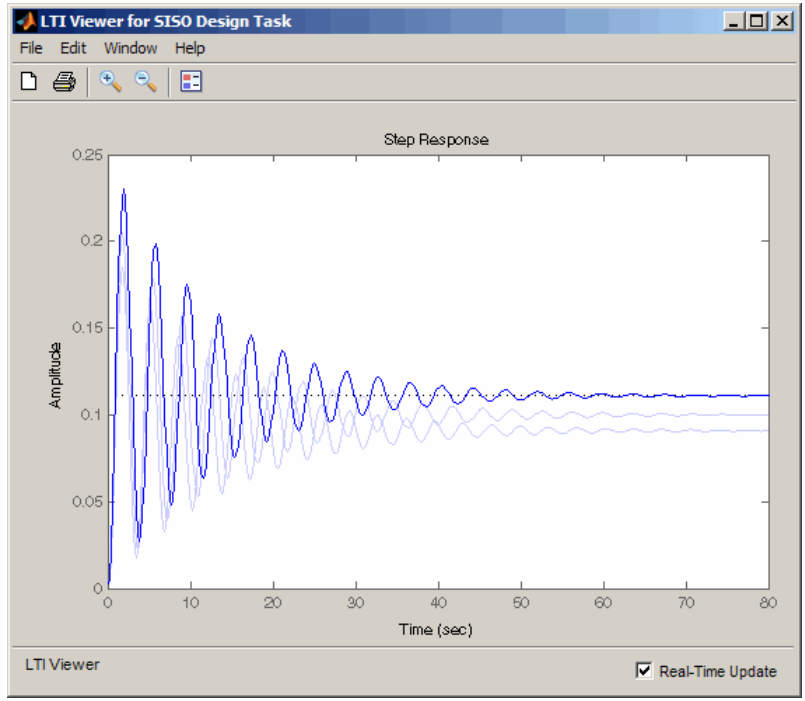
The settling time of the DC motor is about 1.5 seconds, which is too slow for many applications. Also, the plot shows a large steady-state error. You can use Bode diagram techniques to improve the response time and steady-state error of the DC motor step response, as described in "Bode Diagram Design" on page 4-54.

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 4-131.

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:

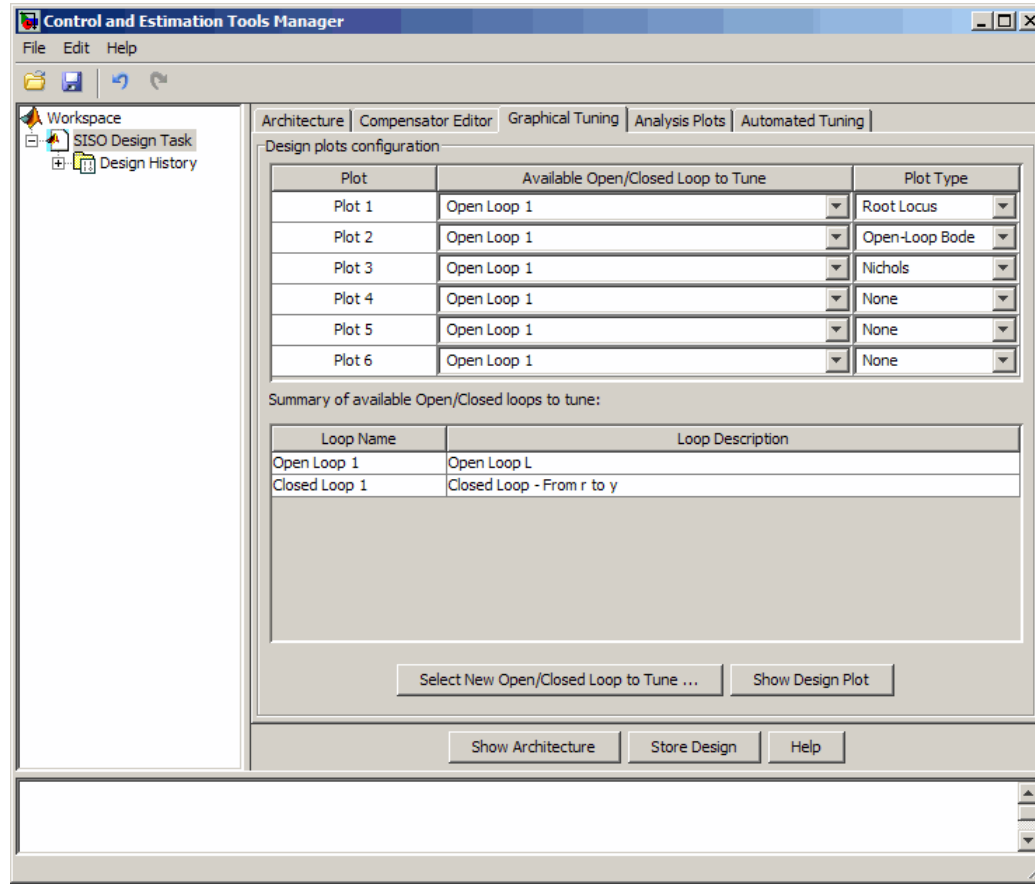
- A large number of responses
- Responses of a large number of models

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**.



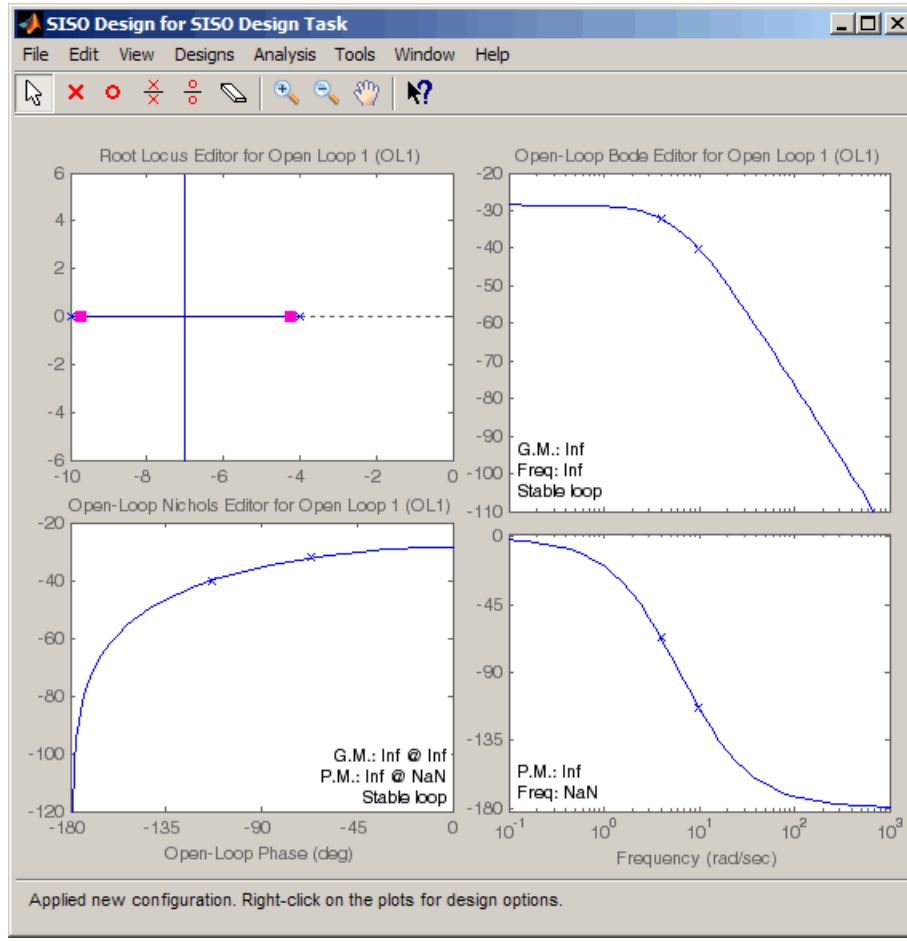
This topic describes some of the methods you can use to navigate in the Graphical Tuning window and manipulate the window's appearance.

Graphical Tuning Window Display

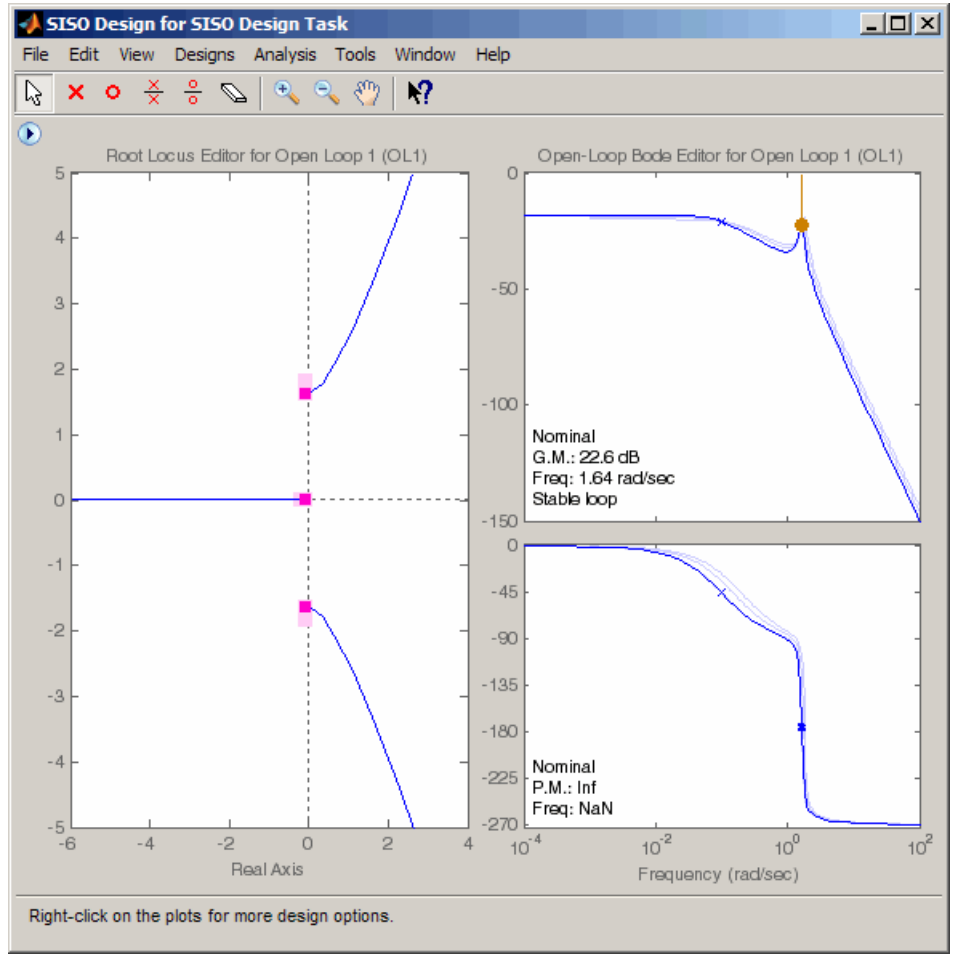
The Graphical Tuning window shows:

- 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.

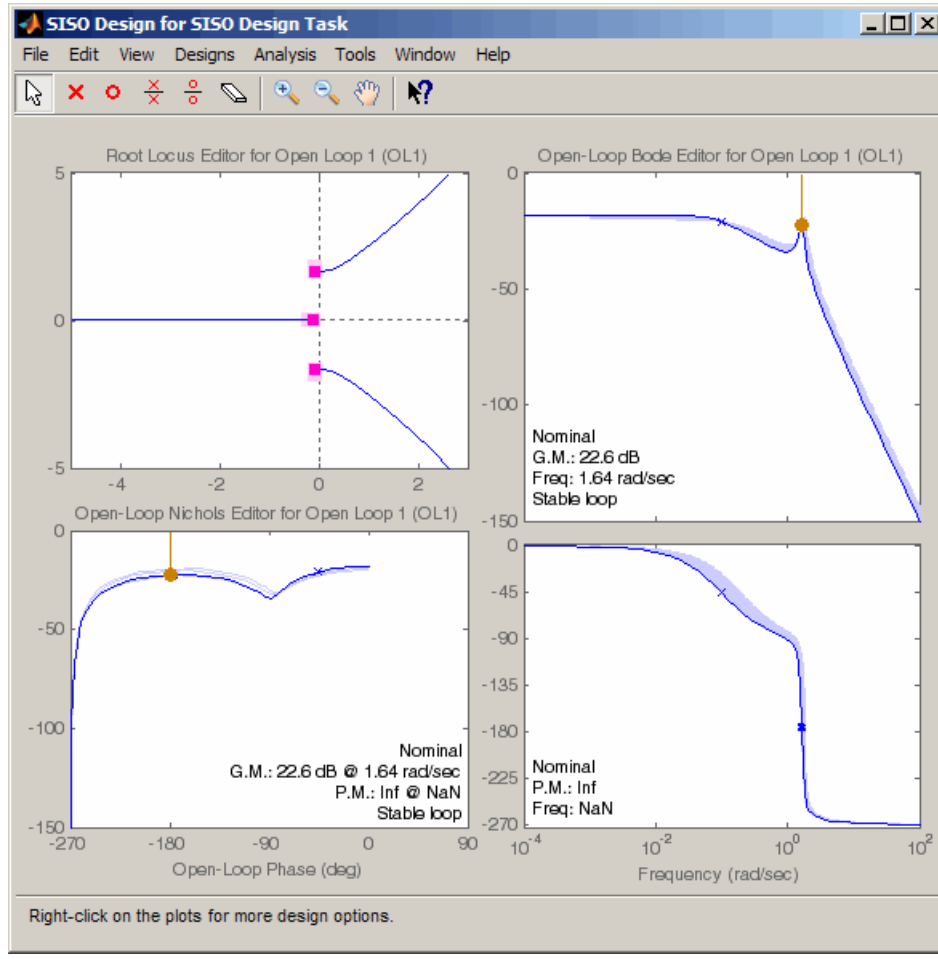


For a row or column arrays of LTI models, the plots show the individual responses and poles and zeros of all models in the array, as shown in the following figure.



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.



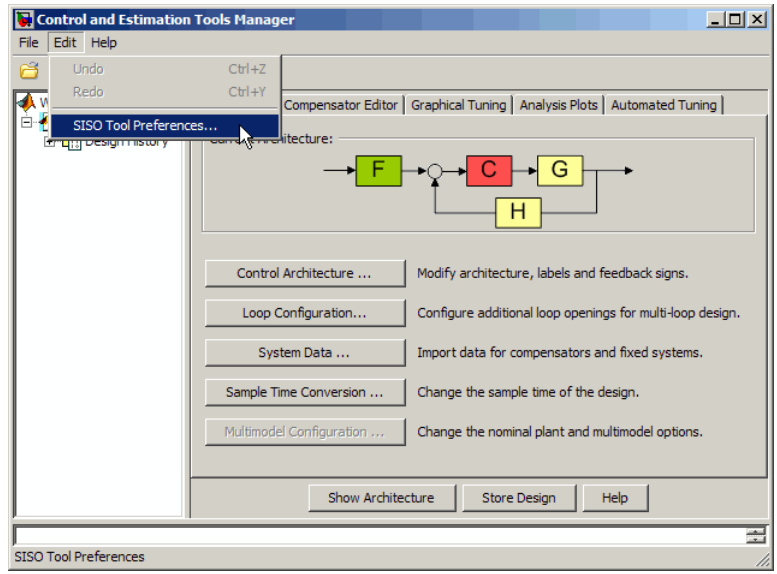
As you are designing a controller for the nominal plant, you can simultaneously analyze the controller's effect on the remaining models in the array. The software computes the response for each model and plots them. This computation is expensive and results in a slower refresh of the plots for:

- Large number of responses
- Responses for a large number of models
- Dense frequency grid

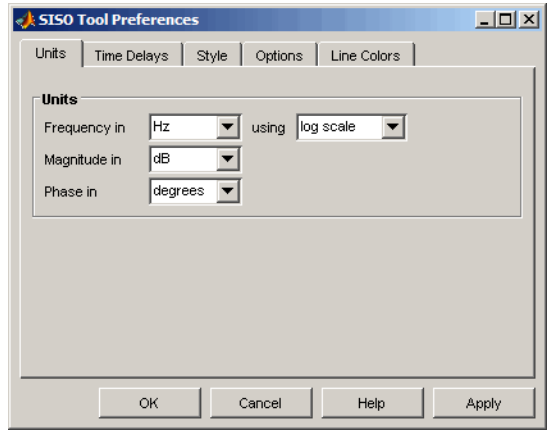
To speed up the plot updates, change the display to plot the envelope instead of individual responses. For more information, see “Control Design Analysis of Multiple Models” on page 4-131.

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.

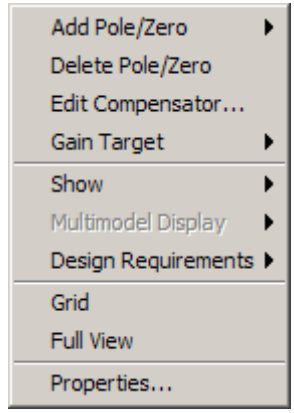


Use the options on the **Units** page to make the change. This unit change persists for the entire session.

For more information about property and preference settings, see "Customization".

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.

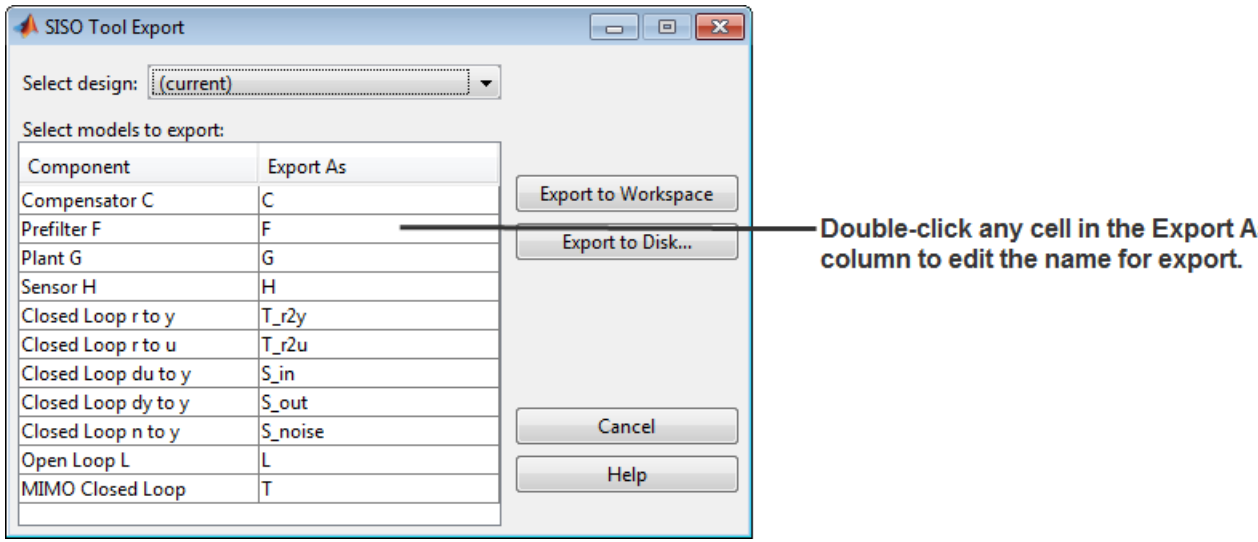
The right-click menus contain numerous features. The DC motor example uses many of the available features; for a complete discussion of the right-click menus, see the help for the SISO Design Tool."

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
```

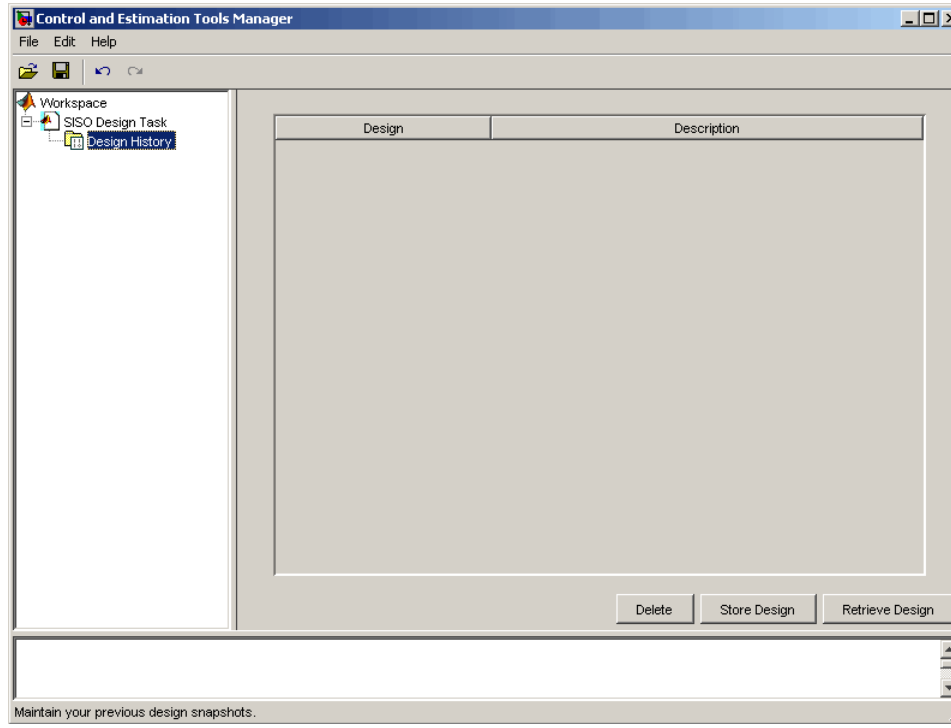
Clicking **Export to Disk** opens the Export to Disk dialog box.

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.

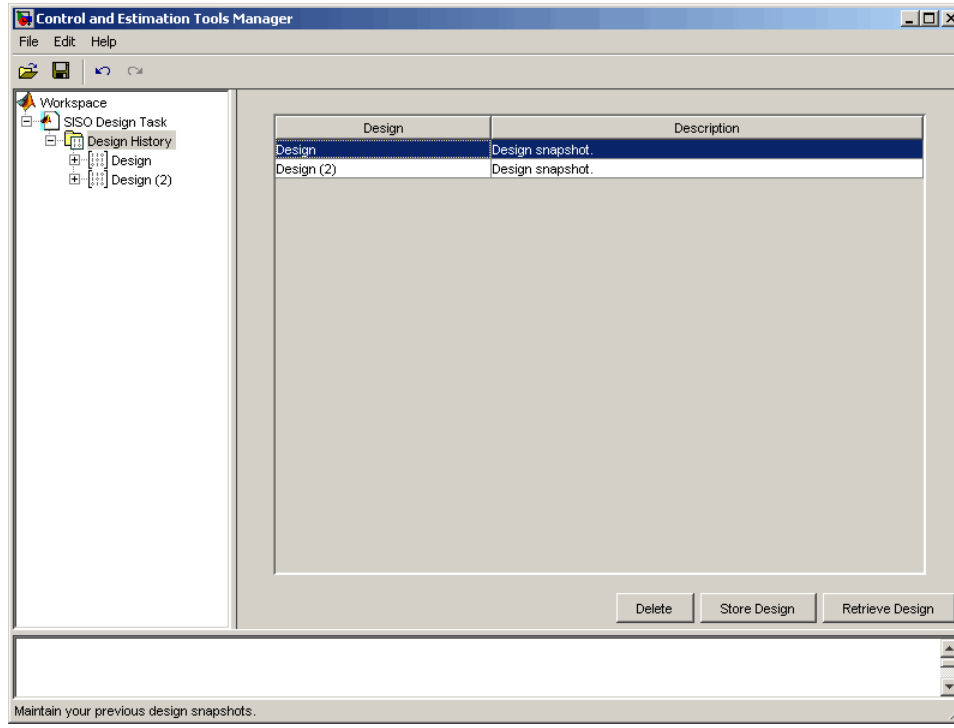
Alternatively, you can select **Store/Retrieve** from the **Designs** menu in the Graphical Tuning window. Using either method, the following **Design History** page opens.



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**.

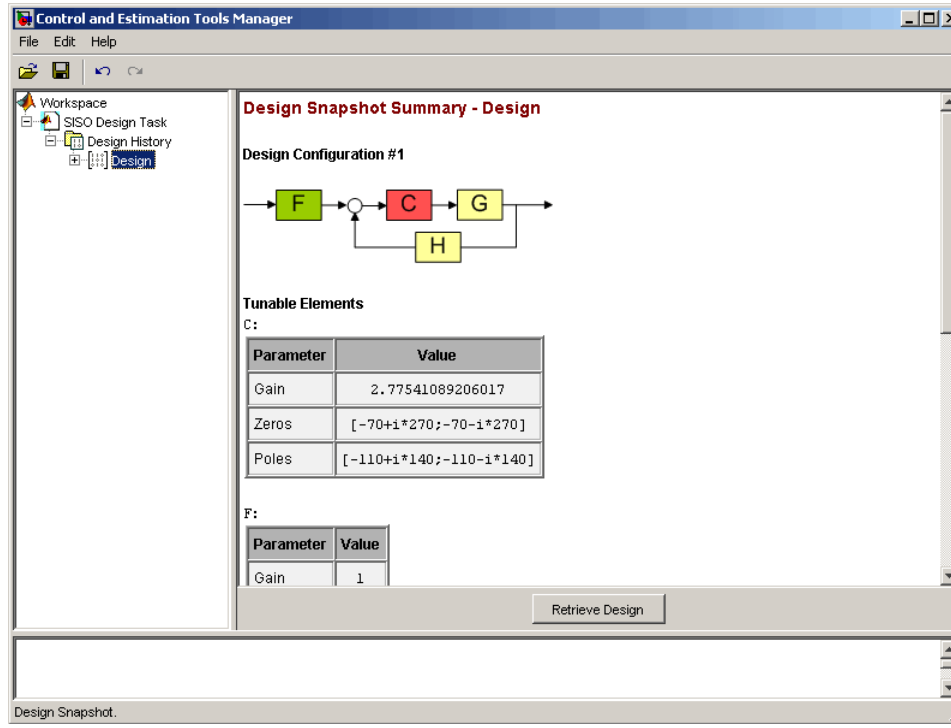
To retrieve intermediate designs, again click the **Design History** node or select **Store/Retrieve** from the **Designs** menu. From the **Design History** page, select the design to retrieve, and then click **Retrieve Design**, as shown next.



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

In this section...

"What Is Bode Diagram Design?" on page 4-54
"Example: DC Motor" on page 4-54
"Adjusting the Compensator Gain" on page 4-55
"Adjusting the Bandwidth" on page 4-56
"Adding an Integrator" on page 4-58
"Adding a Lead Network" on page 4-62
"Moving Compensator Poles and Zeros" on page 4-67
"Adding a Notch Filter" on page 4-70
"Modifying a Prefilter" on page 4-75

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

Example: 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:

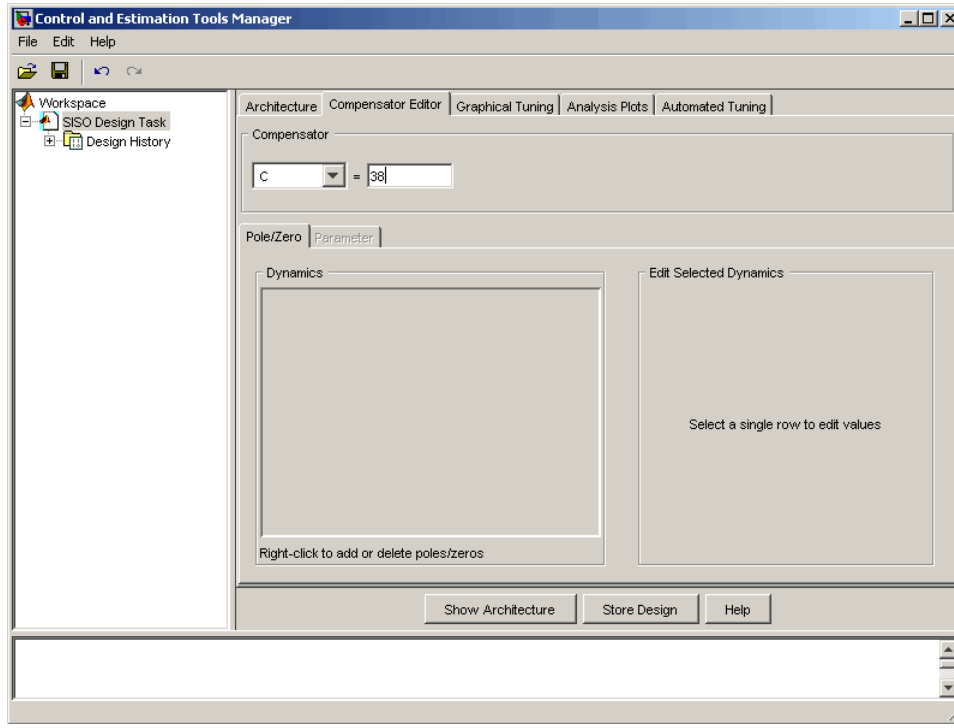
- Rise time of less than 0.5 second
- Steady-state error of less than 5%
- Overshoot of less than 10%
- Gain margin greater than 20 dB
- Phase margin greater than 40 degrees

Adjusting the Compensator Gain

The LTI Viewer Showing the Step Response for the DC Motor on page 4-38, 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.

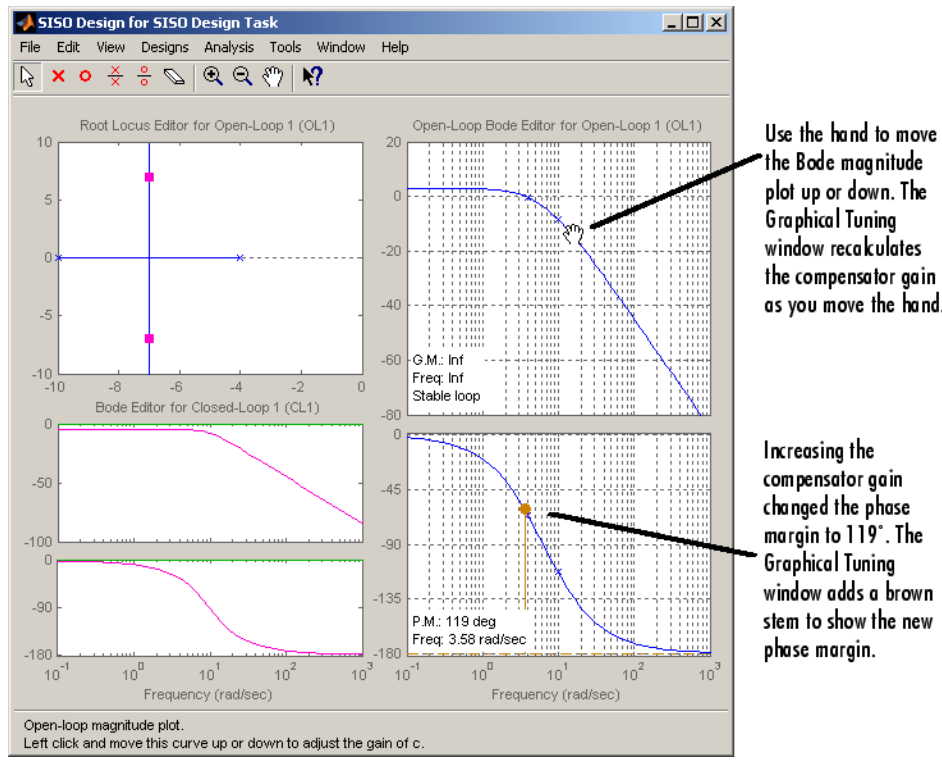
Adjusting the Bandwidth

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.

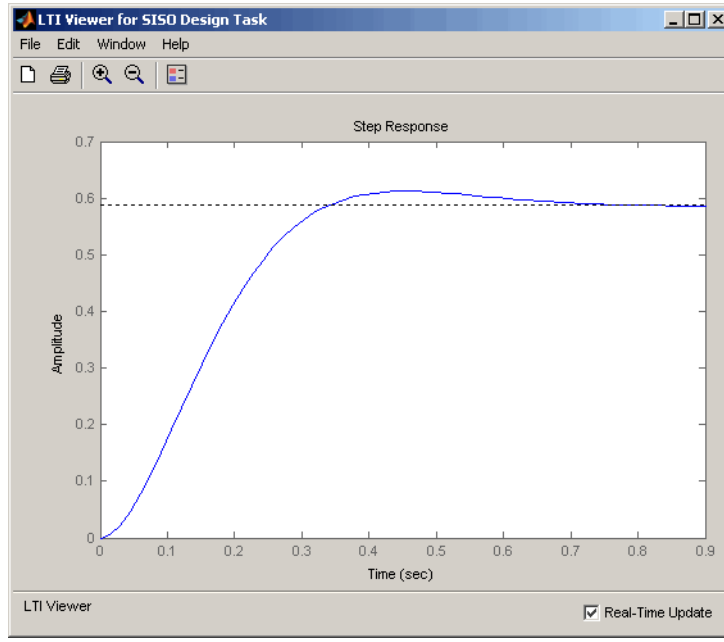
For a crossover at 3 rad/s, the compensator gain should be about 38. By default, the Graphical Tuning window shows gain and phase margin information in the lower-left corners of the Bode diagrams. In the Bode magnitude plot, it also tells you if your closed-loop system is stable or unstable.

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

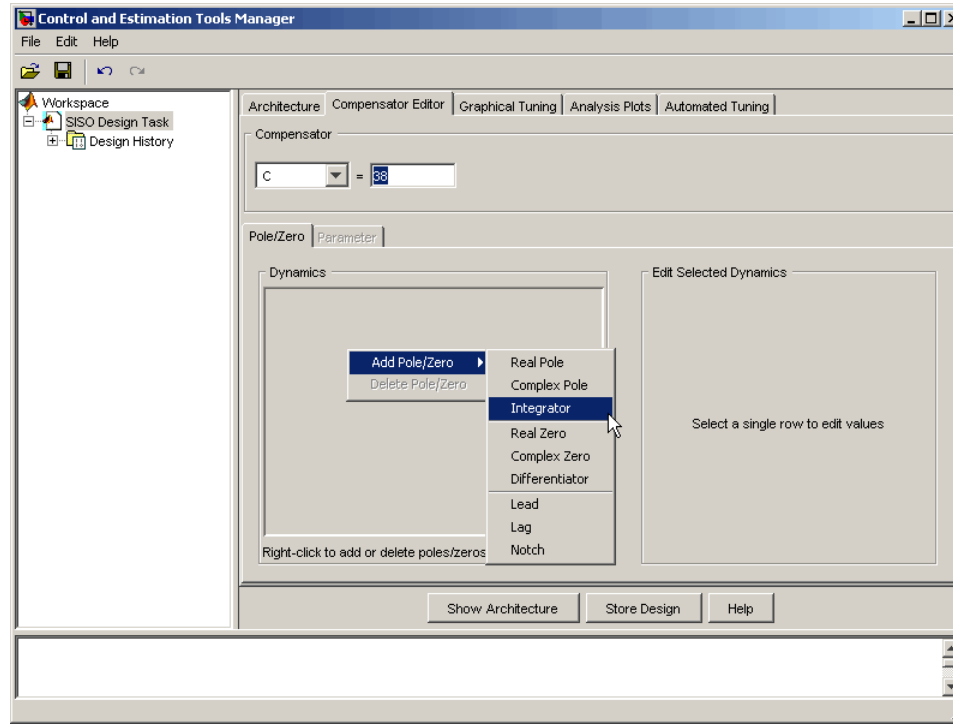
The step response shows that the steady-state error and rise time have improved somewhat, but you must design a more sophisticated controller to meet all the design specifications, in particular, the steady-state error requirement.

Adding an Integrator

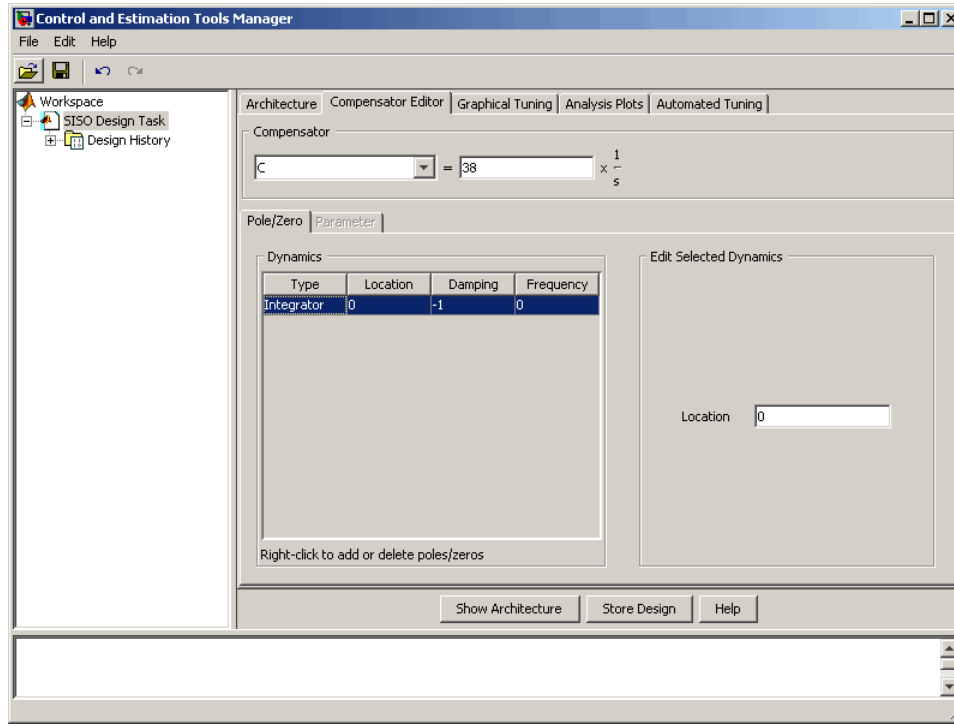
One way to eliminate steady-state error is to add an integrator. To add an integrator:

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.



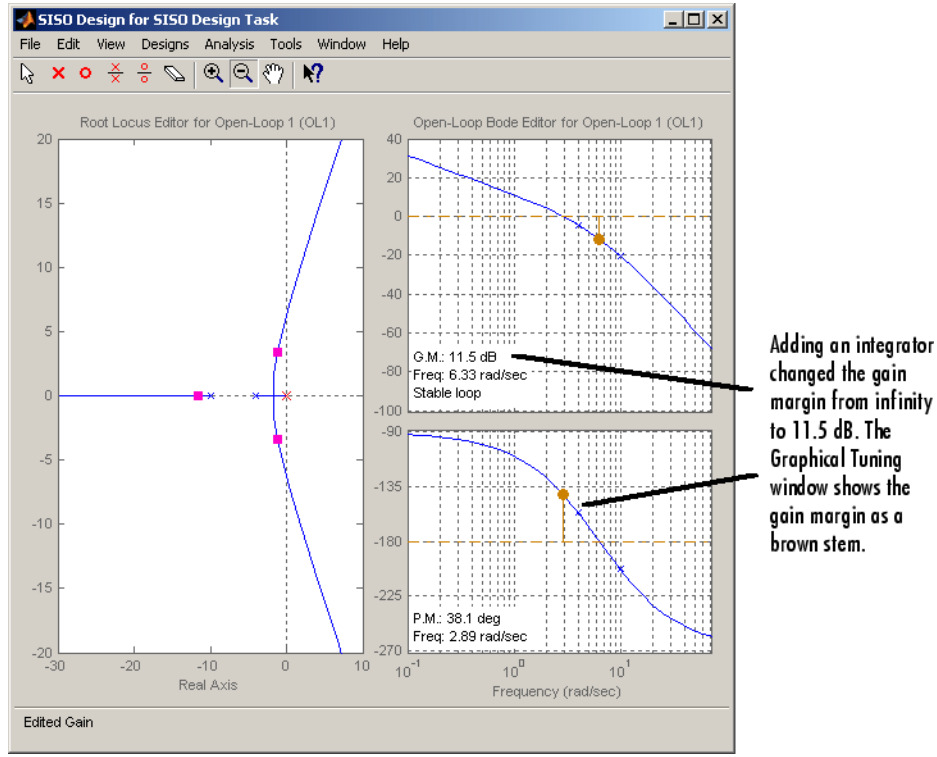
Adding an Integrator in the Dynamics Table



Editable Integrator Parameters

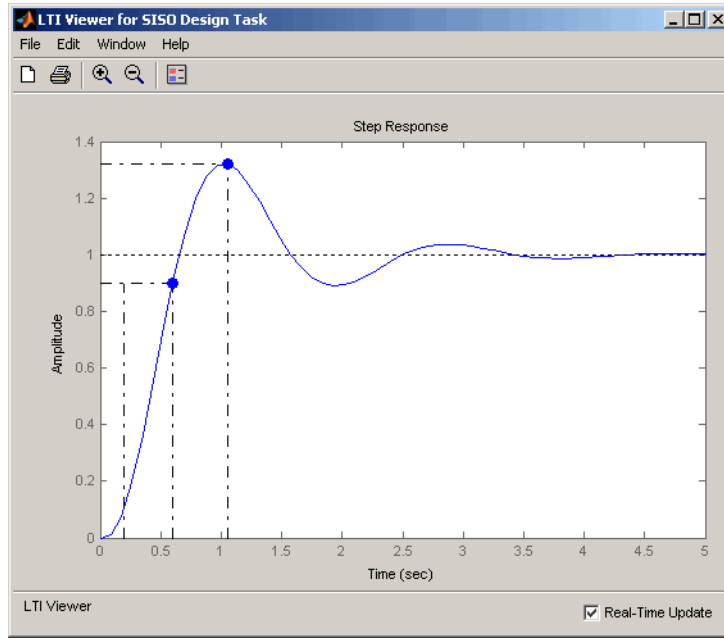
Notice adding the integrator changed the crossover frequency of the system. Readjust the compensator gain in the **Compensator Editor** page to bring the crossover back to 3 rad/s; the gain should be 99.

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

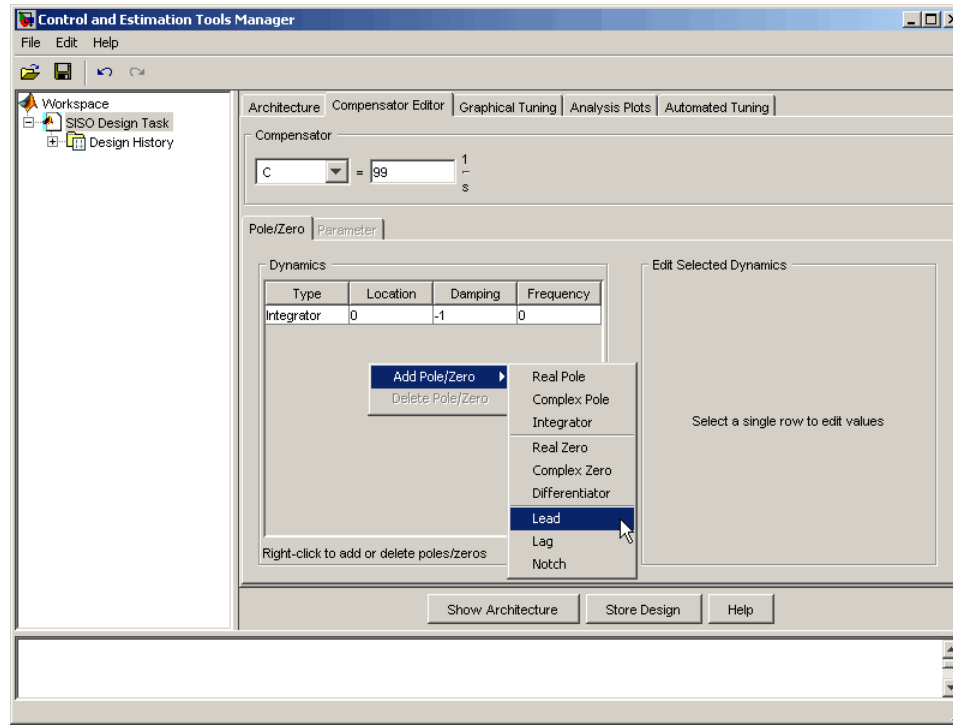
Part of the design requirements is a gain margin of 20 dB or greater and a phase margin of 40° or more. In the current compensator design, the gain margin is 11.5 dB and the phase margin is 38.1°, both of which fail to meet the design requirements. The rise time needs to be shortened while improving 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.

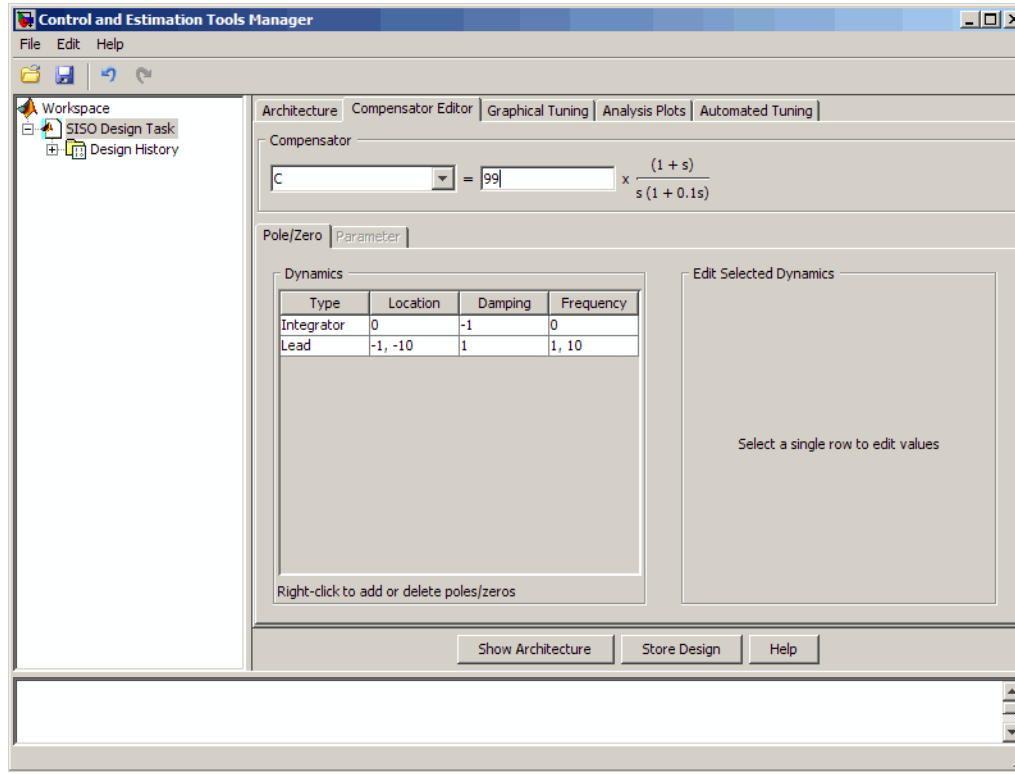
One possible solution is to add a lead network to the compensator. To add the lead network:

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.

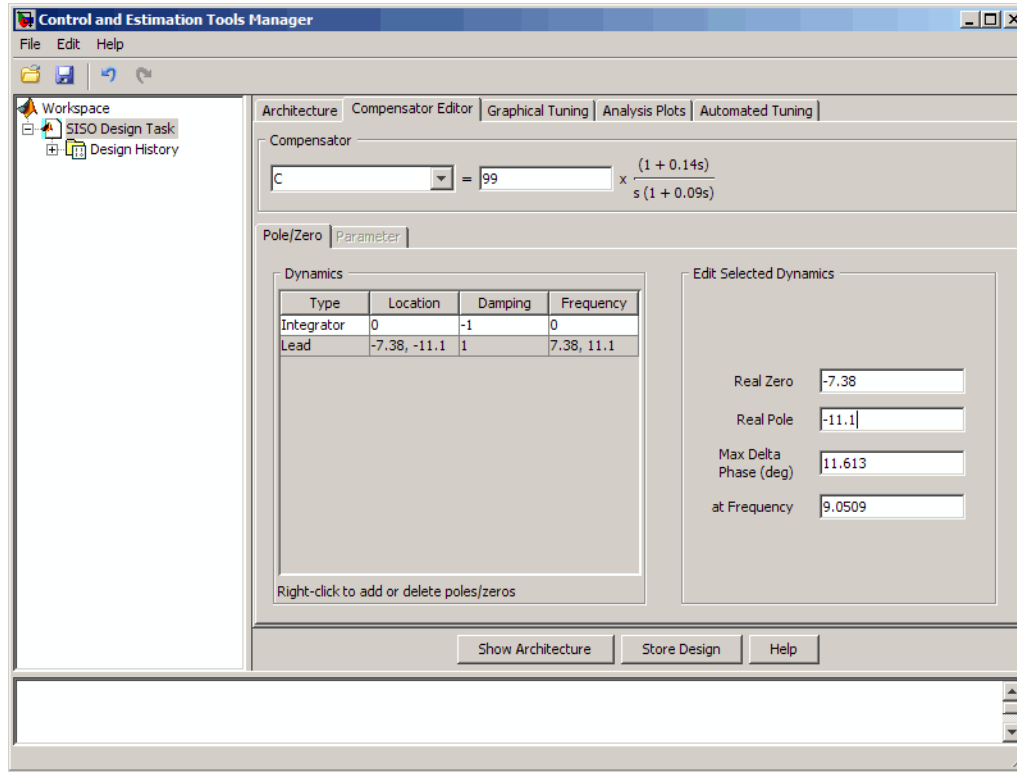


Adding a Lead Network to the DC Motor Compensator on the Compensator Editor Page



Lead Network Added

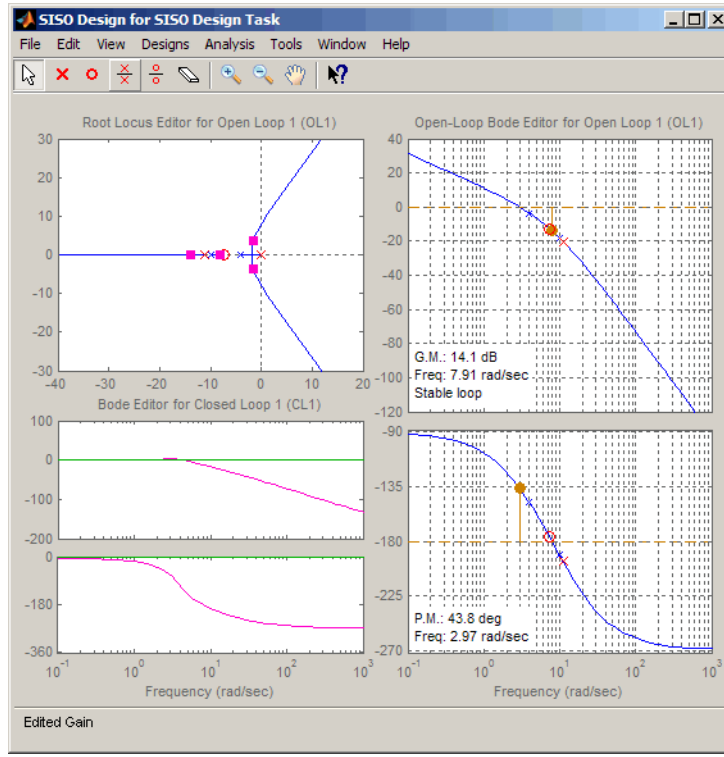
Editable fields are shown in the **Edit Selected Dynamics** group box (right side of page) when an item in the **Dynamics** table has been selected, as shown in the following figure.



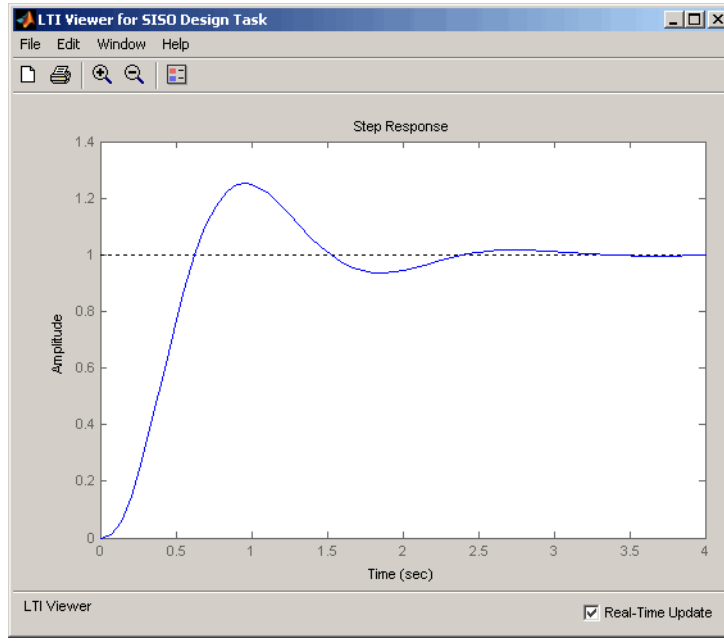
For this example, change Real Zero to `-7.38` and change Real Pole to `-11.1`.

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.

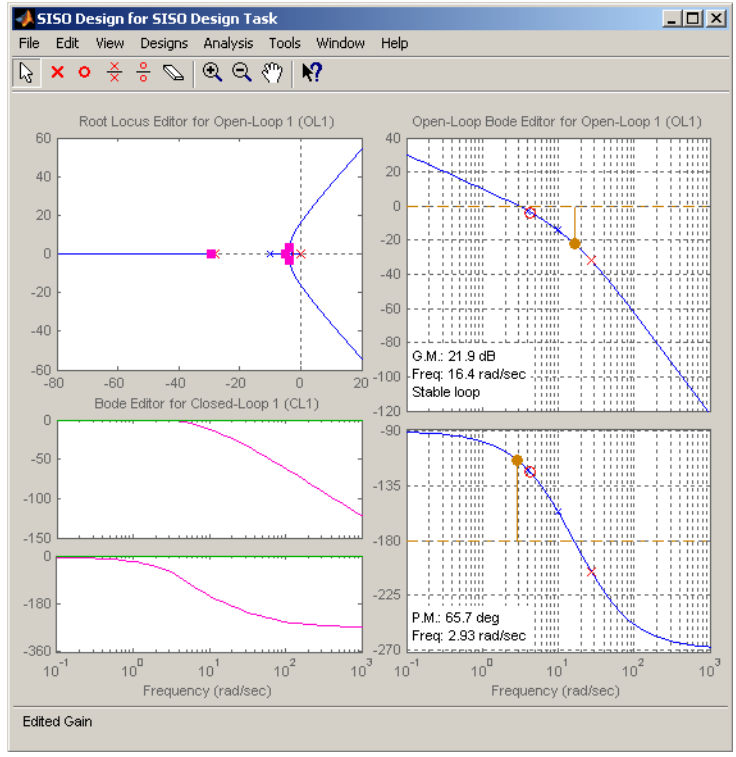
Moving Compensator Poles and Zeros

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.

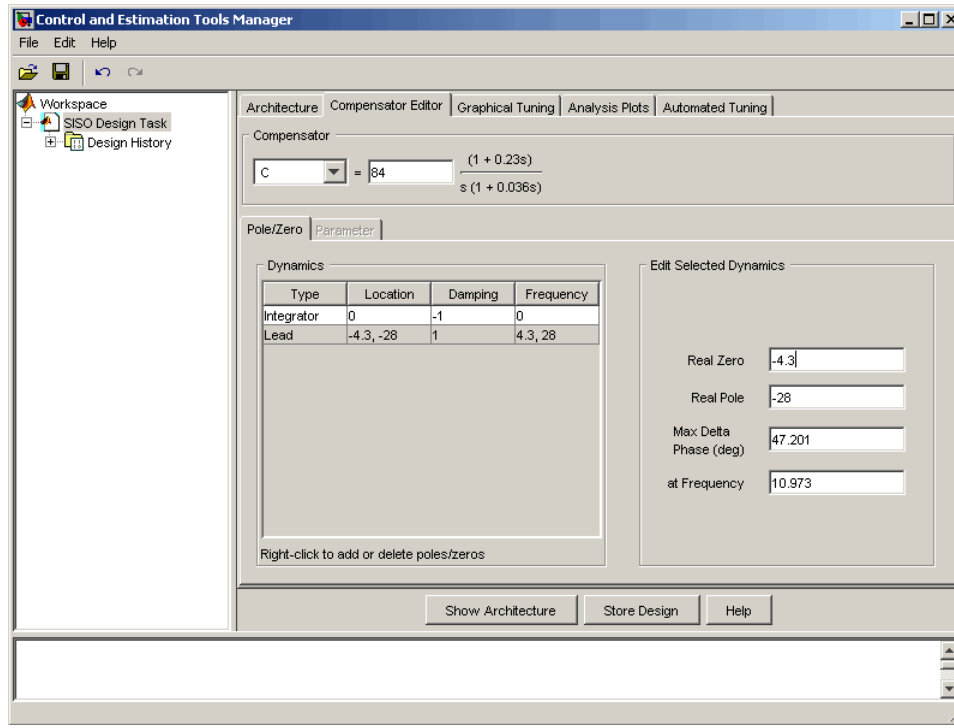


Graphical Tuning Window with Final Design Parameters for the DC Motor Compensator

The values for this final design are as follows:

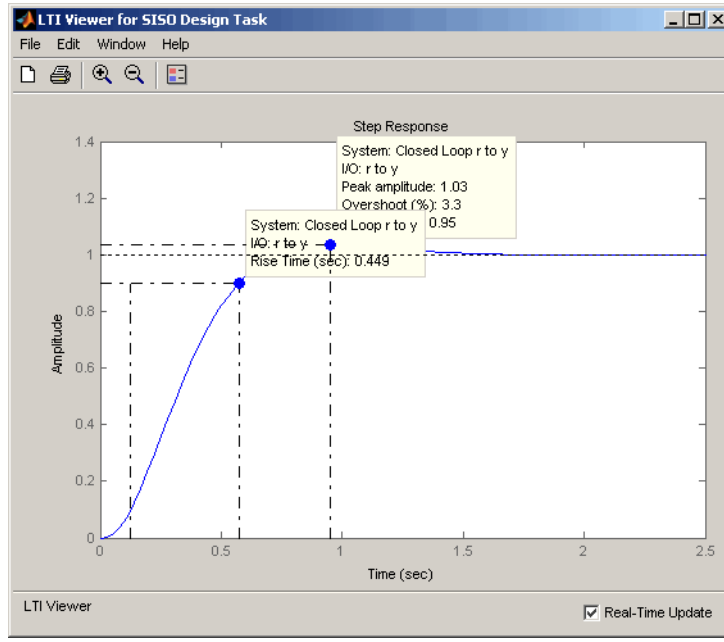
- Poles at 0 and -28
- Zero at -4.3
- Gain = 84

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.

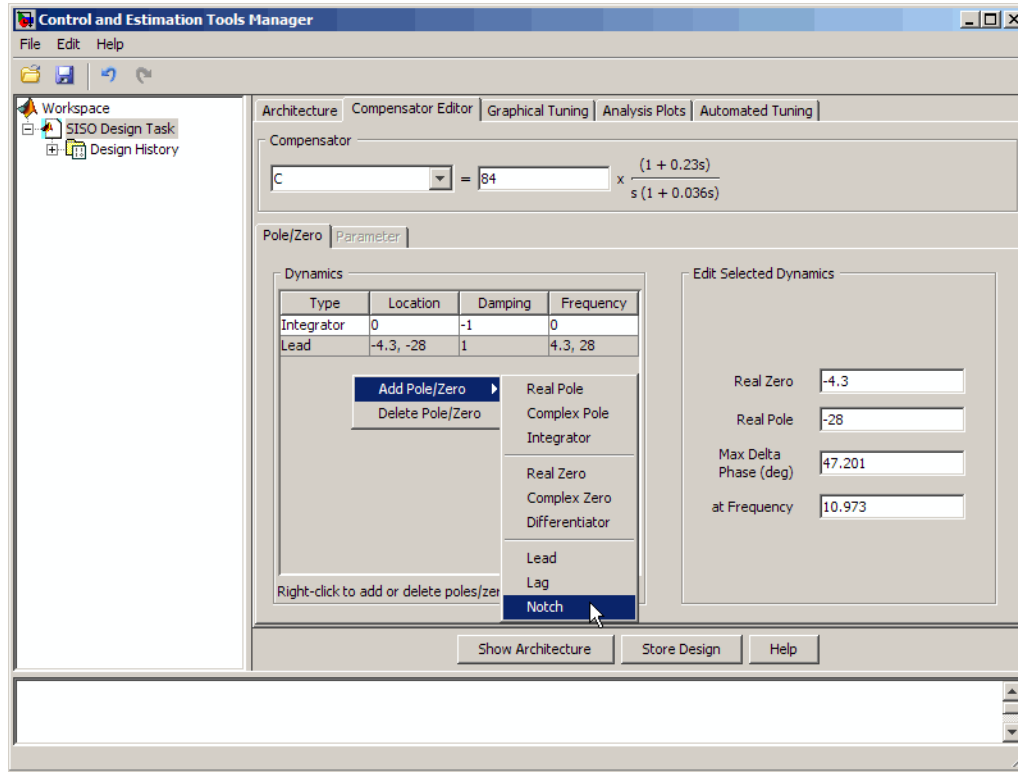


Step Response for the Final Compensator Design

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.

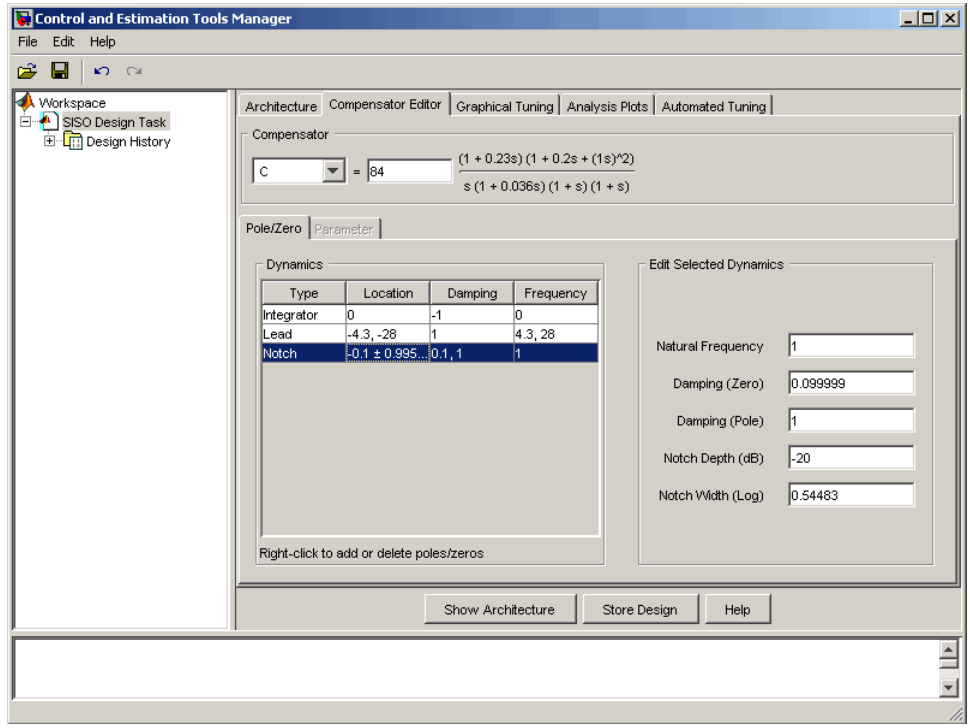
Adding a Notch Filter

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.



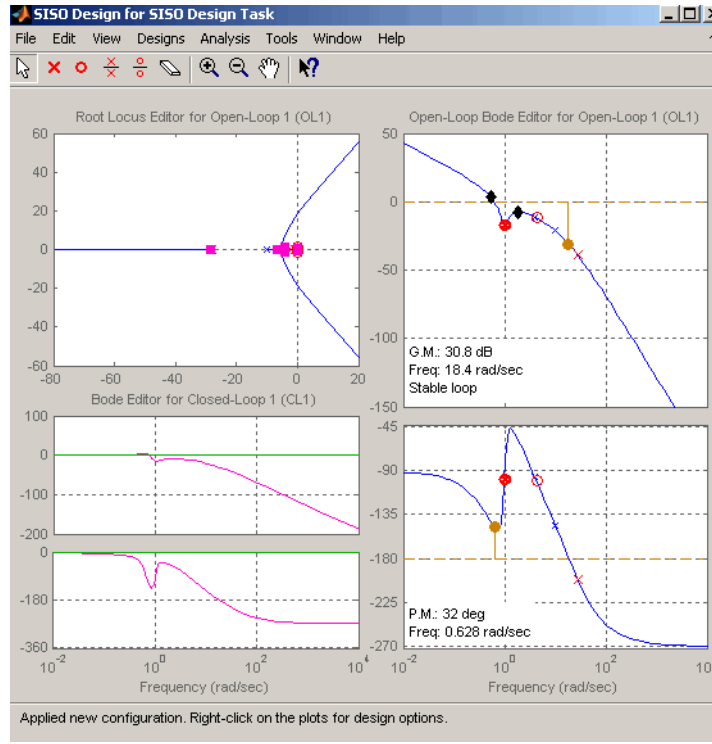
Adding a Notch Filter with the Dynamics Right-Click Menu

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

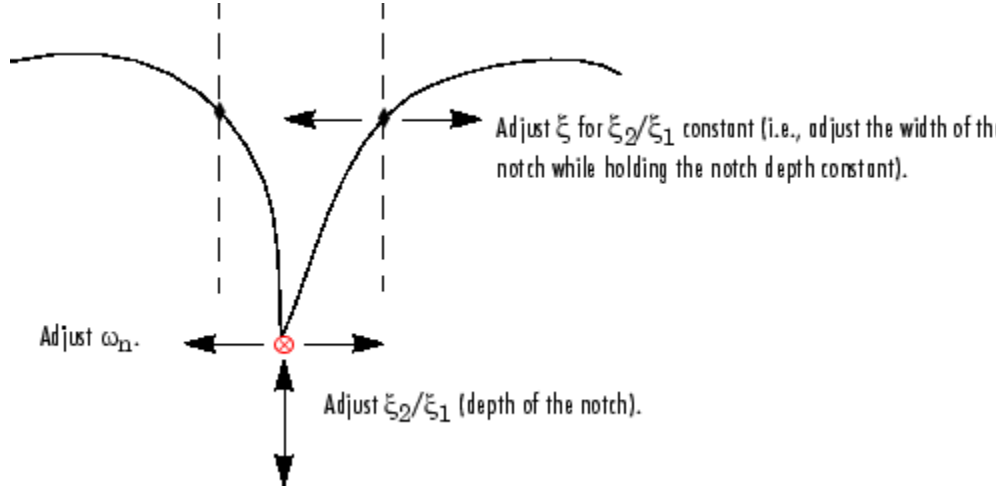
icon on the Graphical Tuning window. In the Open-Loop Bode Editor, press the left mouse button and drag your mouse to draw a box around the notch filter. When you release the mouse, the Graphical Tuning window will zoom in on the selected region.

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

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

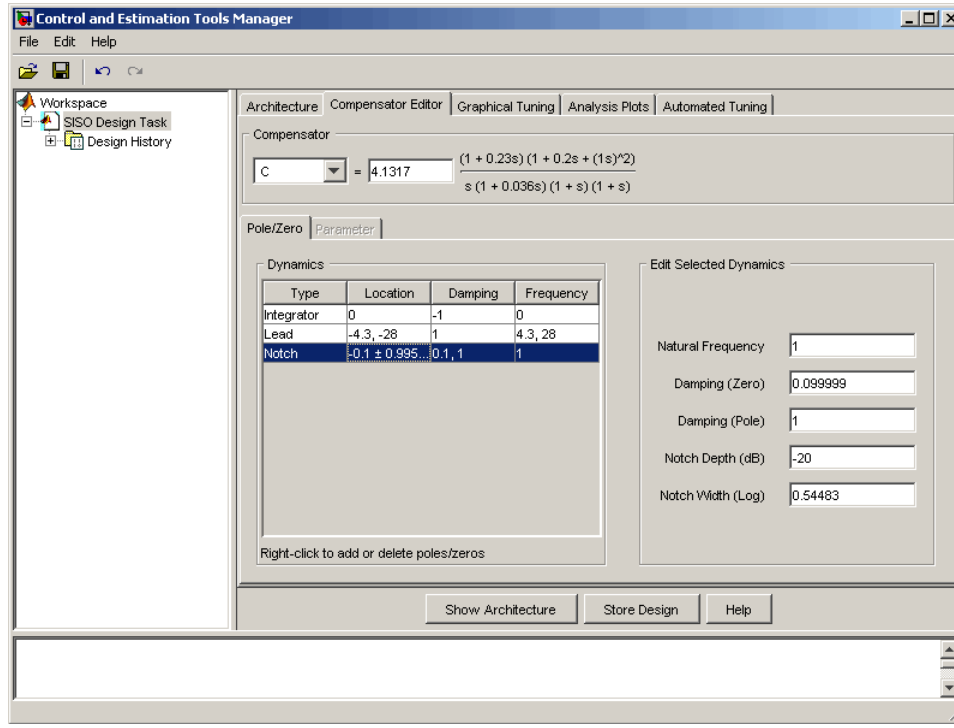
The three adjustable parameters are ξ_1, ξ_2, and ω_n. The ratio of ξ_2/ξ_1 sets the depth of the notch, and ω_n is the natural frequency of the notch.

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

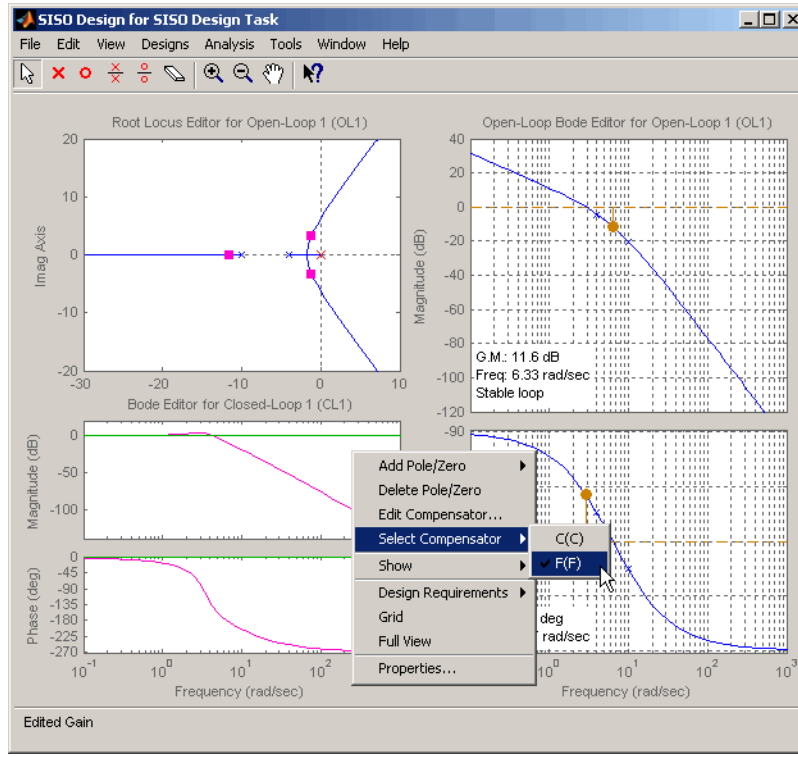
Modifying a Prefilter

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

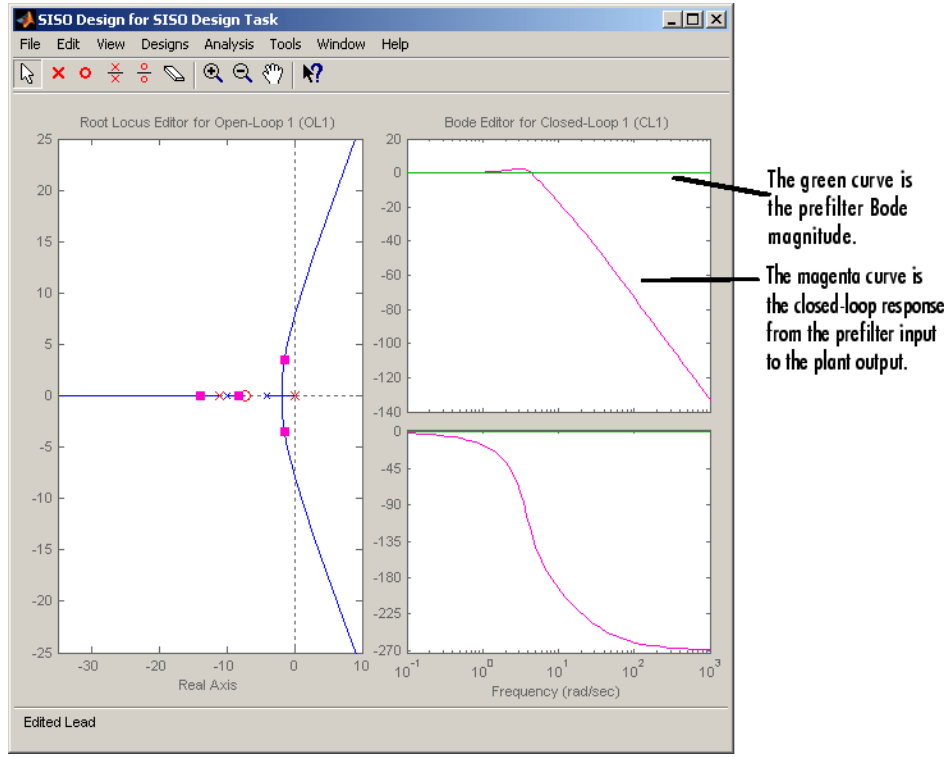
A common prefilter is a simple lowpass filter that reduces noise in the input signal.

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

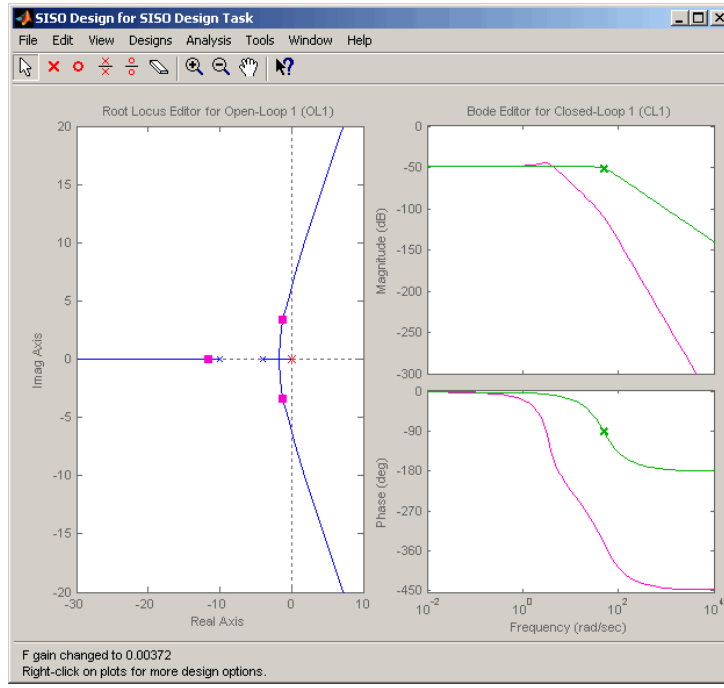
For clarity, the previous figure does not show the open-loop Bode diagram for the compensator (**C**). To remove the Bode diagram from the Graphical Tuning window, go to the **SISO Design Task** node on the Control and Estimation Tools Manager, click the **Graphical Tuning** tab, and for Plot 2, Open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



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

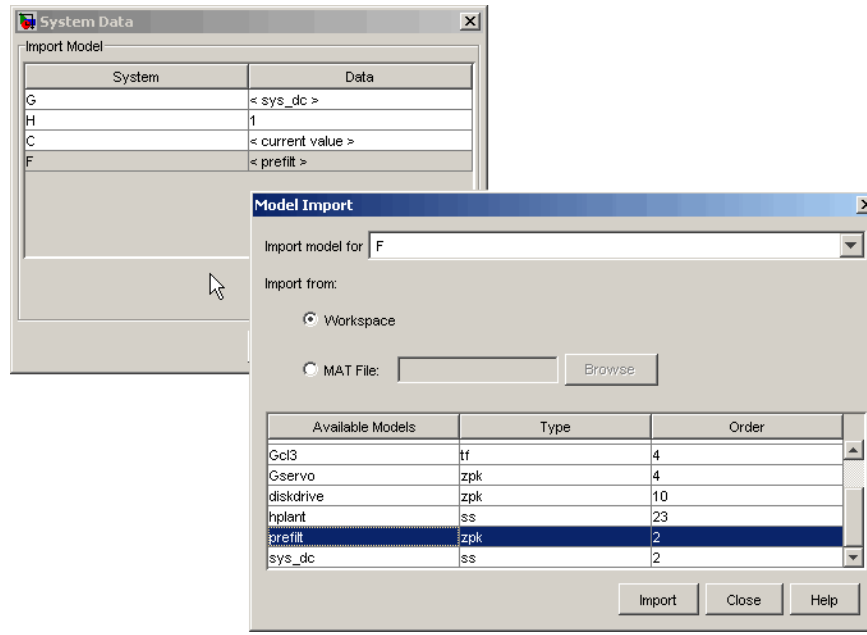
By default, the damping ratio of the complex pair is 1.0, which means that there are two real-valued poles at about -50 rad/s. The green curve, which represents the prefilter Bode response, shows the -3 dB point for the roll-off is at about 50 rad/s. The magenta curve, which represents the closed-loop 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

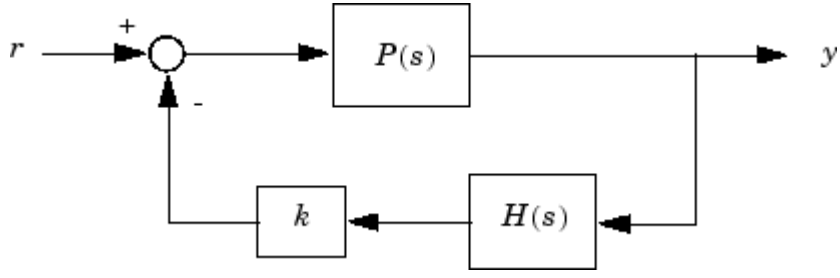
In this section...
"What Is Root Locus Design?" on page 4-80
"Example: Electrohydraulic Servomechanism" on page 4-81
"Changing the Compensator Gain" on page 4-87
"Adding Poles and Zeros to the Compensator" on page 4-90
"Editing Compensator Pole and Zero Locations" on page 4-95
"Viewing Damping Ratios" on page 4-99
"Exporting the Compensator and Models" on page 4-48
"Storing and Retrieving Intermediate Designs" on page 4-50

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.

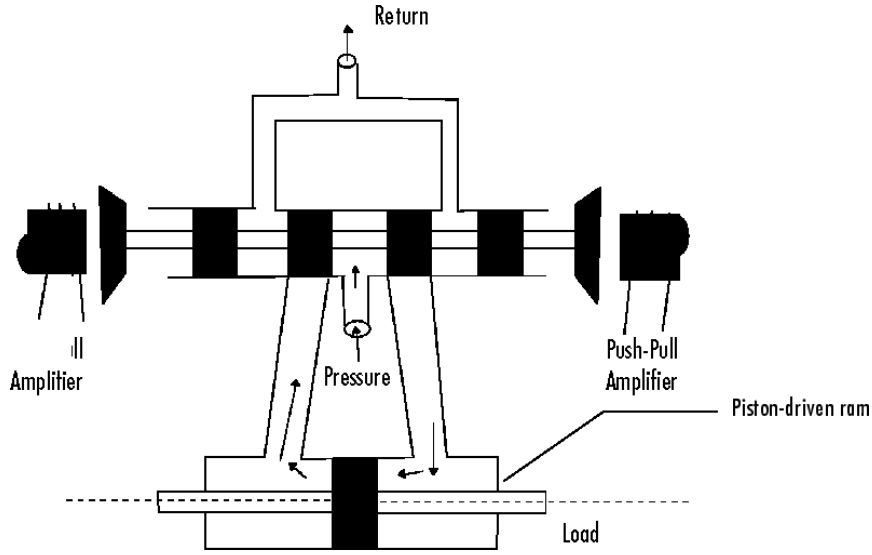
The SISO Design Tool's root locus and Bode diagram design tools provide complementary perspectives on the same design issues; each perspective offers insight into the design process. Because the SISO Design Tool shows both root locus and Bode diagrams, you can also choose to combine elements of both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

Example: 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

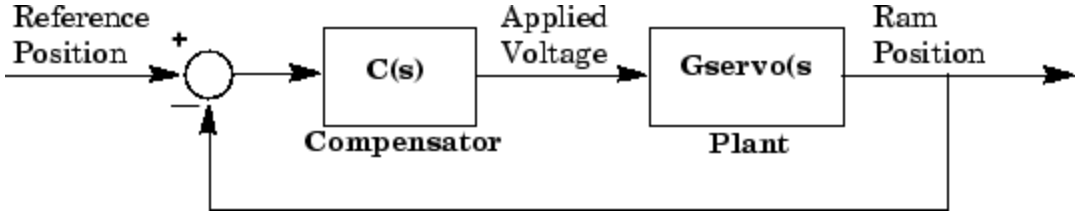
This figure shows a schematic of this servomechanism.



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.



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%.

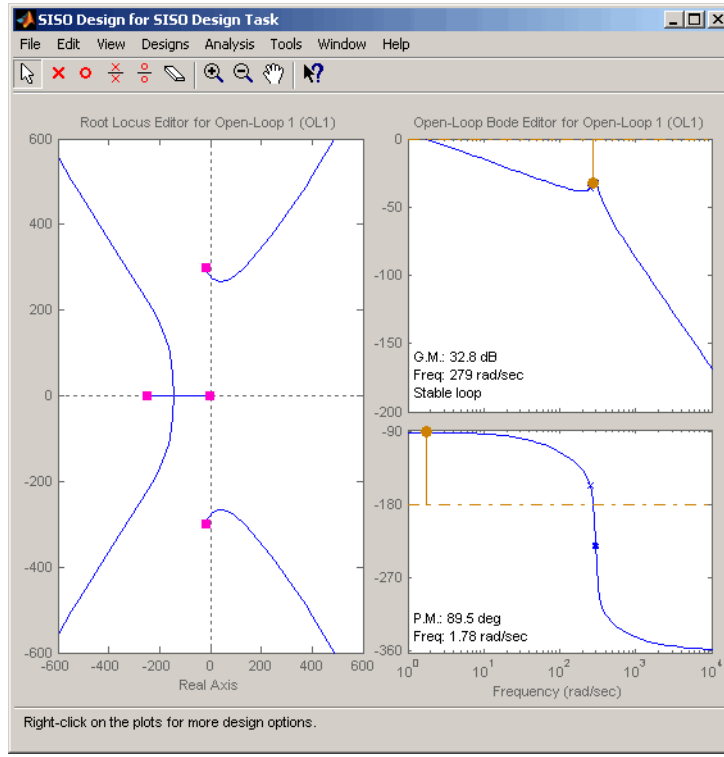
The remainder of this topic discusses how to use the SISO Design Tool to design a controller to meet these specifications.

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.

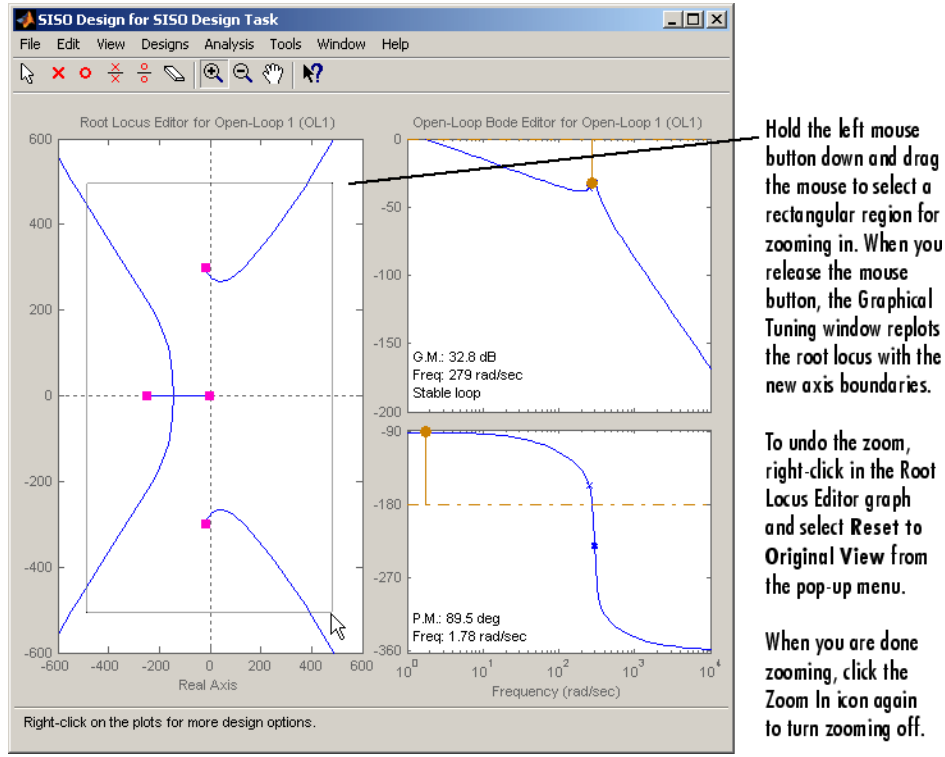


Graphical Tuning Window Showing the Root Locus and Bode Plots for the Electrohydraulic Servomechanism Plant

Zooming

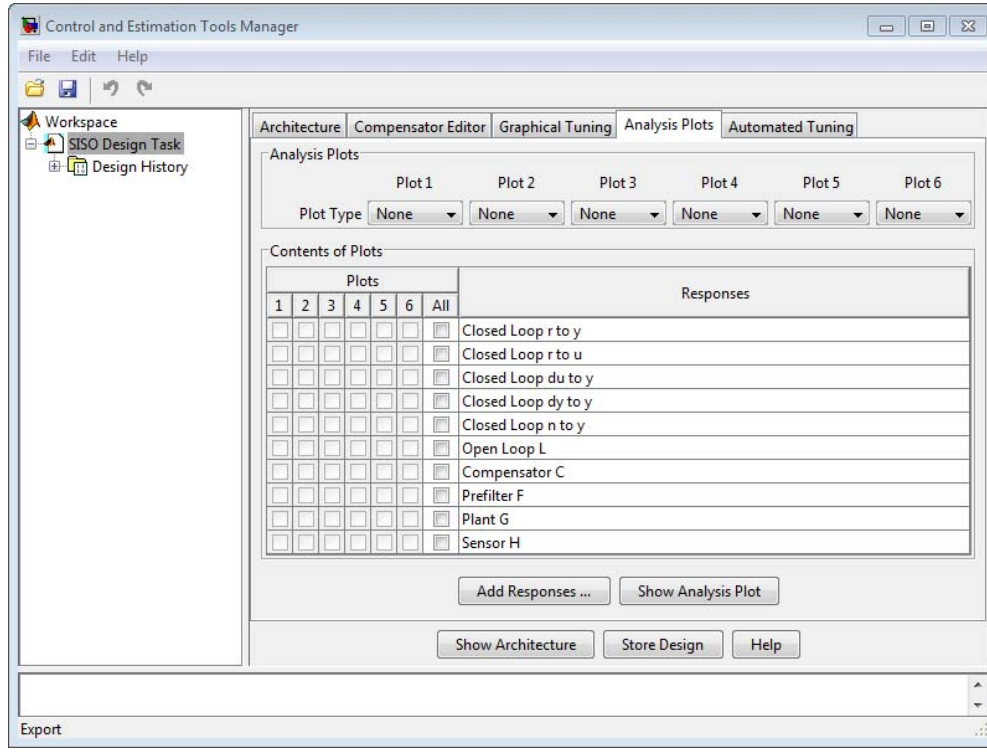
Click the Zoom In

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.



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.

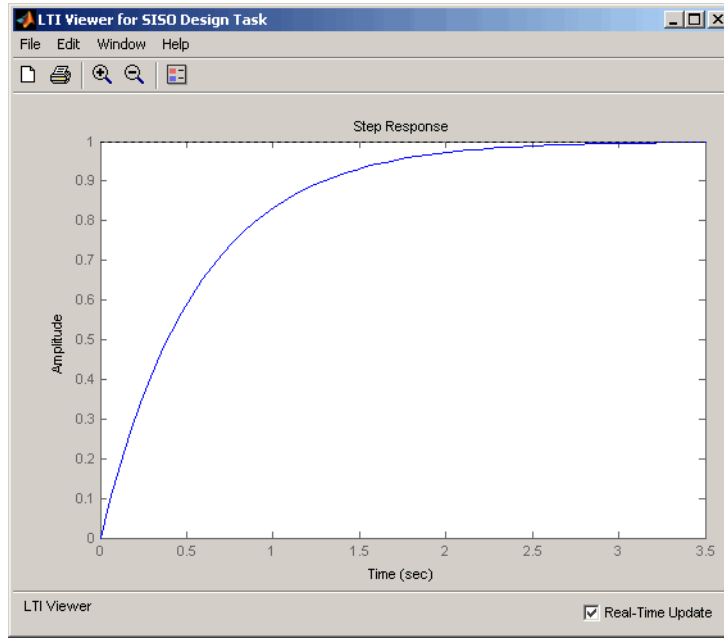


Analysis Plots Loop Response Selection

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.



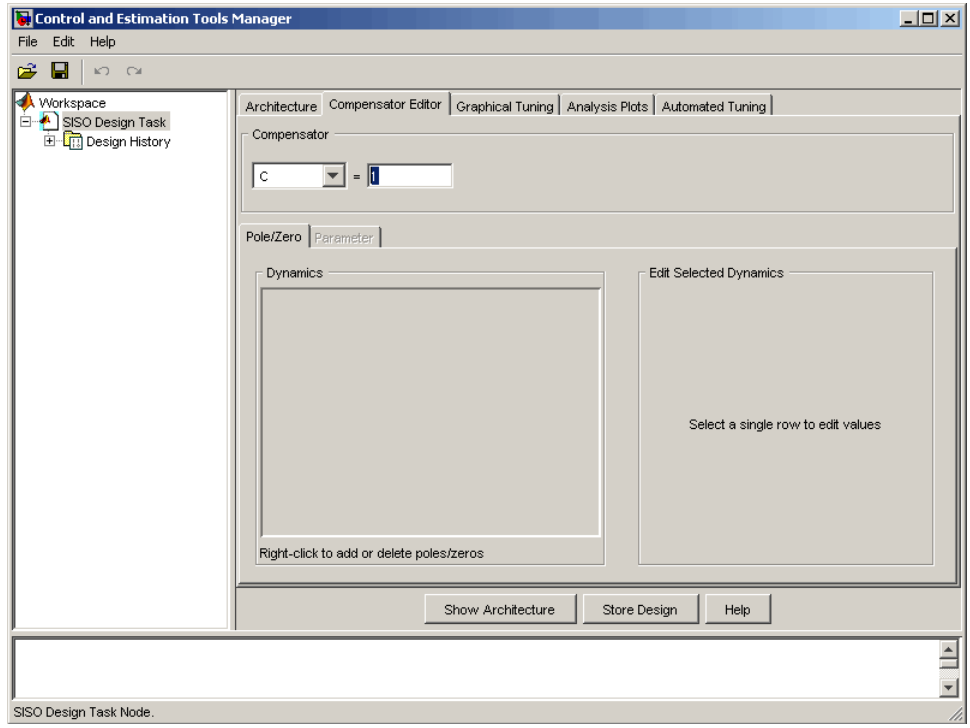
LTI Viewer for the Electrohydraulic Servomechanism

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 4-83.

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.

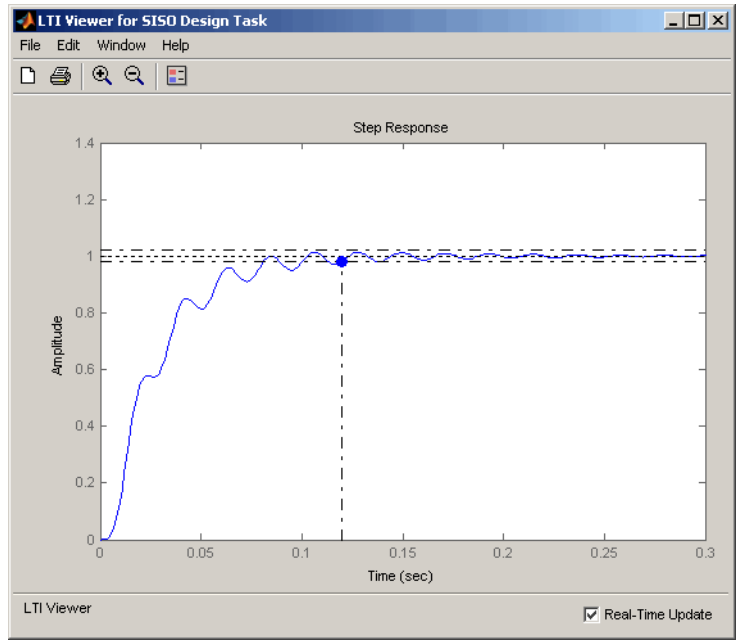
Experiment with different gains and view the closed-loop response in the associated LTI Viewer.

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.

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 4-90 shows how to design a compensator so that you meet the required specifications.

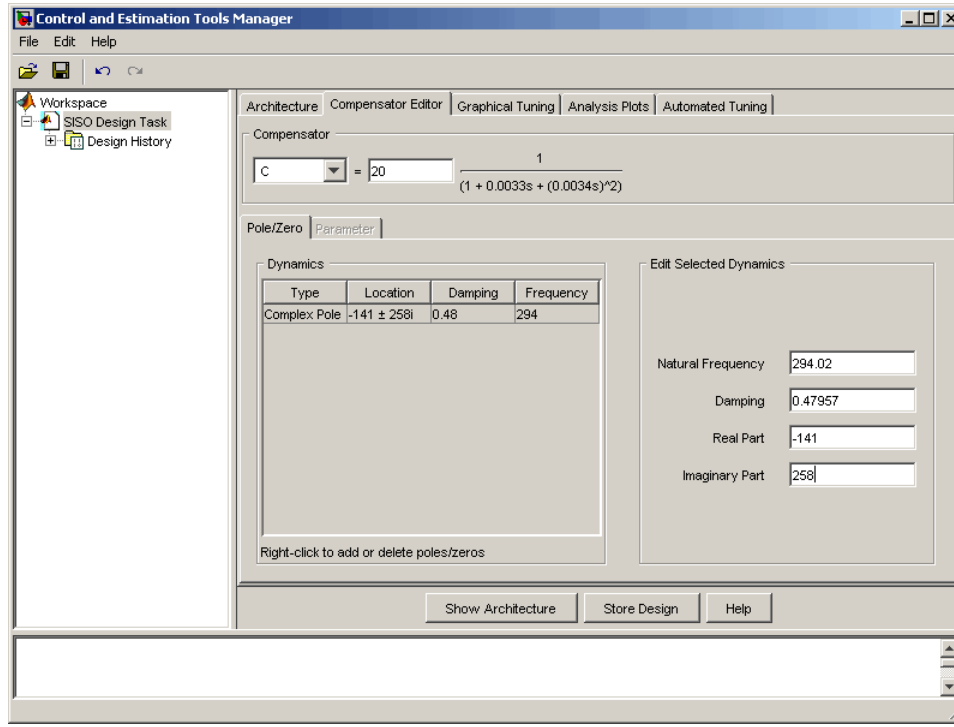
Adding Poles and Zeros to the Compensator

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 4-95.



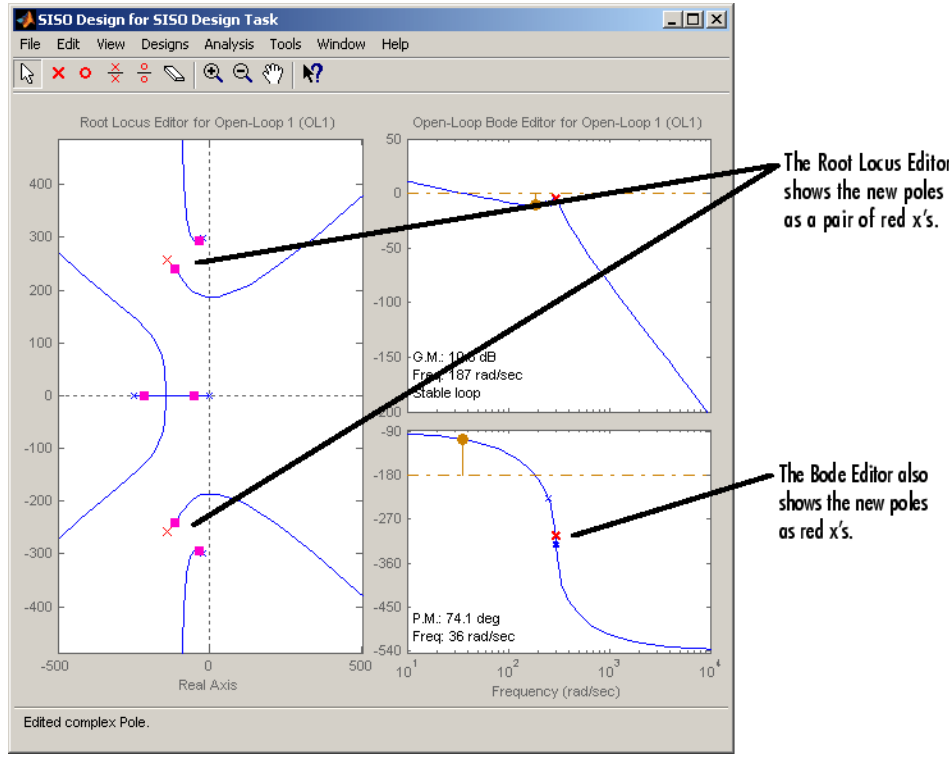
Adding a Complex Pair of Poles to the Compensator on the Compensator Editor Page

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.

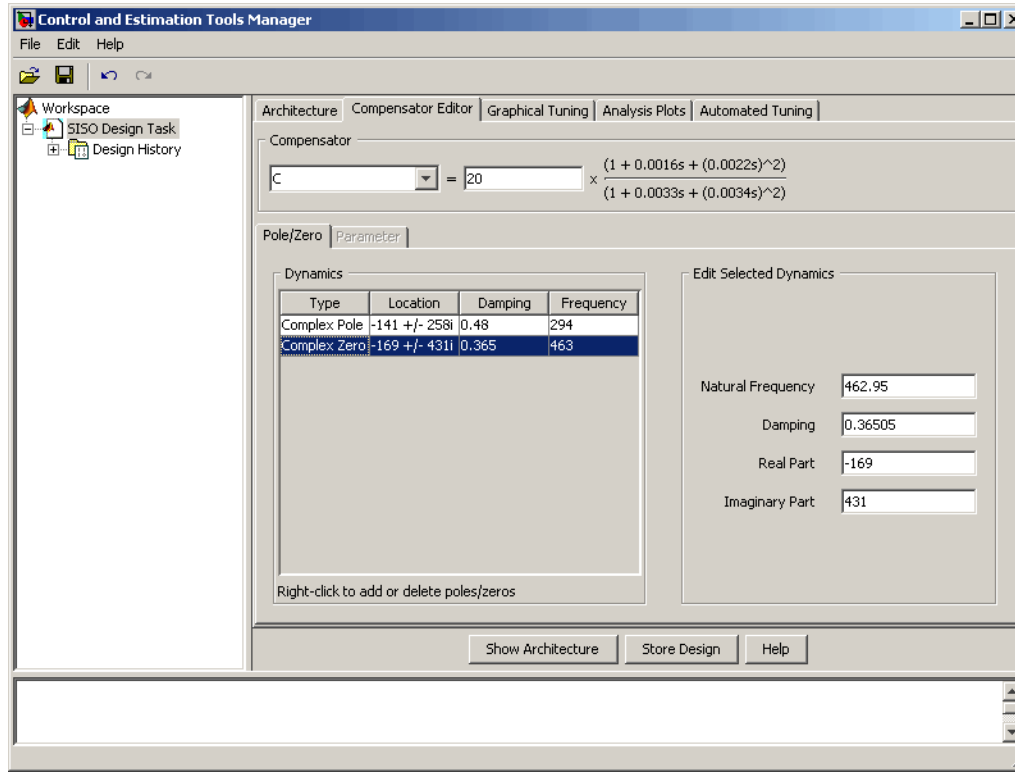


Result of Adding a Complex Pair of Poles to the Compensator

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 4-95.

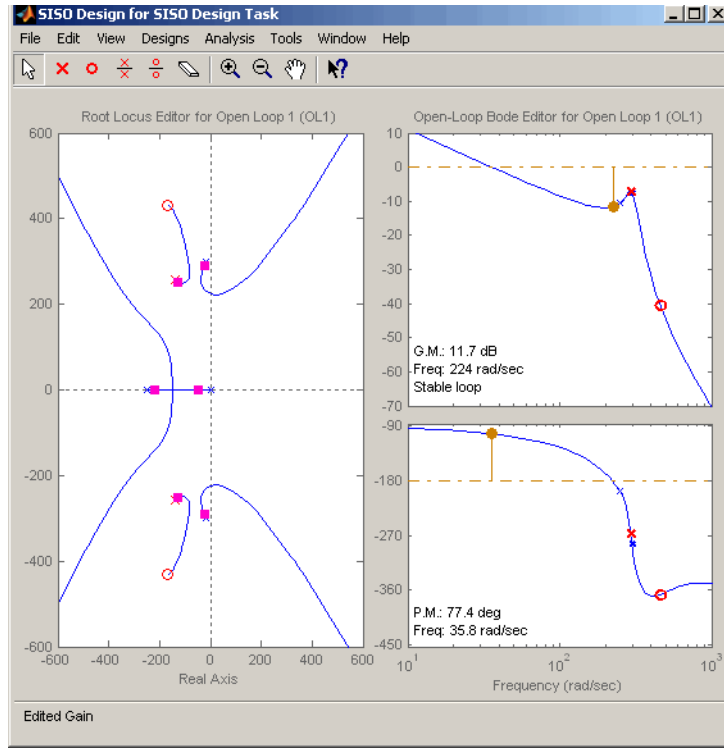


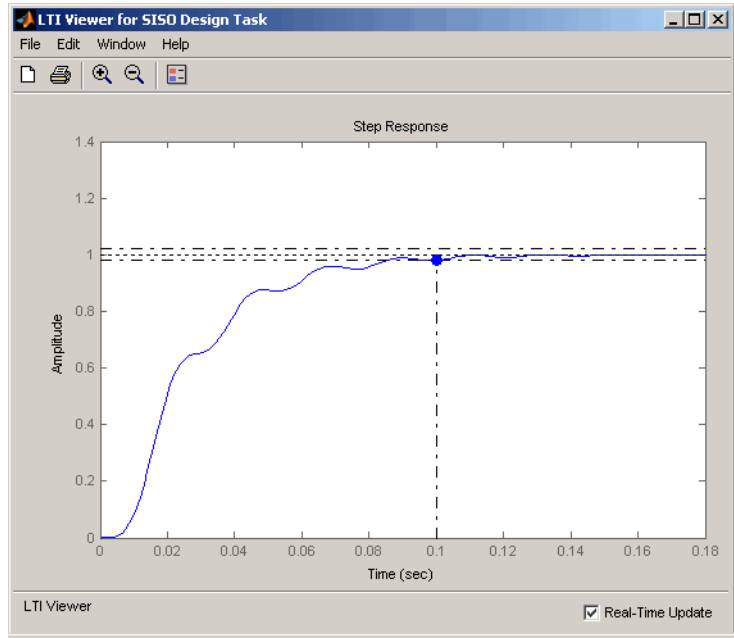
Adding Complex Zeros to the Compensator on the Compensator Editor Page

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.


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)
G.M.: 11.7 dB
Freq: 224 rad/sec
Stable loop
P.M.: 77.4 deg
Freq: 35.8 rad/sec
Real Axis
Frequency (rad/sec)
Edited Gain



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.

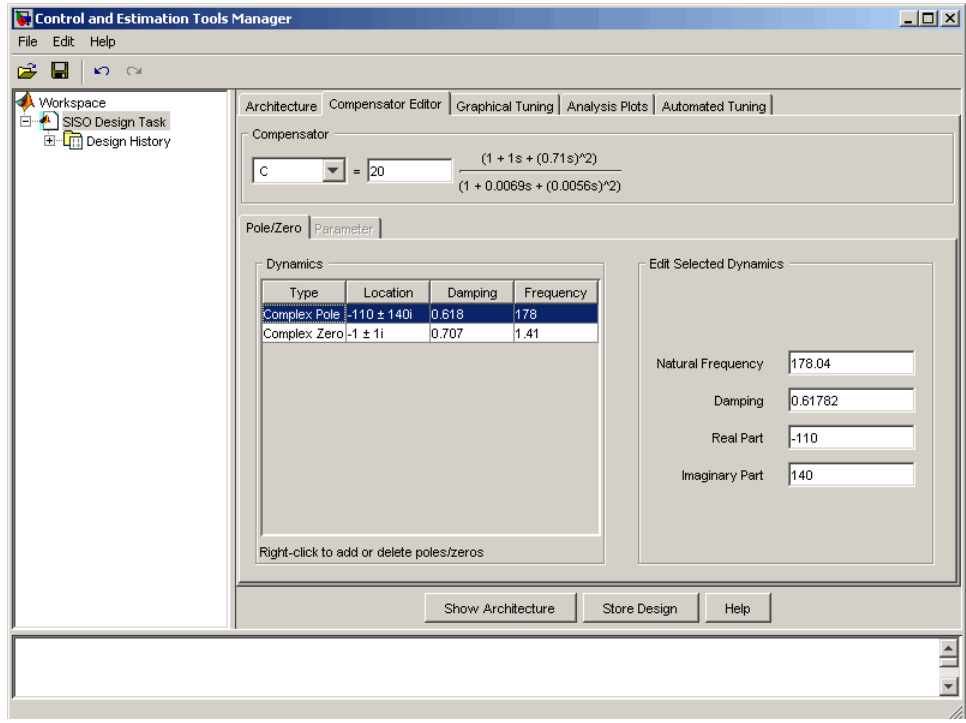
As you can see, the compensator design process can involve some trial and error. You can try dragging the compensator poles, compensator zeros, or the closed-loop poles around the root locus until you meet the design criteria.

"Editing Compensator Pole and Zero Locations" on page 4-95, 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:

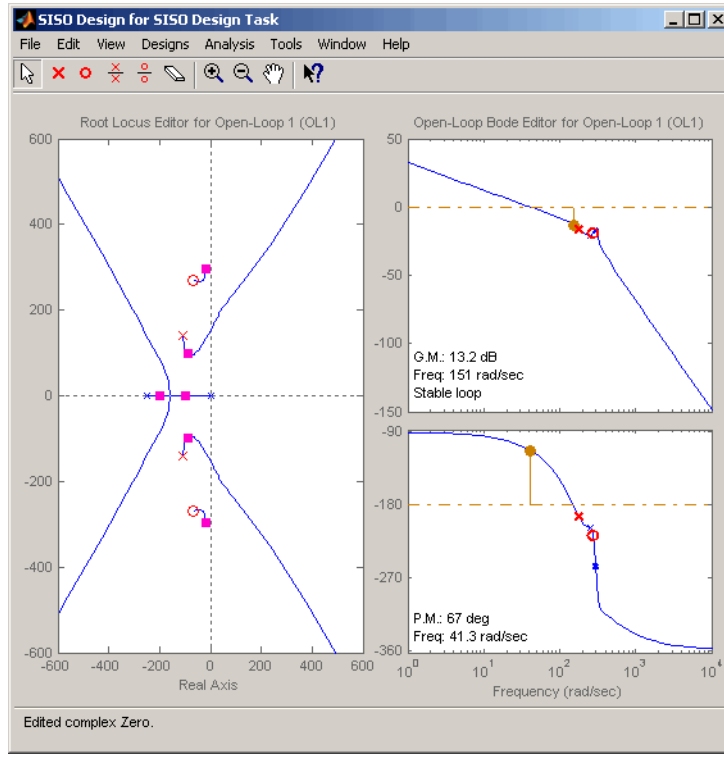
1. Select the compensator to edit in the list box to the left of the equal sign in the **Compensator** area.
2. Enter the gain value in the text box to the right of the equal sign in the **Compensator** area.

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.

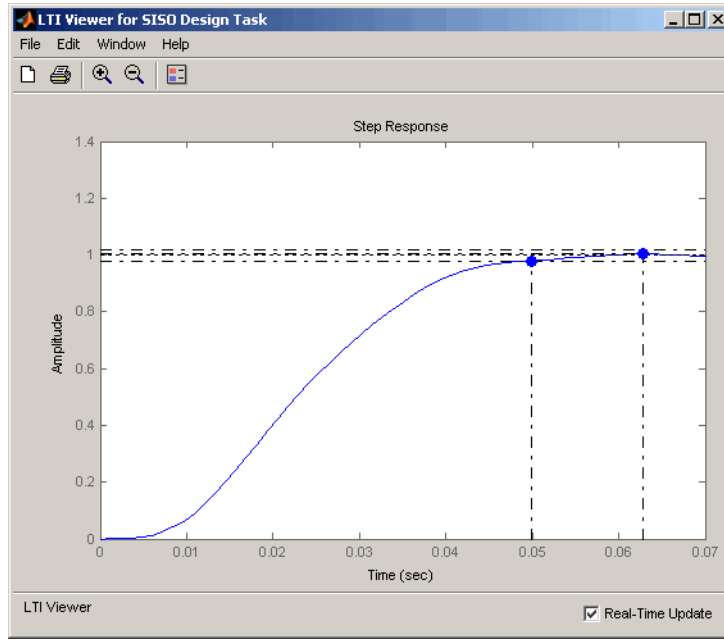
For this example, edit the poles to be at $-110 \pm 140i$ and the zeros at $-70 \pm 270i$. Set the compensator gain to `23.3`.

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.



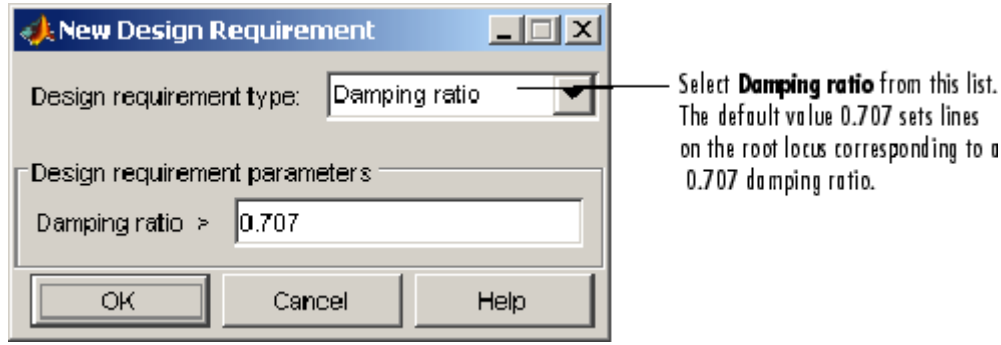
Closed-Loop Step Response for the Final Design of the Electrohydraulic Servomechanism Example

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.

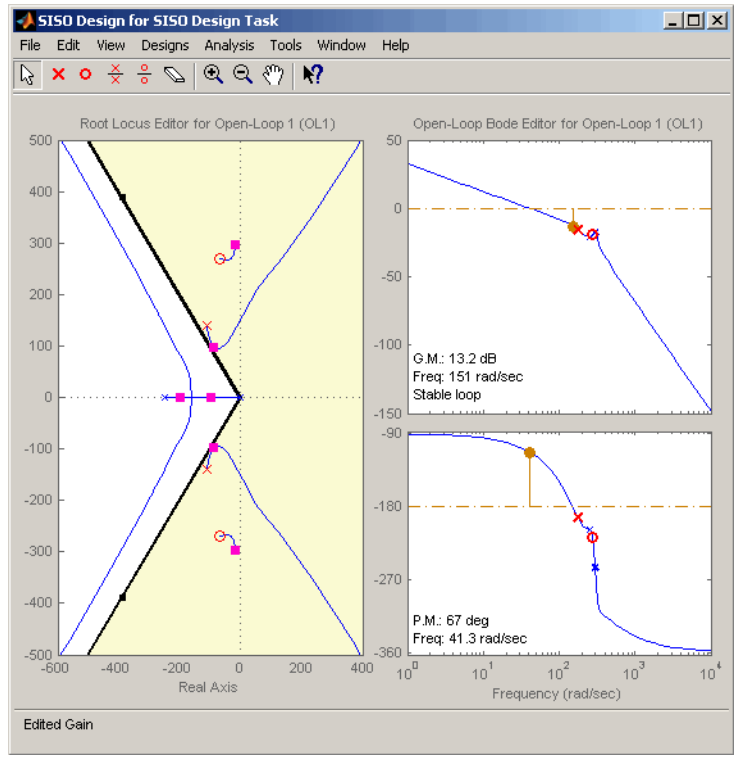
Viewing Damping Ratios

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.



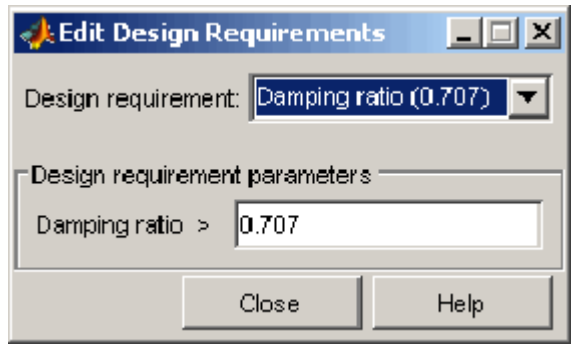
Applying damping ratio requirements to the root locus plot results in a pair of shaded rays at the desired slope, as this figure shows.



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

In this section...

"What Is Nichols Plot Design?" on page 4-102
"DC Motor Example" on page 4-102
"Opening a Nichols Plot" on page 4-103
"Adjusting the Compensator Gain" on page 4-104
"Adding an Integrator" on page 4-107
"Adding a Lead Network" on page 4-109

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 "Example: DC Motor" on page 4-54, 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.

DC Motor Example

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 4-83:

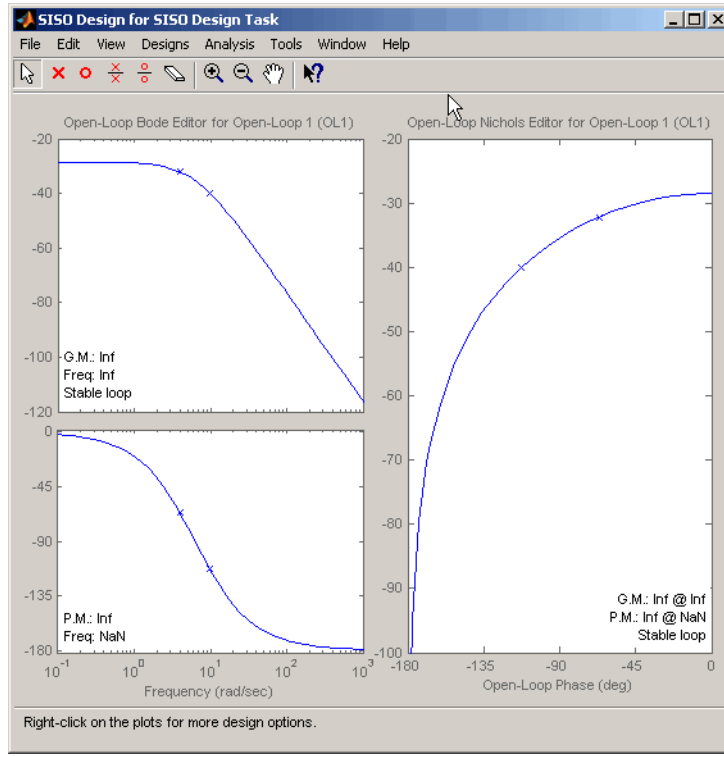
- Rise time of less than 0.5 second
- Steady-state error of less than 5%
- Overshoot of less than 10%
- Gain margin greater than 20 dB
- Phase margin greater than 40 degrees

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.

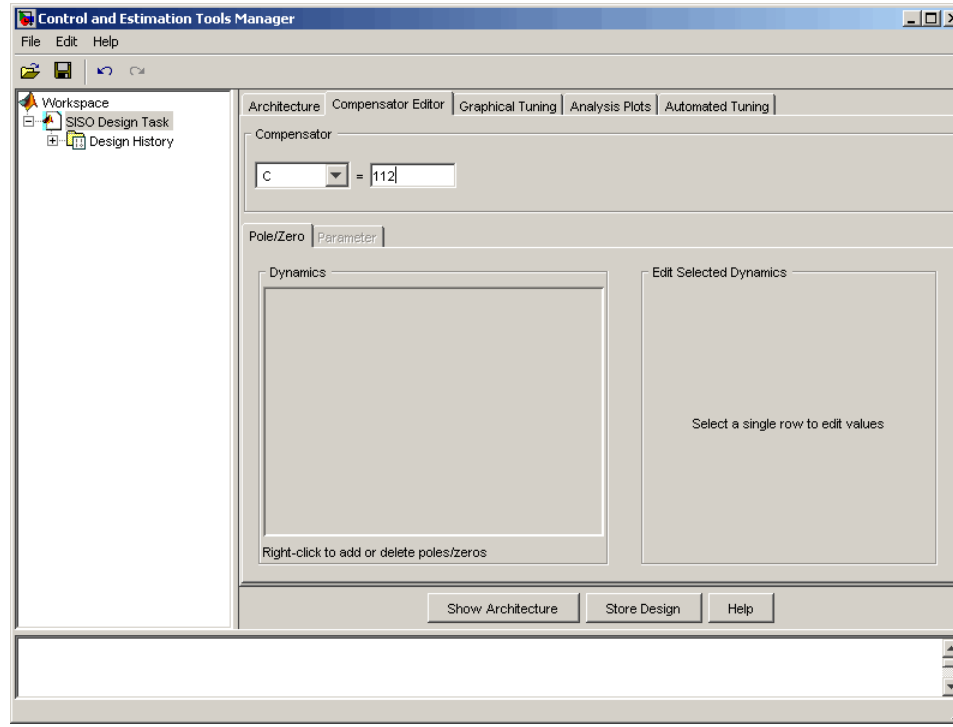


Graphical Tuning Window with a Bode Diagram and a Nichols Plot

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**.

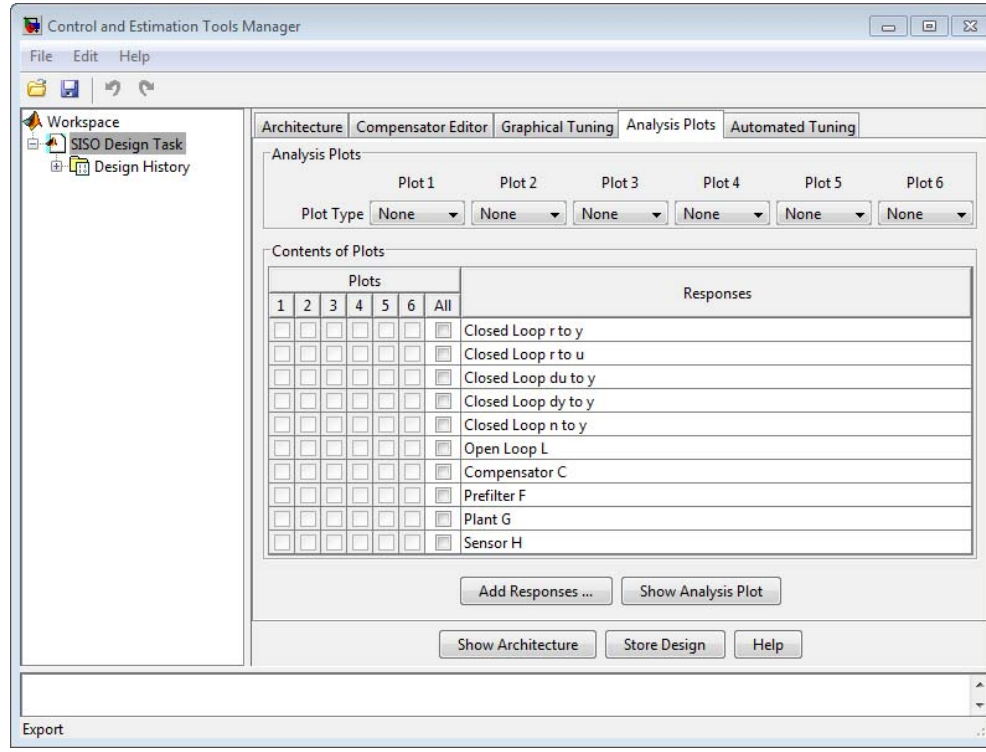


Adjusting Compensator Gain in the Compensator Editor Page

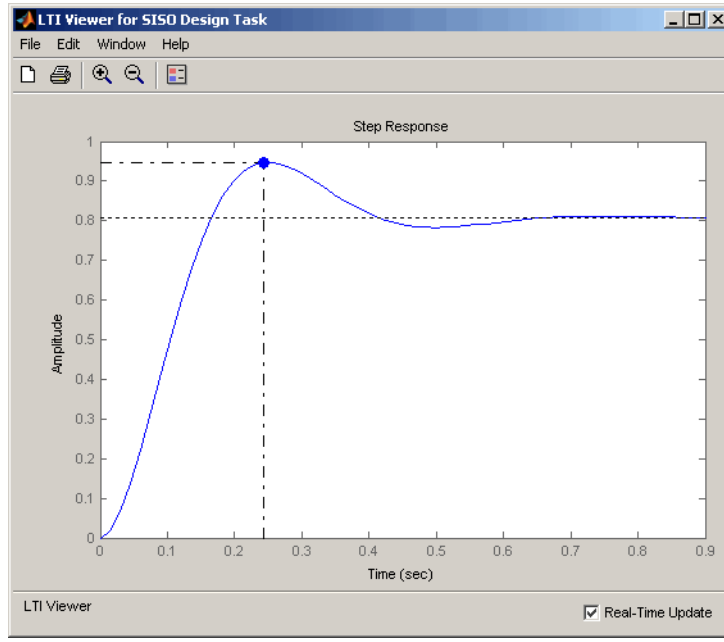
In this example, the new gain is 112.

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.

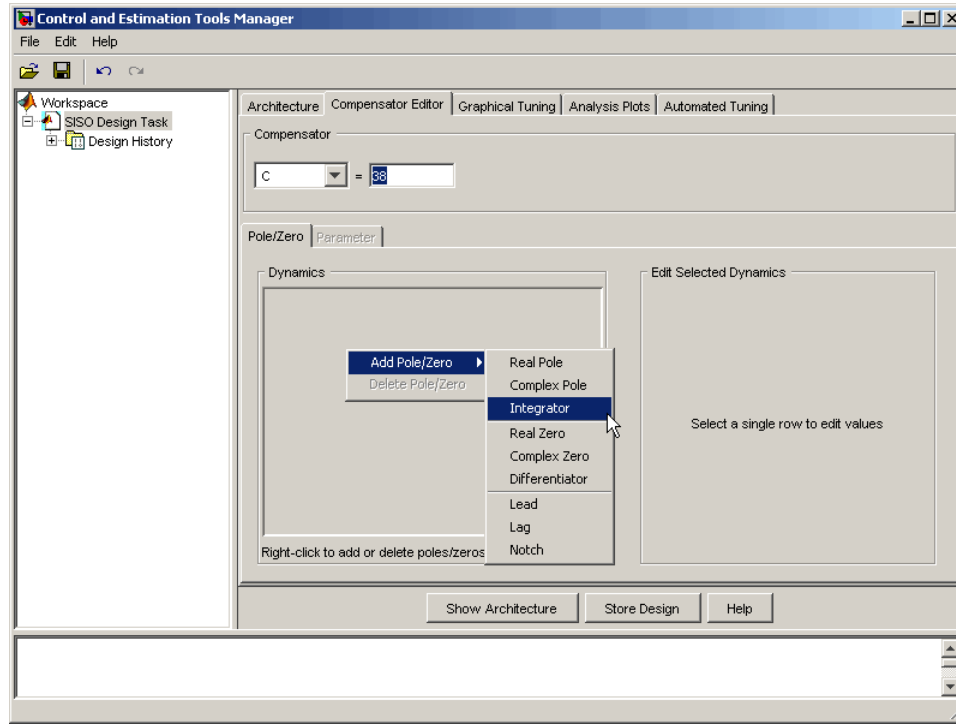
Adding an Integrator

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.

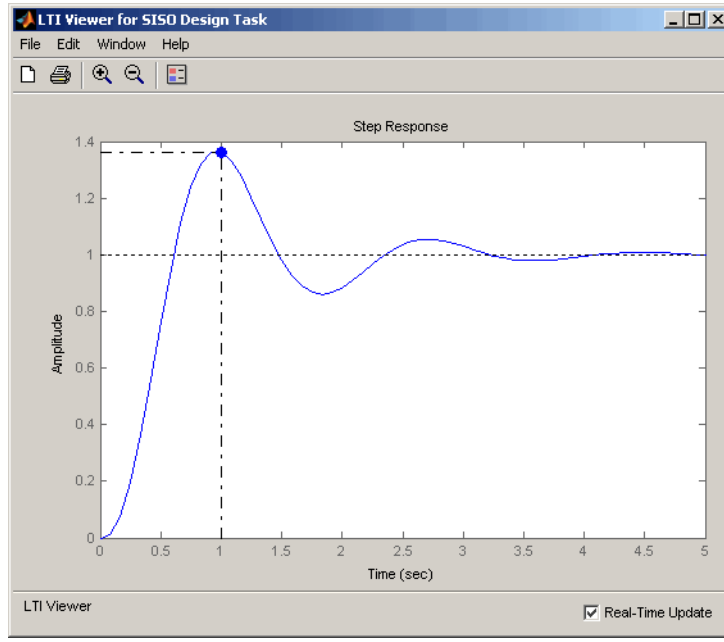


Adding an Integrator in the Dynamics Table

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.

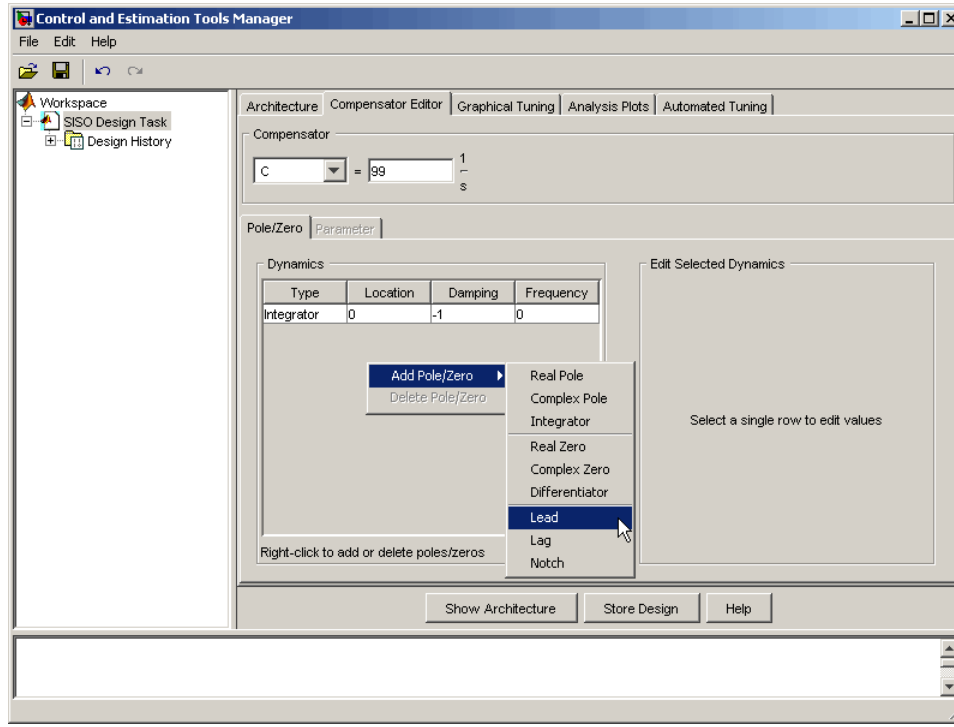
Adding a Lead Network

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**.

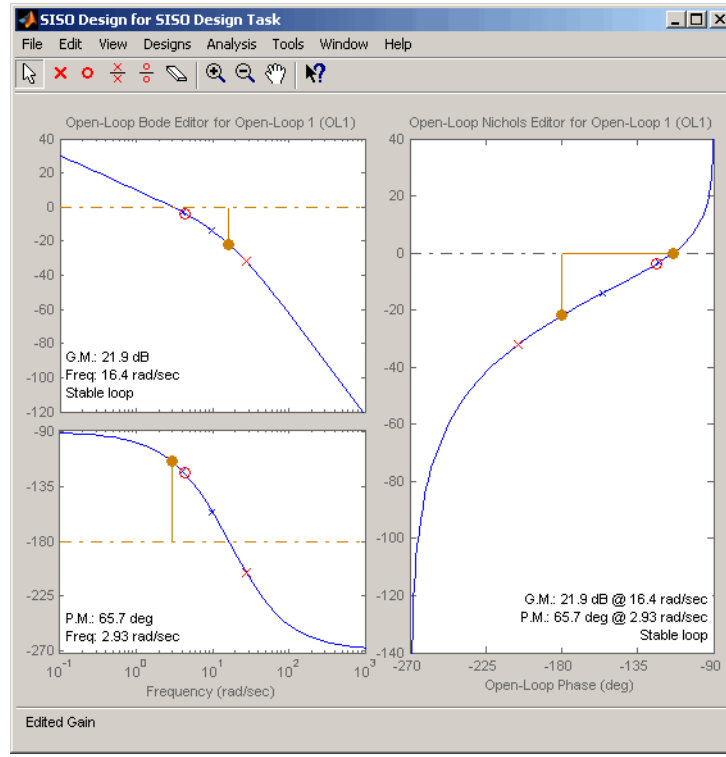
This figure shows the process of adding a lead network on the **Compensator Editor** page.



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.

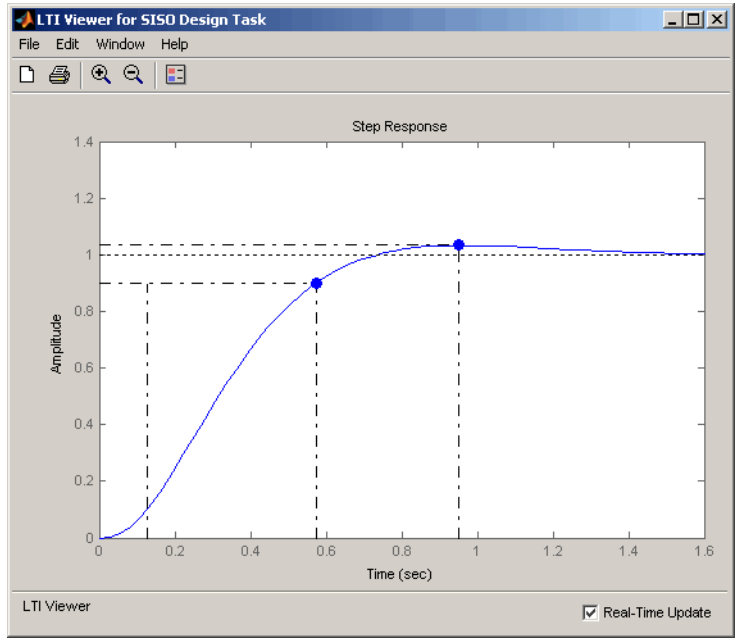
You can track the pole's movement in the status bar at the bottom of the Graphical Tuning window. The status bar tells you the current location of the pole.

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 4-113

"Loading and Displaying the DC Motor Example for Automated Tuning" on page 4-113

"Applying Automated PID Tuning" on page 4-115

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 tuning 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" in the *Control System ToolboxUser's Guide*.

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

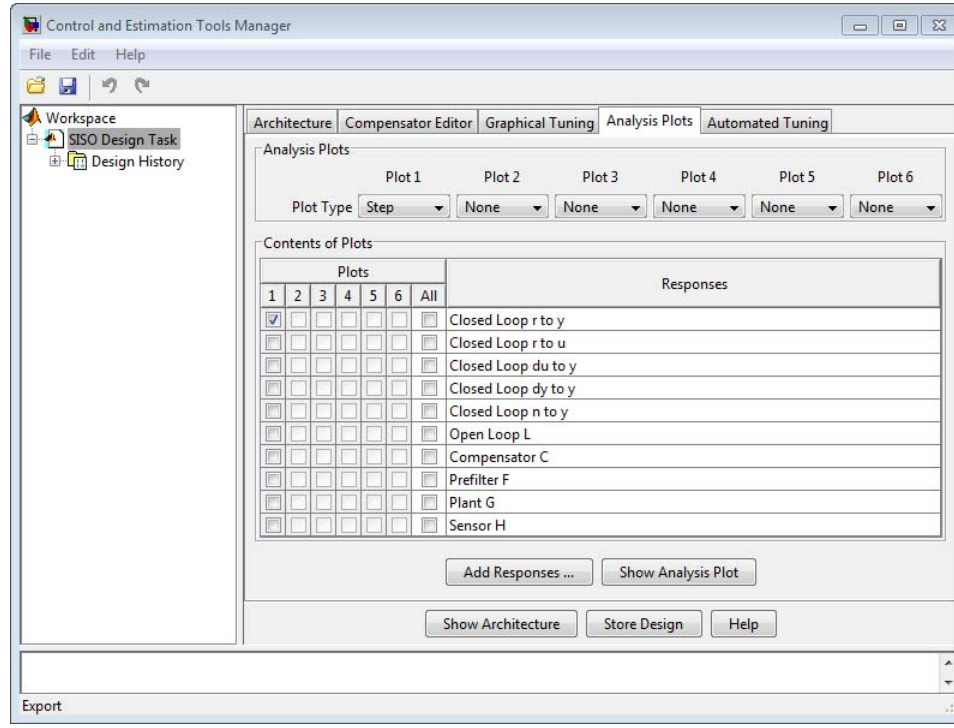
```
load ltiexamples
```

2 To open the SISO Design Tool and import the DC motor, type

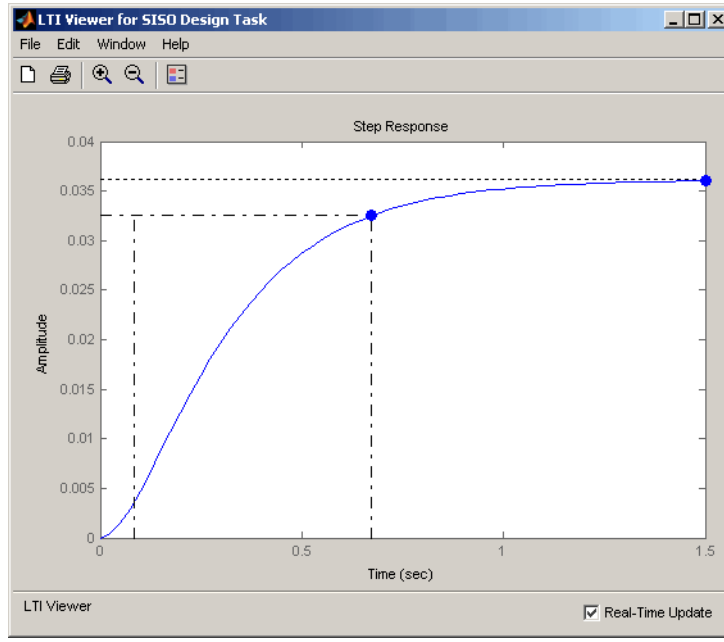
```
sisotool(sys_dc)
```

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.

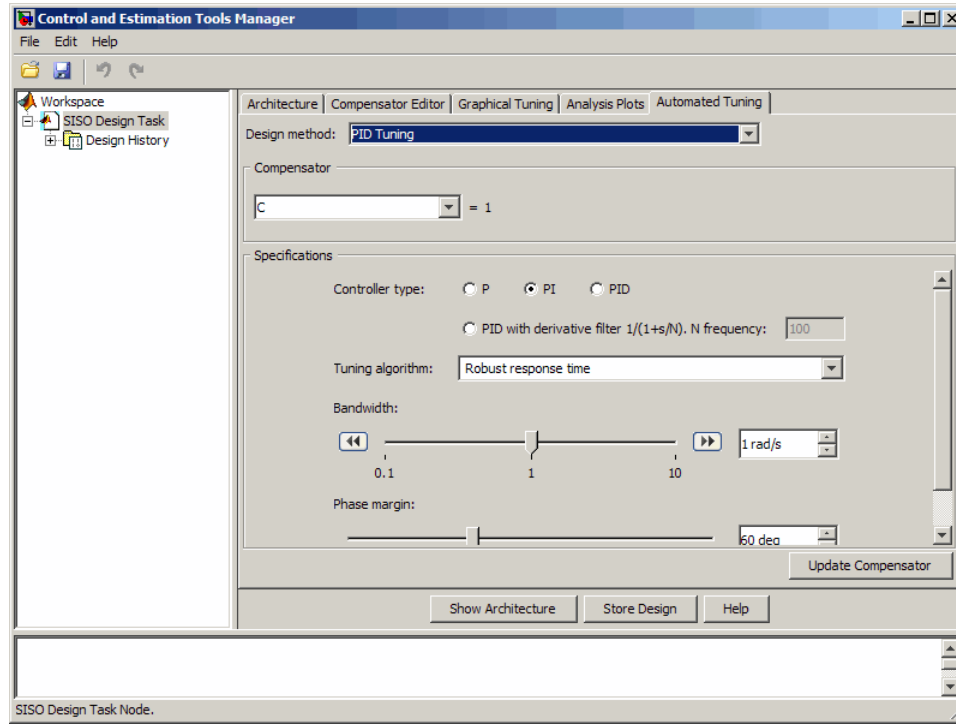


Step Response When Compensator = 1

Note that by default, the compensator is 1 and unit negative feedback is used (see Architecture in the *Control System Toolbox User's Guide*). 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.



You can choose from the following controller types:

- `P`
- `PI`
- `PID`
- `PID` with derivative filter $\frac{1}{(1+s/N)}$

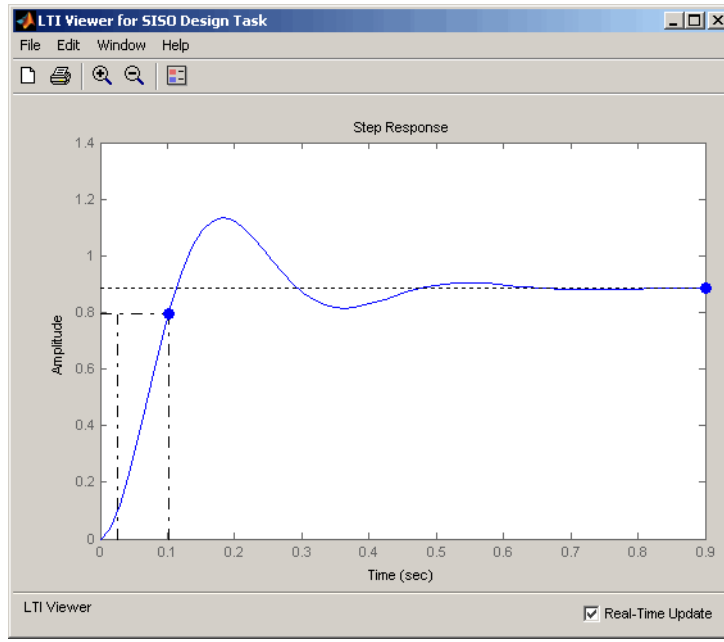
Designing a Proportional-Only Controller

1 For the controller type, click the **P** option button for proportional-only control ($C = K_p$).

2 Select the `Ziegler-Nichols open loop` tuning algorithm from the **Tuning algorithm** list.

3 Click the **Set-Point Tracking** option button.

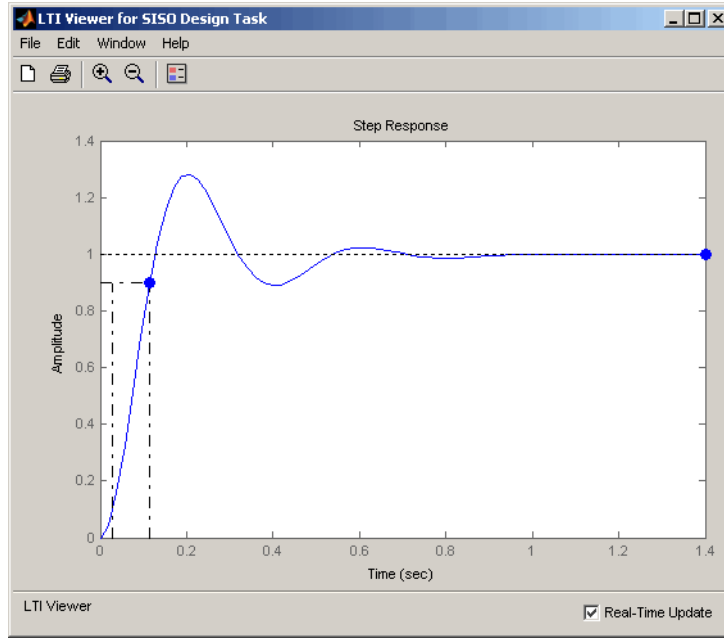
4 Click **Update Compensator**. The LTI Viewer is updated with the application of PID automated tuning with P and the compensator value is now 205.49.



Note that the rise time is arrived at in just 0.0774 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`, as described in "Designing a Proportional-Integral Controller" on page 4-118.

Designing a Proportional-Integral Controller

1. For the controller type, click the **PI** option button for proportional-integral control ($C = K_p + \frac{K_I}{s}$).
2. Select the `Ziegler-Nichols open loop` tuning algorithm from the **Tuning algorithm** list.
3. Click the **Set-Point Tracking** option button.
4. Click **Update Compensator**. The LTI Viewer is updated with the application of PID automated tuning with PI and the compensator value is now $487.64 \times \frac{(1+0.36s)}{s}$.



Although the rise time has increased slightly (0.0876), the steady state value is 1. Applying automated tuning using PID tuning set to `PI` will guarantee zero offset.

Multi-Loop Compensator Design

In this section...

"When to Use Multi-Loop Compensator Design" on page 4-119

"Workflow for Multi-Loop Compensator Design" on page 4-119

"Example: Position Control of a DC Motor" on page 4-119

When to Use Multi-Loop Compensator Design

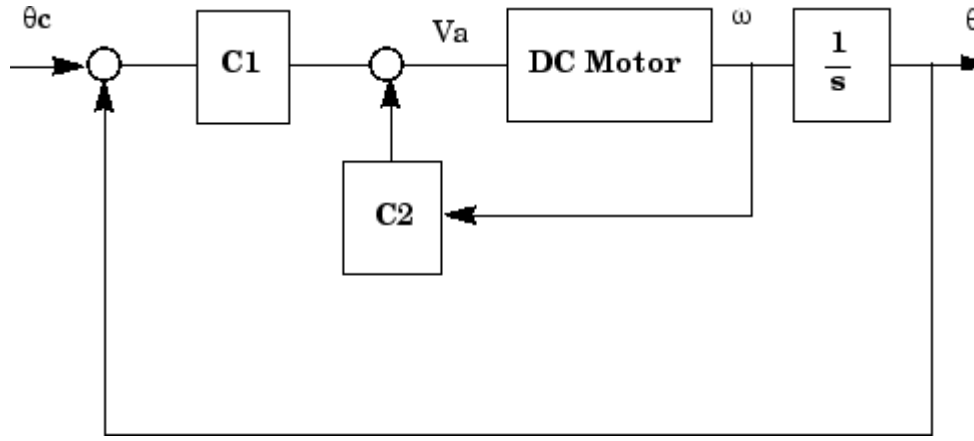
In many applications, a single-loop design is not feasible. If you have a design with inner loops, you can use the SISO Design Tool to design a compensator that meets your specifications.

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. "Example: Position Control of a DC Motor" on page 4-119 shows an example of this procedure.

Example: 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 65° 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 4-120
- "Selecting the Architecture and Importing the Model" on page 4-122
- "Designing the Inner Loop" on page 4-125
- "Tuning the Outer Loop" on page 4-127
- "Validating the Design with the LTI Viewer for SISO Design" on page 4-129

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
```

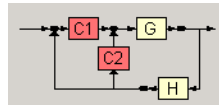
```
Kf= 0.2;          % Nms
J = 0.02          % kg.m^2/s^2
```

First, construct a state-space model of the DC motor with one input, the applied voltage (`Va`). The output is the angular rate `w`.

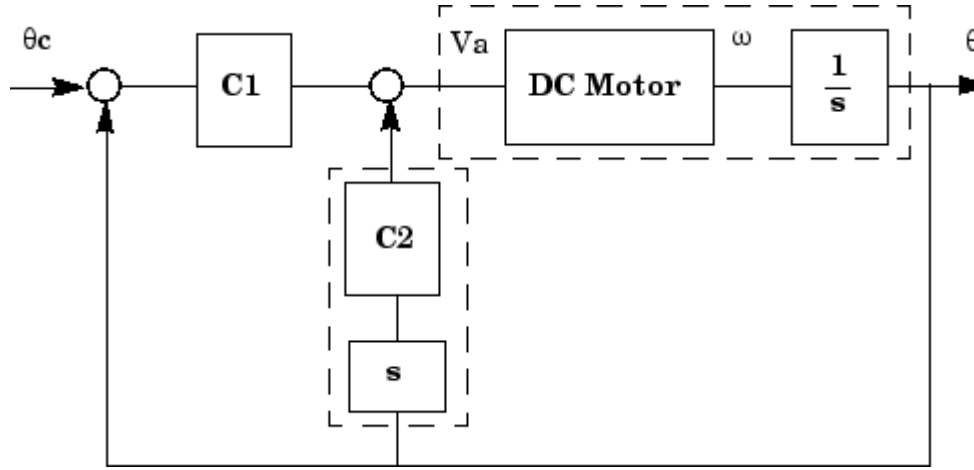
```
h1 = tf(Km,[L,R]); % Armature
h2 = tf(1,[J, Kf]) % Equation of motion
dcm = ss(h2) *h1; % w = h2 cascaded with h1
dcm = feedback(dcm, Kb, 1, 1);% Closes back emf loop
```

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 on page 4-120, 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.

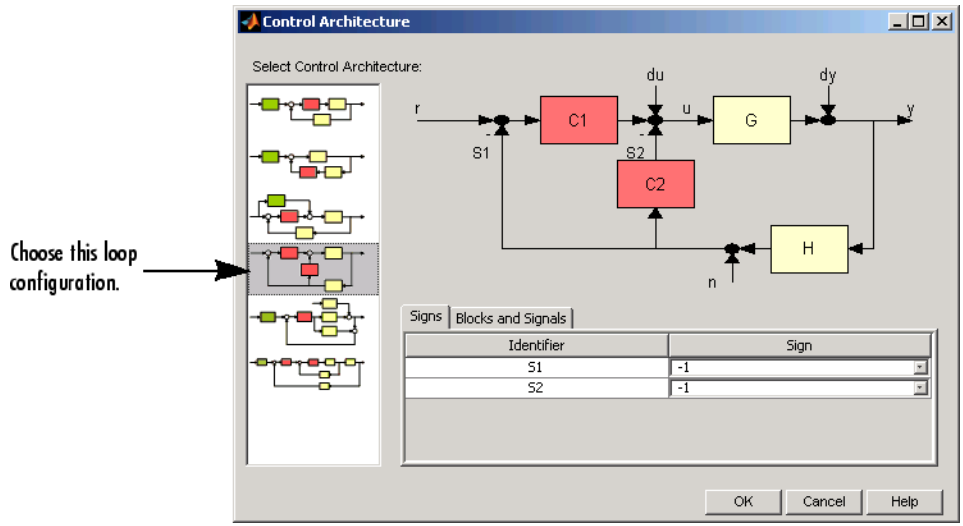
```
G = dcm*tf(1,[1,0]) % Motor with integrator; output is theta.
C2 = tf('s') % Differentiator
```

Selecting the Architecture and Importing the Model

Open the SISO Design Tool by typing

```
sisotool
```

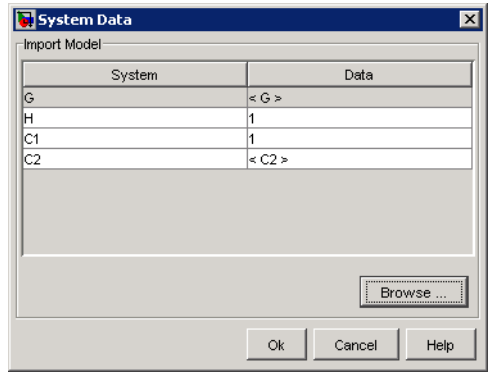
at the MATLAB prompt. Once the Controls & Estimation Tools Manager opens, click **Control Architecture** on the **Architecture** page. Select the multi-loop configuration with two compensators, **C1** in the forward path and C2 in the inner feedback loop — located in the lower-right corner.



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.



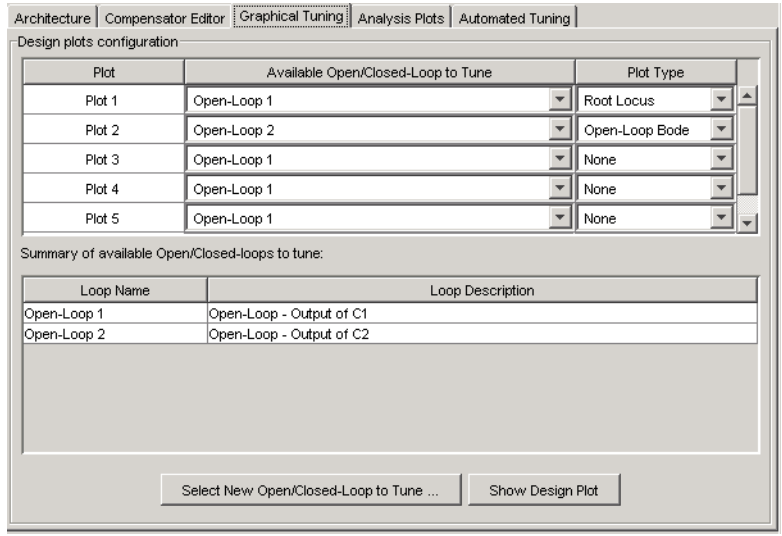
Selecting SISO Design Graphical Editor Views

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"

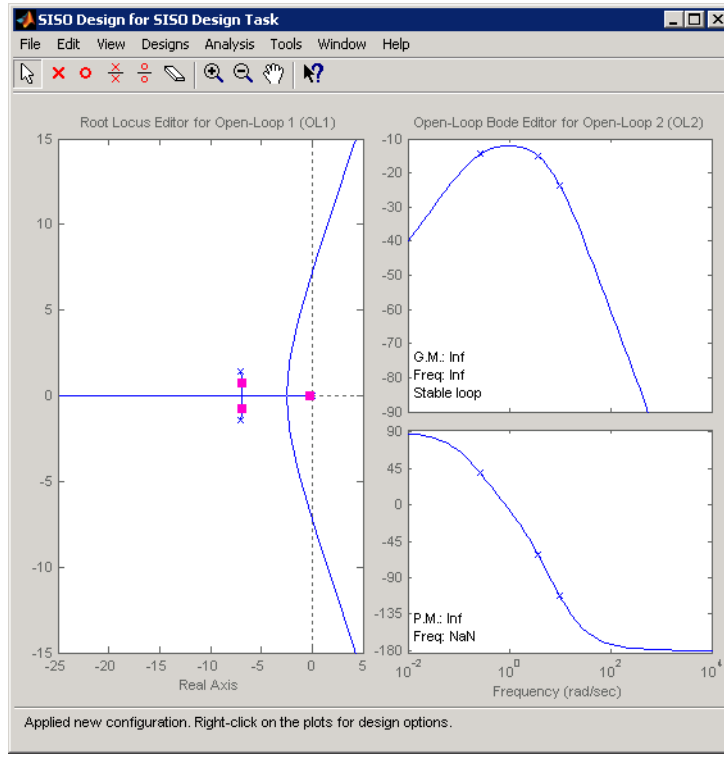
2 Open-Loop 2 to "Open-Loop Bode"

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.



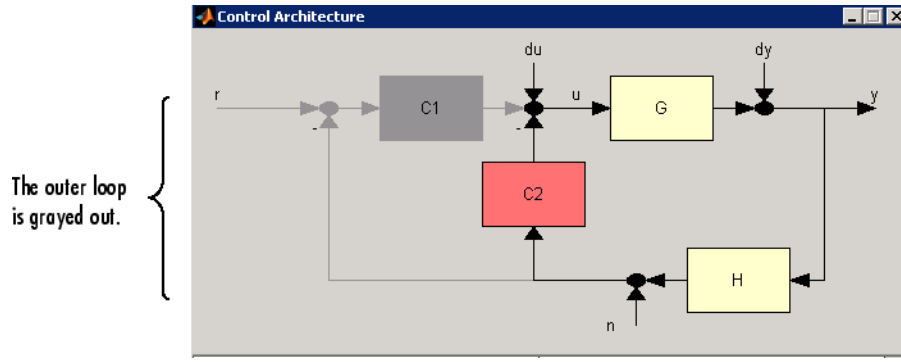
Designing the Inner Loop

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.

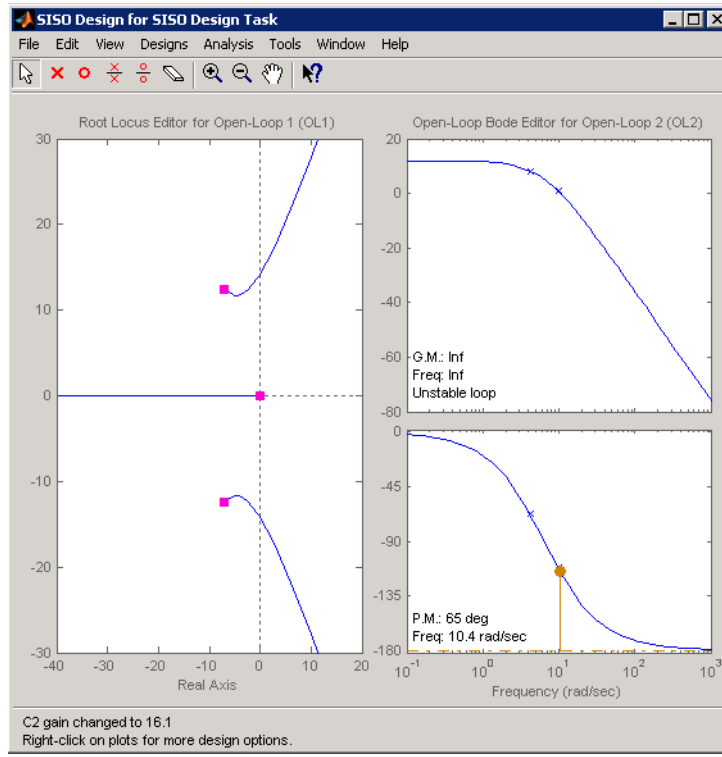
2. From the pull-down menu, select **Open-Loop Output of C2**.

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.

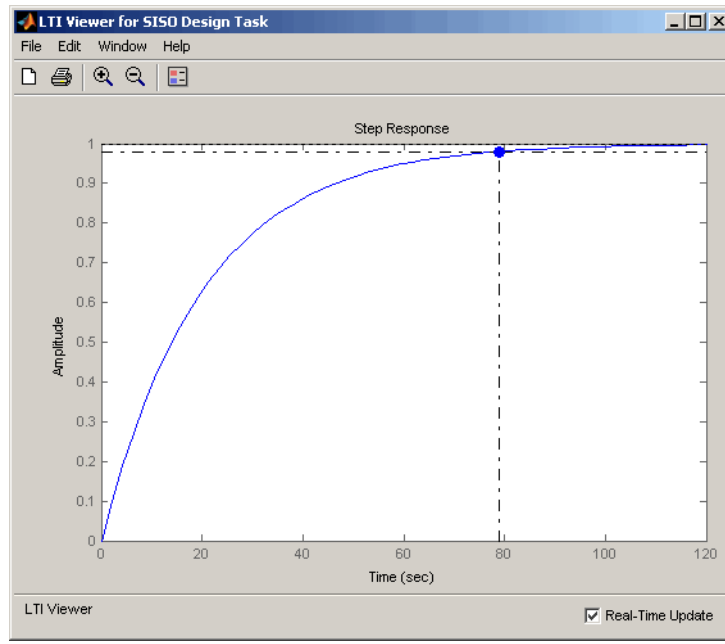
Tuning the Outer Loop

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.

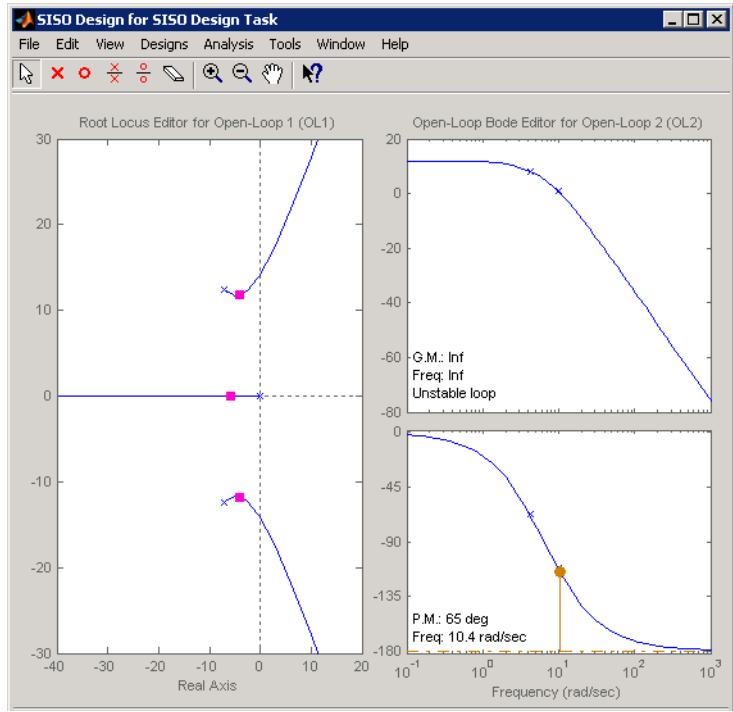
3 Right-click in the step response plot and select **Characteristics>Settling Time**. Your LTI Viewer should look like this.



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.

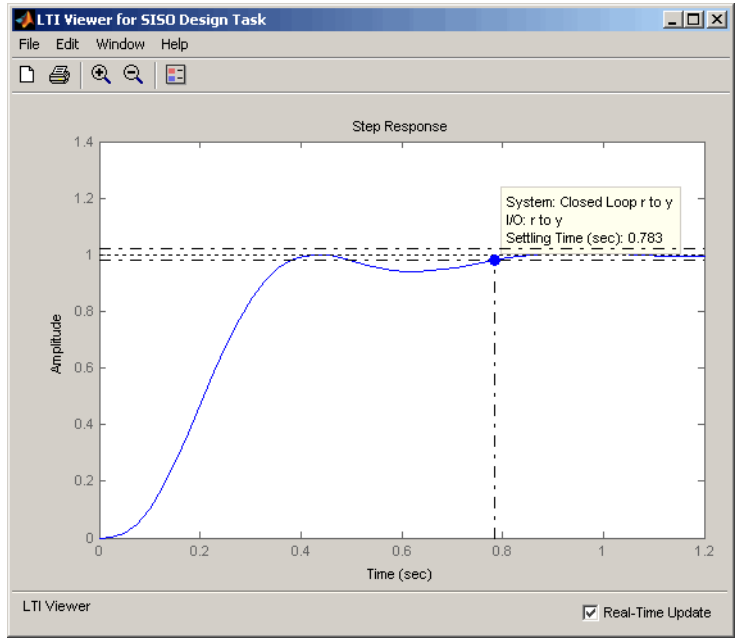
The 90.2 gain seems to yield a good compromise between rise and settling time.



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 4-131

"Control Design Analysis Using the SISO Design Tool" on page 4-131

"Specifying a Nominal Model" on page 4-132

"Frequency Grid for Multimodel Computations" on page 4-135

"How to Analyze the Controller Design for Multiple Models" on page 4-136

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 4-136.

Specifying a Nominal Model

- "What Is a Nominal Model?" on page 4-132
- "How Is a Nominal Model Computed?" on page 4-132
- "Criteria for Choosing a Nominal Model" on page 4-134

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?

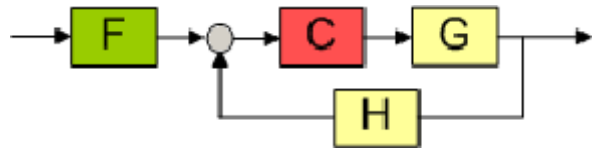
The following table summarizes how the software computes the nominal model when the plant G and sensor H are arrays of LTI models in a control architecture.

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.

Only *G* or *H*	Uses scalar expansion for the specified index and *G* or *H* value.

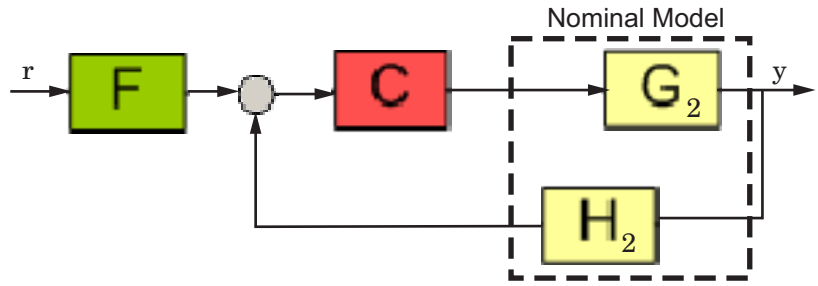
For example, in the following default control architecture:



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:



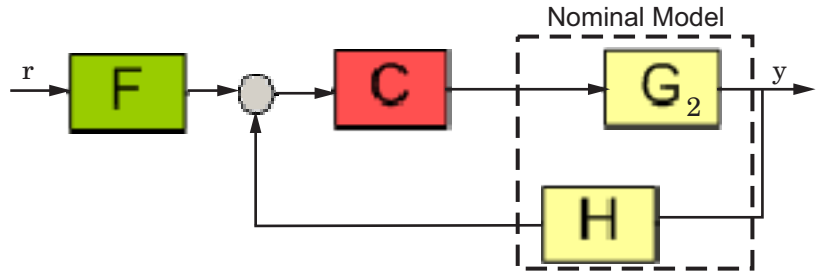
The nominal response from r to y is:

$$T = \frac{CG_{nom}}{1 + CG_{nom}H_{nom}},$$

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 4-136 for more information.

Frequency Grid for Multimodel Computations

- "Algorithm for Frequency Grid Computation" on page 4-135
- "When to Specify Custom Frequency Grid" on page 4-135

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 4-136.

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:

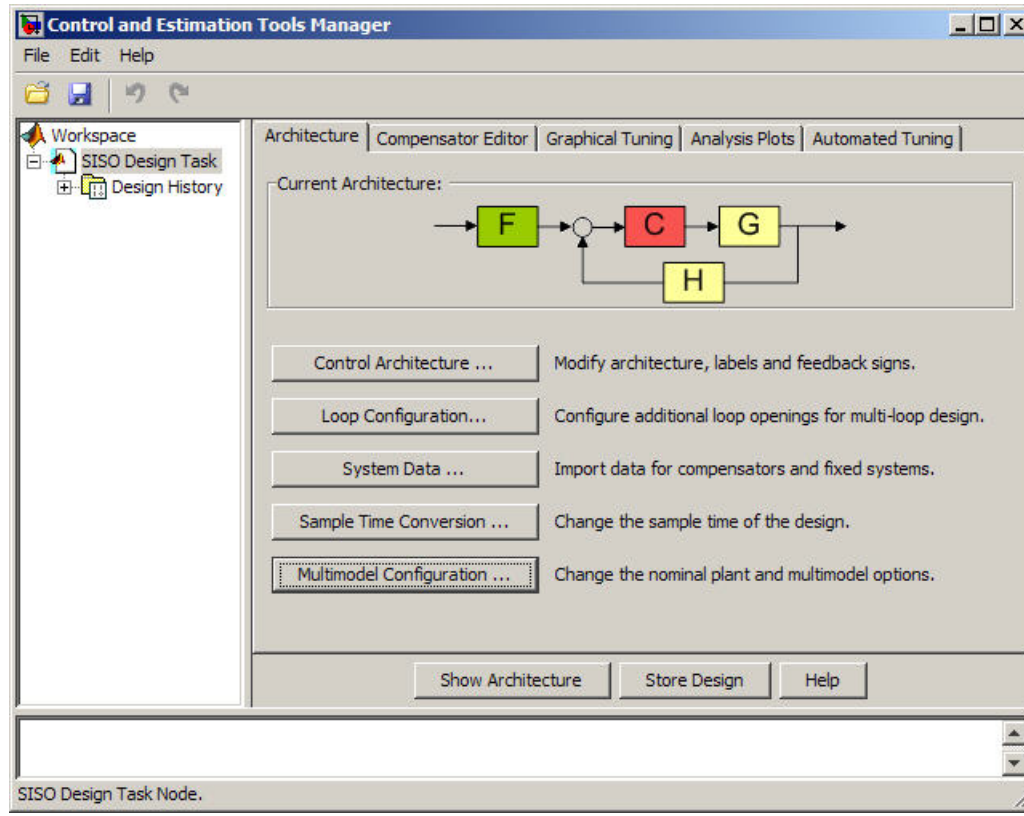
```
% m = 3, b = 0.5, k = 8:1:10.
% Create an array of LTI models to model plant (G) variations.
G1 = tf(1, [1 1 8]);
G2 = tf(1, [1 1 9]);
G3 = tf(1, [1 1 10]);
% Create an array of LTI models
G = stack(1, G1, G2, G3);

 % 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 Reference Tracking of a DC Motor with Parameter Variations demo.
- (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 Working with Models Containing Core Simulink or Custom Blocks.
- (Requires Robust Control Toolbox software) Compute uncertain state-space (`uss`) models from Simulink models. hen use `usubs` or `usample` to create an array of LTI models. See Working with Models Containing Uncertain State Space Blocks.

1 Open SISO Design Tool.

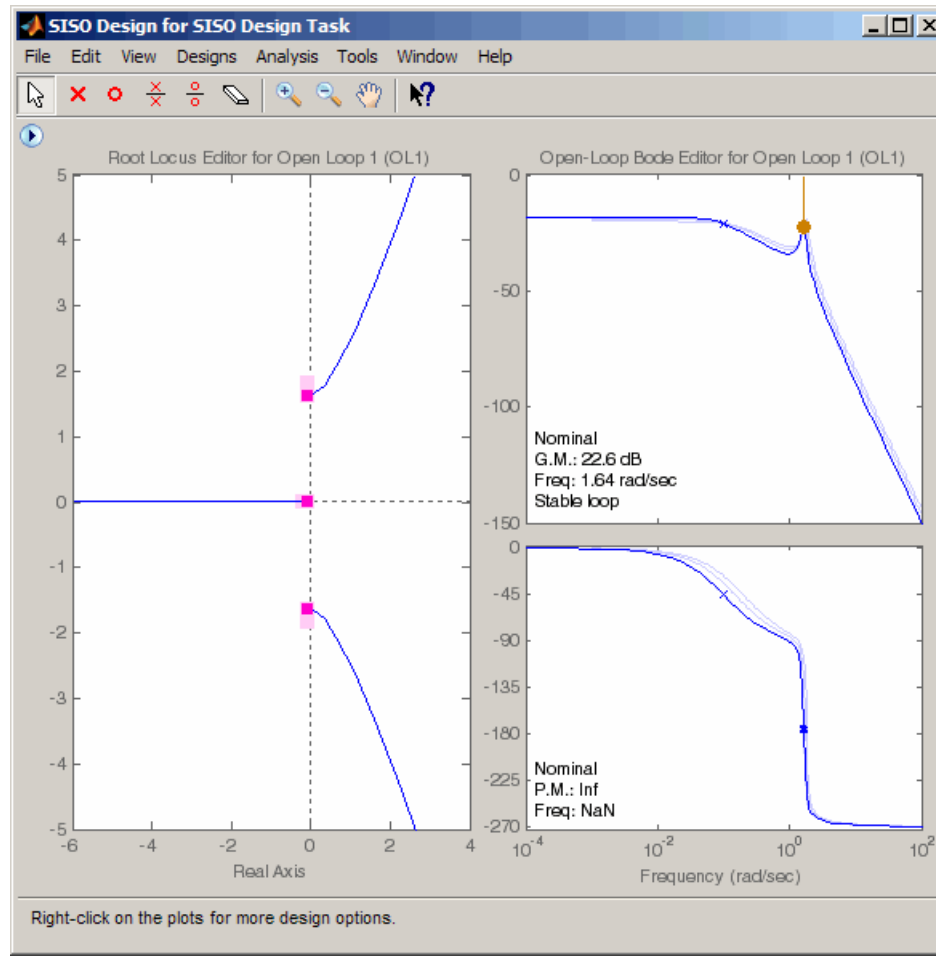
```
sisotool(G,1,H)
```

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 4-132.

The Graphical Tuning window, which opens with the SISO Design Tool, shows the individual responses of all models in the arrays.



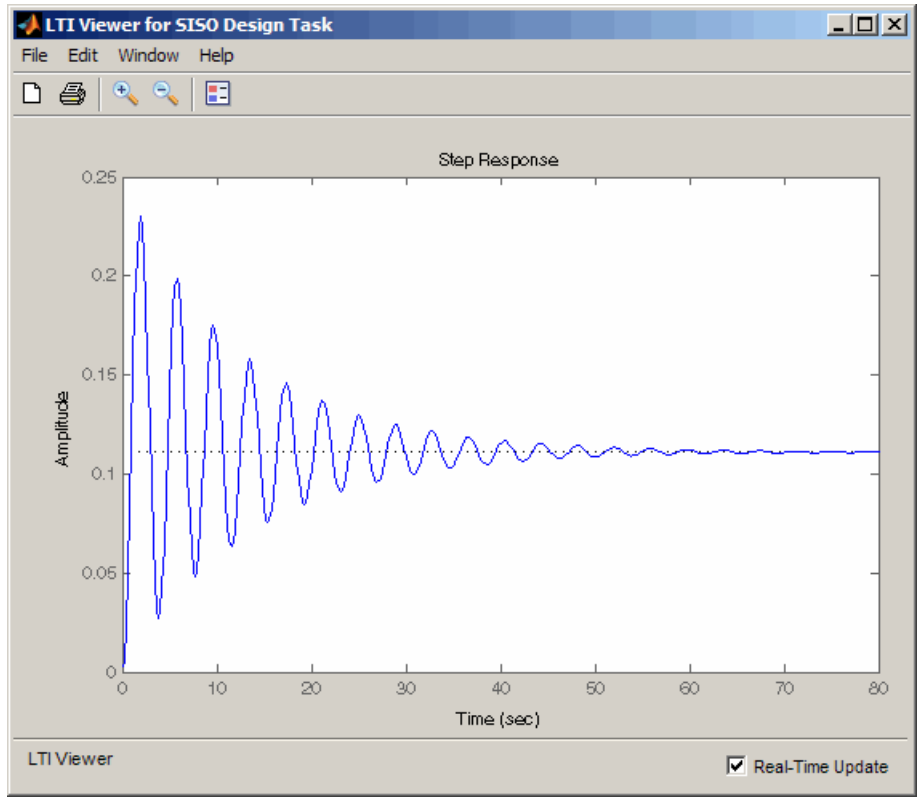
Tip You can view the envelope of the Bode response instead by right-clicking the plot and selecting **Multimodel Display > Bounds**. See "Using the Graphical Tuning Window" on page 4-41 for more information.

Alternatively, to view the responses of all models in the arrays:

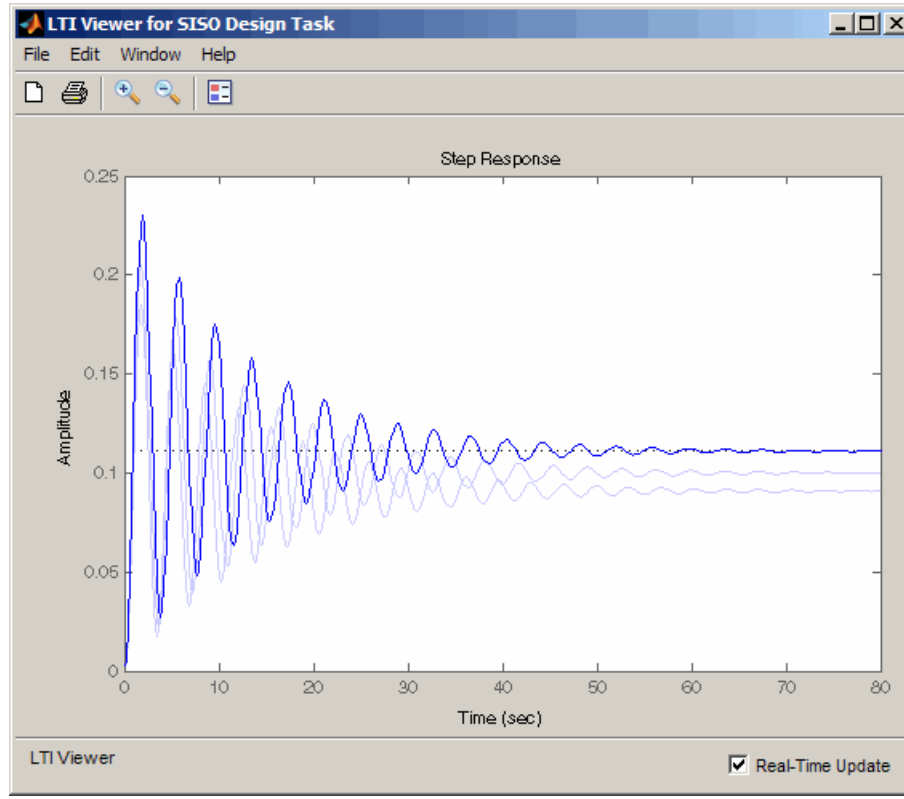
- Open an empty SISO Design Tool by typing `sisotool` and import G and H. See "Importing Models into the SISO Design Tool" on page 4-29 for more information.
- Configure SISO Design Tool using `sisoinit` with the arrays of LTI models and then open the SISO Design Tool.

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.



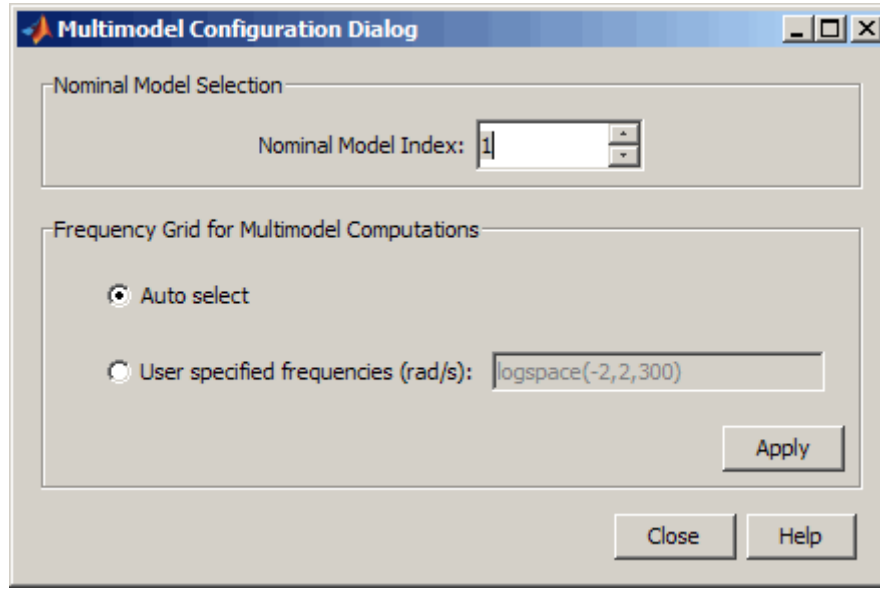
Right-click the plot, and select **Multimodel Configuration > Bounds** or **Multimodel Configuration > Individual Responses** to see the individual response or envelope of all models, respectively.



See “Analysis Plots for Loop Responses” on page 4-35 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 4-135.

5. Design the controller for the nominal plant using graphical or automated tuning.

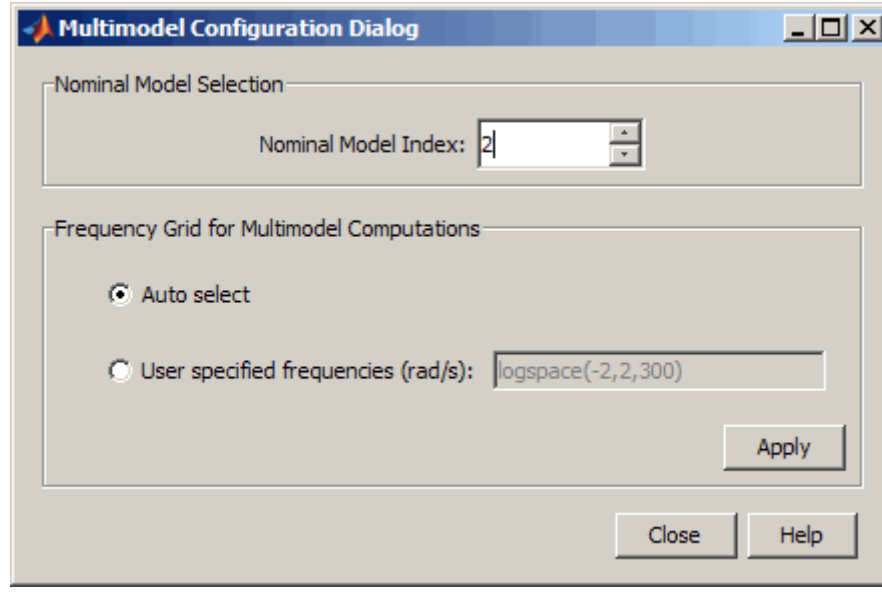
For more information on designing a controller, see the following topics:

- “Bode Diagram Design” on page 4-54
- “Root Locus Design” on page 4-80
- “Nichols Plot Design” on page 4-102
- “Automated Tuning Design” on page 4-113

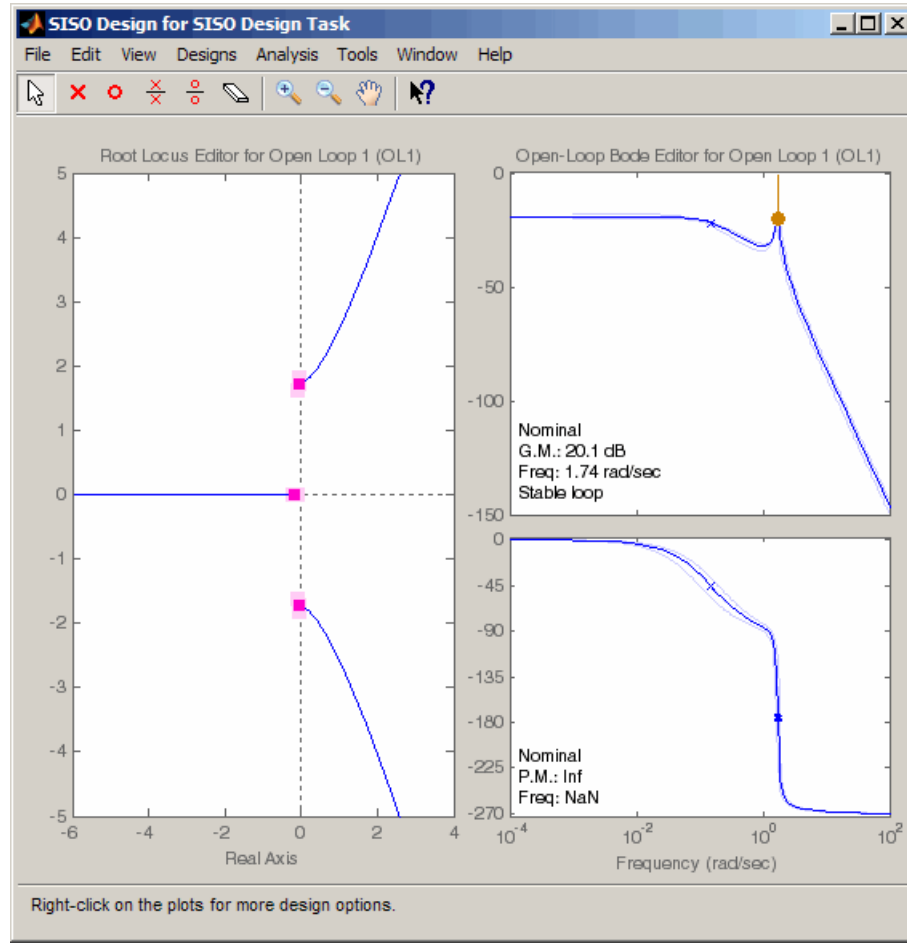
As you design the controller, use the design and analysis plots analyze to analyze the controller’s effects on the remaining models in the array.

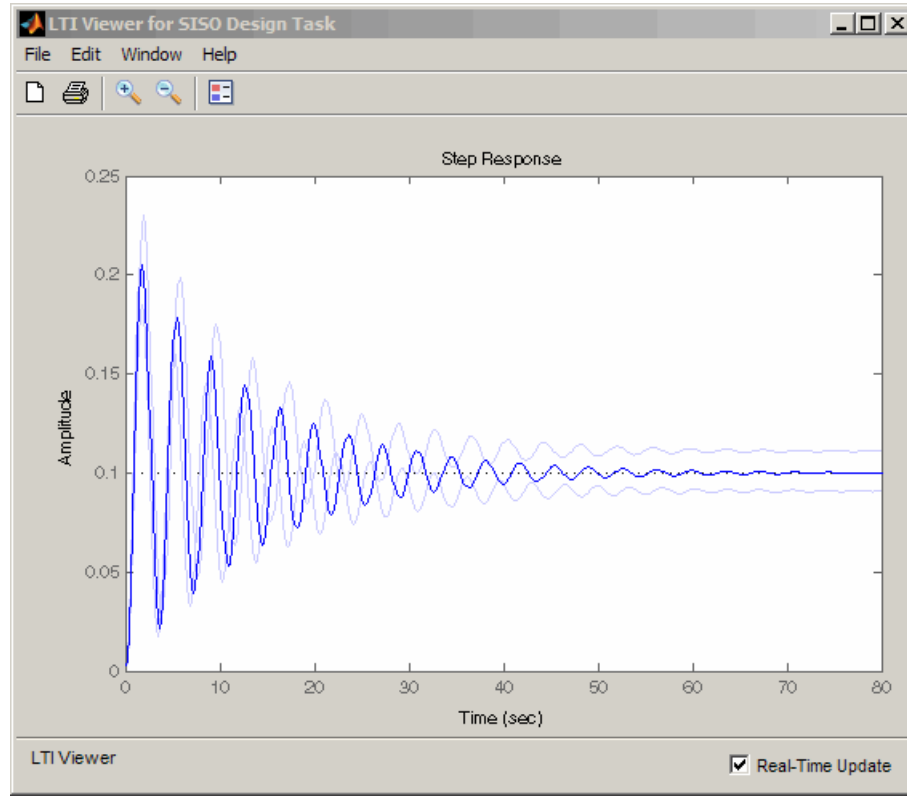
6 (Optional) If the controller design for the nominal model does not meet the design requirements for the remaining models in the array:

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.


SISO Design for SISO Design Task
File Edit View Designs Analysis Tools Window Help
Root Locus Editor for Open Loop 1 (OL1)
Real Axis
Open-Loop Bode Editor for Open Loop 1 (OL1)
Nominal
G.M.: 20.1 dB
Freq: 1.74 rad/sec
Stable loop
Nominal
P.M.: Inf
Freq: NaN
Frequency (rad/sec)
Right-click on the plots for more design options.



b Redesign the controller iteratively.

Functions for Compensator Design

In this section...
"When to Use Functions for Compensator Design" on page 4-146
"Root Locus Design" on page 4-146
"Pole Placement" on page 4-147
"Linear-Quadratic-Gaussian (LQG) Design" on page 4-150
"Example — Designing an LQG Regulator" on page 4-161
"Example — Designing an LQG Servo Controller" on page 4-164
"Example — Designing an LQR Servo Controller in Simulink" on page 4-167

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

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
`acker`	SISO pole placement
`estim`	Form state estimator given estimator gain
`place`	MIMO pole placement
`reg`	Form output-feedback compensator given state-feedback and estimator gains

The function `acker` is limited to SISO systems, and you should use it only for systems with a small number of states. The function `place` is a more general and numerically robust alternative to `acker`.

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.

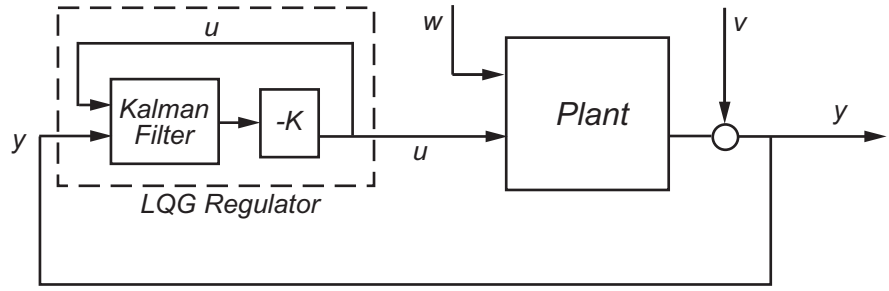
For more information about using LQG design to create LQG regulators , see "Linear-Quadratic-Gaussian (LQG) Design for Regulation" on page 4-151.

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 4-156.

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. • 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 $\hat{x}[n \mid n-1]$.	`lqr`, `kalman`, and `lqgreg` For more information, see • "Constructing the Optimal State-Feedback Gain for Regulation" on page 4-153 • "Constructing the Kalman State Estimator" on page 4-153 • "Forming the LQG Regulator" on page 4-155

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:

$$J(u) = \int_0^\infty \{x^T Qx + 2x^T Nu + u^T Ru\}dt$$

The software computes the gain matrix K by solving an algebraic Riccati equation.

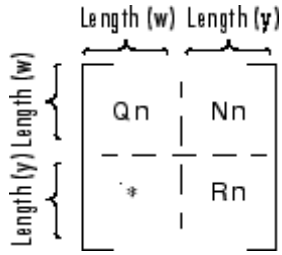
For information about constructing LQ-optimal gain, including the cost function that the software minimizes for discrete time, see the `lqr` reference page.

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

$$\lim_{t\to\infty} E\left((x-\hat{x})(x-\hat{x})^T\right)$$

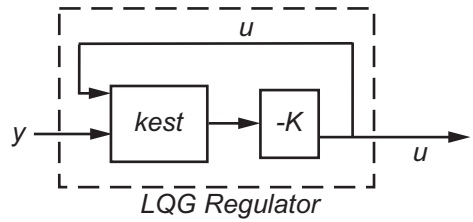
of the estimation error $x-\hat{x}$.

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.



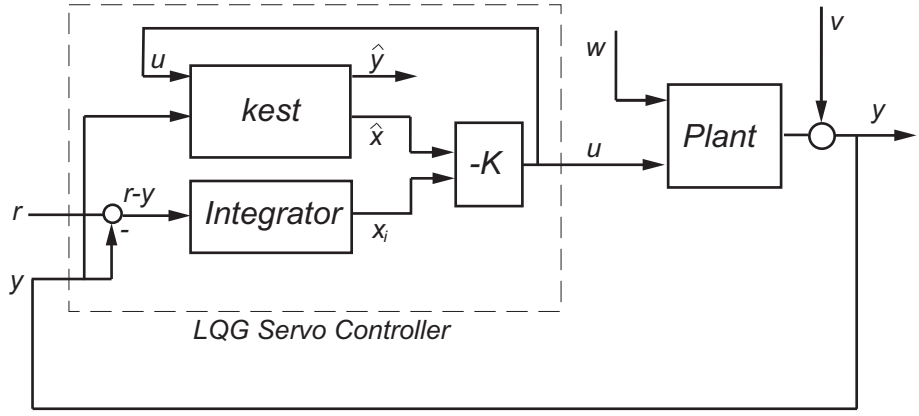
The regulator has the following state-space equations:

$$\frac{d}{dt}\hat{x} = [A - LC - (B - LD)K]\hat{x} + Ly$$
$$u = -K\hat{x}$$

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:



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 4-157 • "Constructing the Kalman State Estimator" on page 4-158 • "Forming the LQG Servo Control" on page 4-160

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.

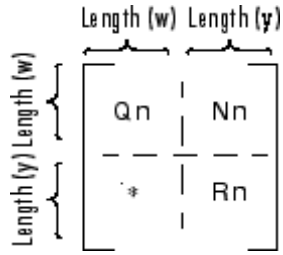
For information about constructing LQ-optimal gain, including the cost function that the software minimizes for discrete time, see the `lqi` reference page.

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

$$\lim_{t\to\infty} E\left((x-\hat{x})(x-\hat{x})^T\right)$$

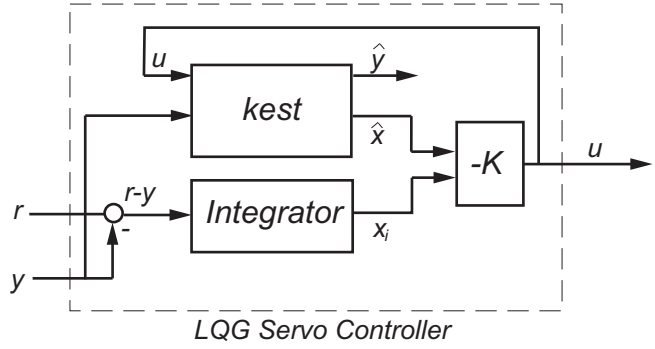
of the estimation error $x-\hat{x}$.

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.



The servo controller has the following state-space equations:

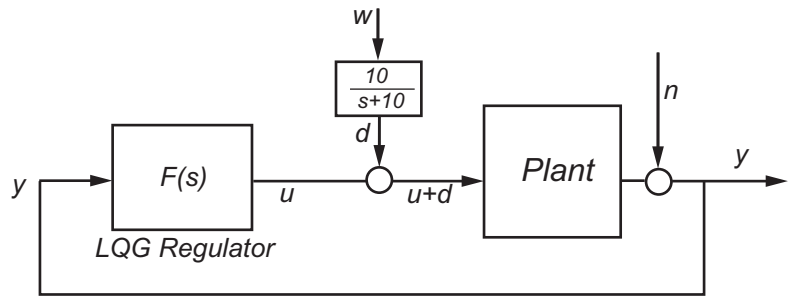
$$\begin{bmatrix}\dot{\hat{x}}\\ \dot{x}_i\end{bmatrix}=\begin{bmatrix}A-BK_x-LC+LDK_x & -BK_i+LDK_i\\ 0 & 0\end{bmatrix}\begin{bmatrix}\hat{x}\\ x_i\end{bmatrix}+\begin{bmatrix}0 & L\\ I & -I\end{bmatrix}\begin{bmatrix}r\\ y\end{bmatrix}$$

$$u=\begin{bmatrix}-K_x & -K_i\end{bmatrix}\begin{bmatrix}\hat{x}\\ x_i\end{bmatrix}$$

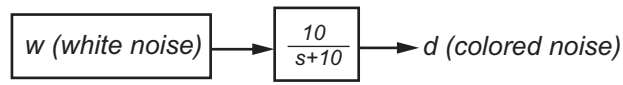
For more information on forming LQG servo controllers, including how to form a one-degree-of-freedom LQG servo controller, see the `lqgtrack` reference page.

Example — Designing an LQG Regulator

As an example of LQG design, consider the following regulation problem.

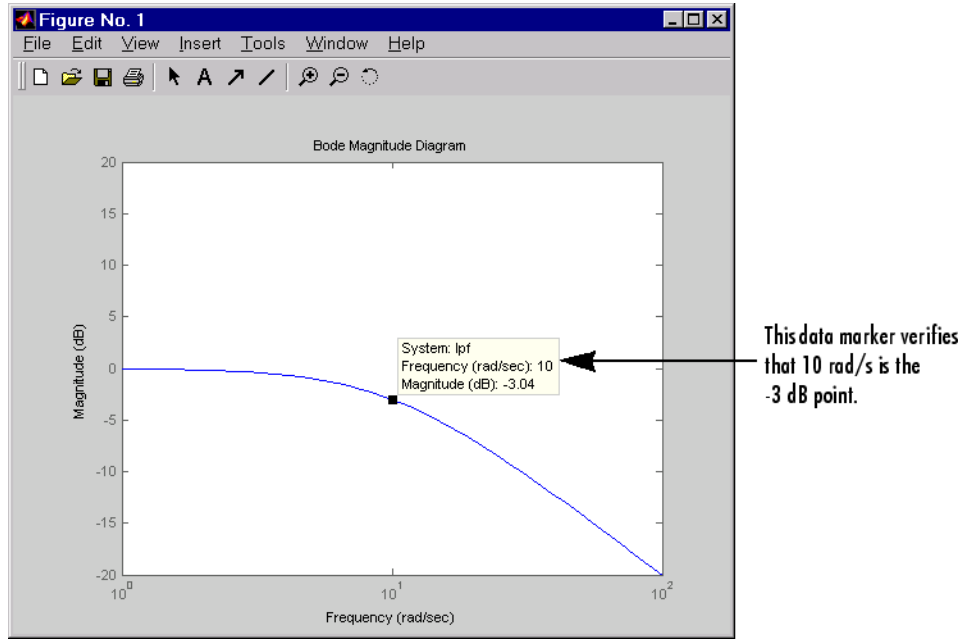


The goal is to regulate the plant output y around zero. The input disturbance d is low frequency with power spectral density (PSD) concentrated below 10 rad/s. For LQG design purposes, it is modeled as white noise driving a lowpass filter with a cutoff at 10 rad/s, shown in the following figure.



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:

$$\dot{x} = Ax + Bu + Bd \quad (state\,equations)$$
$$y = Cx + n \quad (measurements)$$

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:

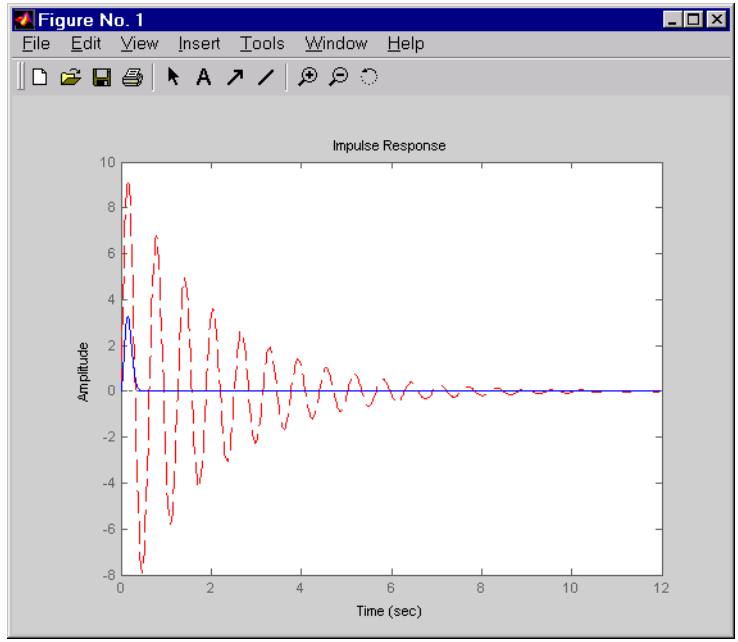
```
sys = ss(tf(100,[1 1 100])) % State-space plant model

% 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-')
```

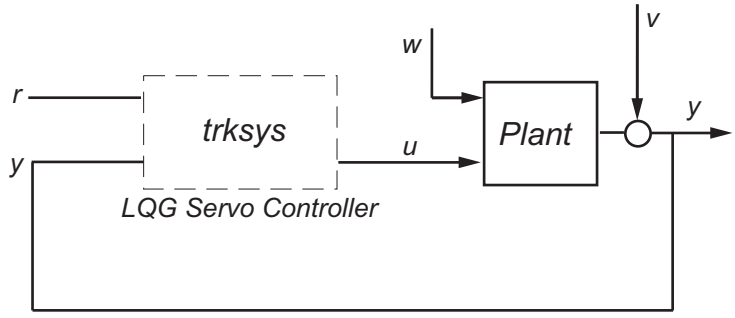
These commands produce the following figure, which compares the open- and closed-loop impulse responses for this example.



Comparison of Open- and Closed-Loop Impulse Response

Example — Designing 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:

$$J(u) = \int_0^\infty \left(0.1x^T x + x_i^2 + u^T \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} u \right) dt$$

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)
```

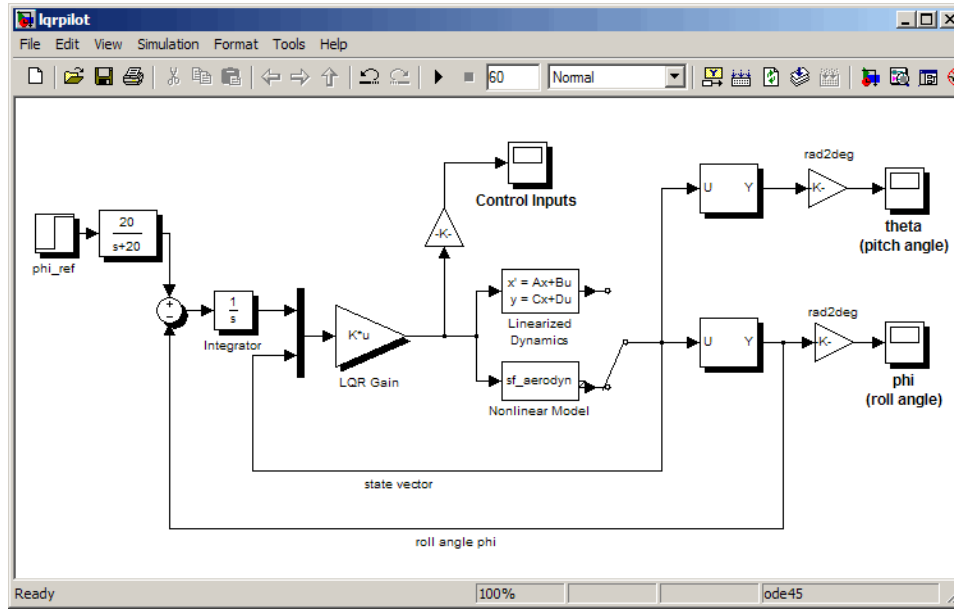
This command returns the following LQG servo controller:

```
>> 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.
```

Example — Designing 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 $(\theta,\phi) = (0,15^\circ)$.
- 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

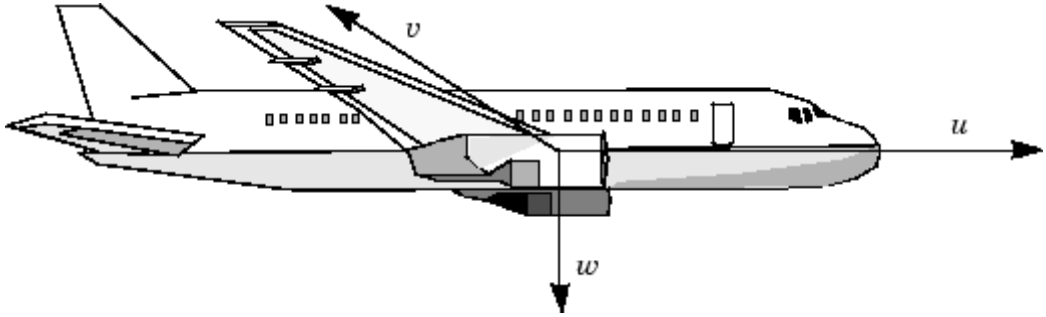
Beginning with the standard state-space equation

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

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.

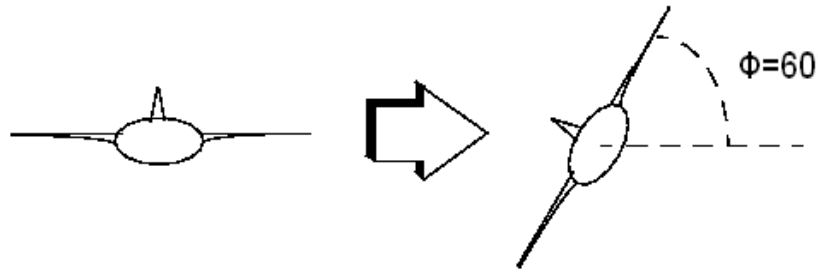
Trimming

For LQG design purposes, the nonlinear dynamics are trimmed at $\phi = 15^\circ$ 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 $(\theta,\phi) = (0,15^\circ)$ 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.

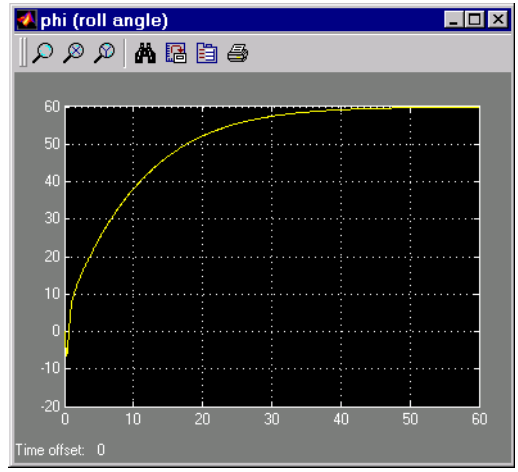
Results

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.

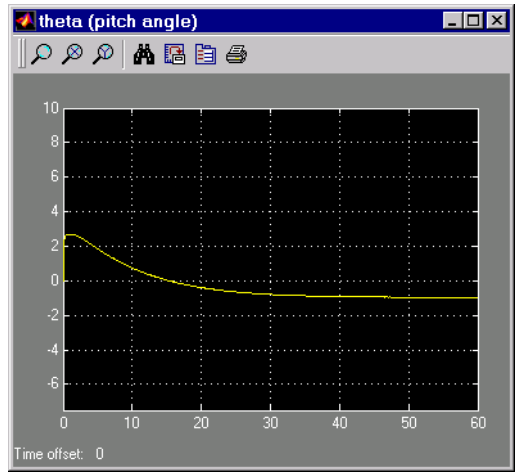
This figure shows the response of ϕ to the 60° step command.



Tracking the Roll Step Command

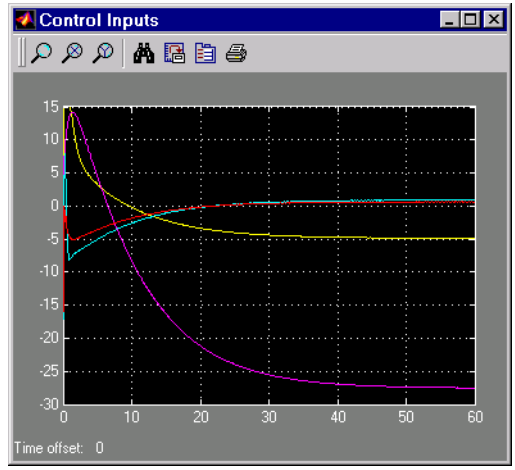
As you can see, the system tracks the commanded 60° roll in about 60 seconds.

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.

Index

A

automated tuning design 4-113

B

building models
  MIMO 2-13
  SISO 2-6

C

cell array 2-16
changing
  units on a plot 4-46
comparing
  models in plots 3-44
compensator design
  multi-loop 4-119
compensators
  editing in SISO Design Tool 4-95
  exporting 4-48
  feedback (state estimator design) 4-148
concatenation
  model 2-16
control design analysis of multiple models 4-131
Control System Toolbox product
  related products 1-4
covariance
  noise 4-154 4-159

D

data markers 3-42
denominator
  specification 2-16
design
  Kalman estimator 4-153
  LQG Tracker, example of 4-167
  LQG, description 4-150
  pole placement 4-147
  regulators 4-155
  root locus 4-146
  state estimator 4-148
  state estimator, Kalman 4-153

E

estimator
  gain 4-148
  Kalman 4-153
exporting compensators and models 4-48

F

final time. *See also* time response 3-40
frequency response 3-41
  customizing plots 3-47
  plotting 3-44
functions
  time and frequency response 3-37

G

gain
  estimator, Kalman 4-154 4-159
  selection 4-81
  state-feedback 4-153
Graphical Tuning window 4-25
  changing units on a plot 4-46
  display 4-42
  right-click menus 4-47
  using 4-41

I

input vector 2-2

K

Kalman estimator
  continuous 4-153

gain 4-154 4-159
steady-state 4-153
Kalman filter. *See* Kalman estimator

L

linear models
comparing multiple models in plots 3-44
exporting 4-48
frequency response. 3-41
model order reduction 2-29
time response 3-37
LQG (linear quadratic Gaussian) method
continuous LQ regulator 4-153
design. *See* design, LQG
gain, optimal state-feedback 4-153
Kalman state estimator 4-153
LQ-optimal gain 4-153
regulator, designing 4-163
regulator, forming 4-155
weighting matrices 4-153 4-158
LTI Viewer
adding plot types 3-16
changing models 3-20
changing plot types 3-14
command line initializing 3-18
comparing multiple models 3-17
importing models 3-9
introduction to 3-8
models, importing 3-9
opening 3-9
plot options
rise time 3-11
settling time 3-19
right-click menus 3-10
SISO Design Task 4-25

M

markers, data 3-42
MIMO 3-40
model order reduction 2-29
models
building 2-13
multi-loop design 4-119

N

noise
covariance 4-154 4-159
white 3-38
notch filter
parameters 4-73
numerator
specification 2-16

O

operations on LTI models
concatenation 2-16
output vector 2-2
overshoot 3-37

P

plotting
changing units 4-46
customizing plots 3-47
multiple systems 3-44
right-click menus 3-41
time responses 3-37
pole placement 4-147
conditioning 4-150

R

reduced-order models 2-29
regulation 4-151 4-162
performance 4-153
Riccati equation 4-153
right-click menus

LTI Viewer 3-10
plot responses 3-41
rise time 3-37
root locus
compensator gain 4-87
designing 4-146
root locus design 4-80
functions for 4-146

S

Servo controller with integral action 4-156
Set point tracking 4-156
settling time 3-37
SISO Design Task in the Control and Estimation Tools Manager 4-25
SISO Design Task pane 4-25
Analysis Plots page 4-35
Automated Tuning page 4-115
Design History page 4-51
using 4-28
SISO Design Tool 4-25
adding a lead network 4-62
adding a lead network to a Nichols plot 4-109
adding a notch filter 4-70
adding an integrator 4-58
adding an integrator to the Nichols plot 4-107
adding poles and zeros 4-90
adjusting compensator gain in a Nichols plot 4-104
adjusting the compensator gain in Bode design 4-55
applying automated PID tuning 4-115
automated tuning design 4-113
Bode Diagram Design 4-54
changing the compensator gain, Bode magnitude 4-55
changing the compensator gain, root locus 4-87
compensators, editing 4-95
editing preferences 4-46
exporting compensator and models 4-48
feedback structures 4-33
importing models into 4-29
loop responses 4-35
Nichols plot design 4-102
opening 4-26
opening a Nichols plot 4-103
root locus design 4-80
storing and retrieving designs 4-50
Using the Graphical Tuning window 4-41
SISO Tool Preferences Editor 4-46
state
estimator 4-148
vector 2-2
state estimator design
regulators/compensators 4-148
state-space models 2-2
steady state error 3-37

T

time response
customizing plots 3-47
final time 3-40
functions 3-37
MIMO 3-40
plotting 3-44
time range 3-40
to white noise 3-38
vector of time samples 3-41
transfer functions
MIMO 2-15
transfer functions (TF) 2-2

U

units
changing on a plot 4-46

W

white noise 3-38
See also noise

Z

zero-pole-gain (ZPK) models 2-3