

Control System Toolbox

For Use with MATLAB®

- Computation
- Visualization
- Programming

Getting Started

Version 7



How to Contact The MathWorks:



www.mathworks.com
comp.soft-sys.matlab
www.mathworks.com/contact_TS.html

Web
Newsgroup
Technical support



suggest@mathworks.com
bugs@mathworks.com
doc@mathworks.com
service@mathworks.com
info@mathworks.com

Product enhancement suggestions
Bug reports
Documentation error reports
Order status, license renewals, passcodes
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.

Getting Started with the Control System Toolbox

© COPYRIGHT 2000-2006 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, Simulink, Stateflow, Handle Graphics, Real-Time Workshop, and xPC TargetBox are registered trademarks of The MathWorks, Inc. Other product or brand names are trademarks or registered trademarks of their respective holders.

Patents

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

Revision History

November 2000	First printing	New for 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)
July 2005	Third printing	Revised for Version 6.2 (Release 14SP2)
September 2005	Online only	Minor revision for Version 6.2.1 (Release 14SP3)
March 2006	Online only	Revised for Version 7.0 (R2006a)
September 2006	Online only	Revised for Version 7.1 (R2006b)

What Is the Control System Toolbox?

1

Introduction	1-2
Installation	1-3
Demos	1-4
Using the Documentation	1-5

Building Models

2

Linear Models	2-2
Linear Model Representations	2-2
SISO Example: The DC Motor	2-3
Constructing SISO Models	2-5
Discrete Time Systems	2-9
Adding Delays to Linear Models	2-10
LTI Objects	2-10
MIMO Models	2-12
MIMO Example: Jet Transport Aircraft	2-12
Constructing MIMO Transfer Functions	2-14
Accessing I/O Pairs in MIMO Systems	2-16
Arrays of Linear Models	2-17
Model Characteristics	2-20
Interconnecting Linear Models	2-21
Feedback Interconnection	2-22

Continuous/Discrete Conversions	2-23
Discrete DC Motor Model	2-23
Model Order Reduction	2-25
Example: Gasifier Model	2-25

Analyzing Models

3

LTI Viewer	3-2
Example: Time and Frequency Responses of the DC Motor	3-3
Right-Click Menus	3-5
Displaying Response Characteristics on a Plot	3-7
Changing Plot Type	3-10
Showing Multiple Response Types	3-12
Comparing Multiple Models	3-13
Simulating Models with Arbitrary Inputs and Initial Conditions	3-17
Opening the Linear Simulation Tool	3-17
Working with the Linear Simulation Tool	3-18
Importing Input Signals	3-20
Example: Loading Inputs from Excel	3-23
Example: Importing Inputs from the Workspace	3-23
Designing Input Signals	3-29
Specifying Initial Conditions	3-32
Functions for Time and Frequency Response	3-33
Time and Frequency Responses	3-33
Plotting MIMO Model Responses	3-35
Data Markers	3-38
Plotting and Comparing Multiple Systems	3-40
Creating Custom Plots	3-42

SISO Design Tool	4-2
Opening the SISO Design Tool	4-2
Using the SISO Design Task Node on the Control and Estimation Tools Manager	4-4
Importing Models into the SISO Design Tool	4-5
Feedback Structure	4-8
Loop Responses	4-10
Using the Graphical Tuning Window	4-13
Bode Diagram Design	4-17
Example: DC Motor	4-17
Adjusting the Compensator Gain	4-17
Adjusting the Bandwidth	4-18
Adding an Integrator	4-20
Adding a Lead Network	4-24
Moving Compensator Poles and Zeros	4-28
Adding a Notch Filter	4-31
Modifying a Prefilter	4-36
Root Locus Design	4-41
Example: Electrohydraulic Servomechanism	4-42
Changing the Compensator Gain	4-47
Adding Poles and Zeros to the Compensator	4-49
Adding Zeros to the Compensator	4-51
Editing Compensator Pole and Zero Locations	4-53
Viewing Damping Ratios	4-57
Exporting the Compensator and Models	4-59
Storing and Retrieving Intermediate Designs	4-61
Nichols Plot Design	4-65
DC Motor Example	4-65
Opening a Nichols Plot	4-66
Adjusting the Compensator Gain	4-66
Adding an Integrator	4-70
Adding a Lead Network	4-71

Automated Tuning Design	4-75
Loading and Displaying the DC Motor Example for Automated Tuning	4-75
Applying Automated PID Tuning	4-77
Multi-Loop Compensator Design	4-81
Example: Position Control of a DC Motor	4-81
Functions for Compensator Design	4-92
Root Locus Design	4-92
Pole Placement	4-92
Linear-Quadratic-Gaussian (LQG) Design	4-95
Example: LQG Design	4-100
Example: LQG Design for Set Point Tracking	4-103

Learning More

5

Locating Demos	5-2
Online Help	5-2
Setting Plot Preferences and Properties	5-2
The MathWorks Online	5-3

Index

What Is the Control System Toolbox?

Introduction (p. 1-2)

A brief overview of the Control System Toolbox functionality

Installation (p. 1-3)

How to install the Control System Toolbox

Demos (p. 1-4)

Where to find Control System Toolbox demos

Using the Documentation (p. 1-5)

How to find the information you need in this documentation set

Introduction

MATLAB® has a rich collection of functions immediately useful to the control engineer or system theorist. Complex arithmetic, eigenvalues, root-finding, matrix inversion, and fast Fourier transforms are just a few examples of important numerical tools found in MATLAB. More generally, the MATLAB linear algebra, matrix computation, and numerical analysis capabilities provide a reliable foundation for control system engineering as well as many other disciplines.

The Control System Toolbox builds on the foundations of MATLAB to provide functions designed for control engineering. The Control System Toolbox is a collection of algorithms, written mostly as M-files, that implements common control system design, analysis, and modeling techniques. Convenient graphical user interfaces (GUIs) simplify typical control engineering tasks.

Control systems can be modeled as transfer functions, in zero-pole-gain or state-space form, allowing you to use both classical and modern control techniques. You can manipulate both continuous-time and discrete-time systems. Conversions between various model representations are provided. Time responses, frequency responses, and root loci can be computed and graphed. Other functions allow pole placement, optimal control, and estimation. Finally, the Control System Toolbox is open and extensible. You can create custom M-files to suit your particular application.

Installation

Instructions for installing the Control System Toolbox can be found in the MATLAB Installation documentation for your platform. We recommend that you store the files from this toolbox in a subdirectory named `control` under the main `matlab` directory. To determine if the Control System Toolbox is already installed on your system, check for a subdirectory named `control` within the main toolbox directory or folder.

Demos

The Control System Toolbox provides demonstration files that show you how to use the toolbox to perform control design tasks in various settings. To run these demos, type

```
demo
```

at the MATLAB prompt. This opens the **Demos** pane in the Help browser. Select **Toolboxes** and then **Control System** to see a list of available demos. Alternatively, if you have the Help browser open, you can select the **Demos** pane directly and follow the same procedure.

In addition, “Design Case Studies” contains detailed examples of various design problems.

Using the Documentation

If you are a new user, this guide, Getting Started with the Control System Toolbox, is written for you. Specifically, you will learn

- How to build and manipulate linear time-invariant models of dynamical systems
- How to analyze such models and plot their time and frequency responses
- How to design compensators using root locus and pole placement techniques

In addition, this guide discusses model order reduction, linear quadratic Gaussian (LQG) techniques, and presents examples that show how to use these techniques.

This guide is available online under the Control System Toolbox. The rest of the toolbox documentation is also available online; click **Control System Toolbox** to open its product page, which is a roadmap with links to the Control System Toolbox documentation and to PDF versions of the same documentation.

If you are an experienced toolbox user, see the following documentation for detailed discussions of control system design topics:

- “Release Notes” — Details on the latest release
- “Creating and Manipulating Models” — In-depth information on how to create and manipulate linear models and linear time-invariant (LTI) arrays, which are data objects that you can use to store collections of linear models in one variable
- “Customization” — Information about setting plot properties, including how to set preferences that persist from session to session
- “Design Case Studies” — Worked examples, including Kalman filtering and MIMO design
- “Reliable Computations” — Numerical stability and accuracy issues
- “GUI Reference” — Complete descriptions of the LTI Viewer and SISO Design Tool, which are graphical user interfaces (GUIs) that you can use to analyze systems and design single-input/single-output (SISO) compensators

All toolbox users should use “Functions - By Category” for reference information on functions and tools. For functions, reference descriptions include a synopsis of the function’s syntax, as well as a complete explanation of

options and operation. Many reference descriptions also include helpful examples, a description of the function's algorithm, and references to additional reading material. For GUI-based tools, the descriptions include options for invoking the tool.

Building Models

Linear Models (p. 2-2)

How to build linear, time-invariant (LTI) models

MIMO Models (p. 2-12)

How to build multiple input/output (MIMO) LTI models

Arrays of Linear Models (p. 2-17)

How to create and manipulate LTI arrays, MATLAB objects that store multiple linear models in one variable

Model Characteristics (p. 2-20)

How to access and manipulate model properties and characteristics

Interconnecting Linear Models
(p. 2-21)

How to build models by interconnecting models of subsystems

Continuous/Discrete Conversions
(p. 2-23)

How to convert between continuous- and discrete-time systems, specifying sample times, and introducing time delays to your systems

Model Order Reduction (p. 2-25)

Reducing the number of states in a large-scale system, using a gasifier model as an example

Linear Models

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 to build linear models based on these equations.

Linear Model Representations

The Control System Toolbox supports 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,

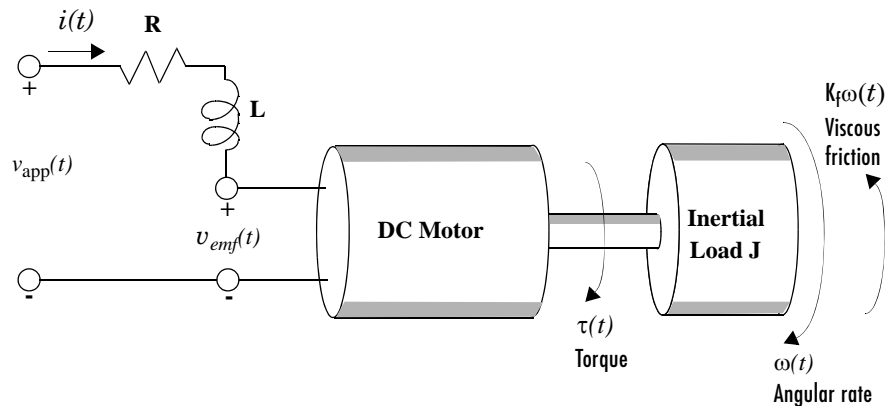
$$H(z) = 3 \frac{(z + 1 + j)(z + 1 - j)}{(z + 0.2)(z + 0.1)}$$

- 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 picture 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. The important thing here is that 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, v_{emf} , is a voltage proportional to the angular rate ω seen at the shaft,

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

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,

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

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

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

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

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

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

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,

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

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, v_{app} , is the input to the system, and the angular velocity ω is the output.

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

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

State-Space Representation of the DC Motor Example

Constructing SISO Models

Once you have a set of differential equations that describe your plant, you can construct SISO models using simple commands in the Control System Toolbox. 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

Listed below are 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)
```

This is the output of the last command.

```
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, 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)
```

The Control System Toolbox builds this transfer function.

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

Alternatively, 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
```

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

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

This is the resulting zero/pole/gain representation.

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

Discrete Time Systems

The Control System Toolbox 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:

$$\frac{1}{z + 1}$$

Sampling time: 0.01

Adding Time Delays to Discrete-Time Models

You can add time delays to discrete-time models by specifying an input or output time 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, 'outputdelay', 5);
```

produces a system with an output delay of 0.05 second.

Transfer function:

$$z^{-5} * \frac{1}{z + 1}$$

Sampling time: 0.01

Adding Delays to Linear Models

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

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

The Control System Toolbox constructs the DC motor transfer function, but adds a 0.05 second delay.

Transfer function:

$$\exp(-0.05s) * \frac{1.5}{s^2 + 14s + 40.02}$$

For a complete description of adding time delays to models, see “Time Delays”.

LTI Objects

For convenience, the Control System Toolbox 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: [] OutputName: {''}
    InputGroup: {0x2 cell}
    OutputGroup: {0x2 cell}
    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. MATLAB 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 “Creating and Manipulating Models”.

MIMO Models

You can use the same functions that apply to SISO systems to create multiple-input, multiple-output (MIMO) models, including arbitrary MIMO transfer functions and zero/pole/gain models. This section begins with an example of how to build a MIMO system. It then discusses how to build MIMO transfer functions by concatenating SISO transfer functions and how to access and manipulate individual SISO transfer functions contained in a MIMO model.

MIMO Example: Jet Transport Aircraft

This example shows how to build a MIMO model of a jet transport. Since 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

$$A = \begin{bmatrix} -0.0558 & -0.9968 & 0.0802 & 0.0415 \\ 0.5980 & -0.1150 & -0.0318 & 0 \\ -3.0500 & 0.3880 & -0.4650 & 0 \\ 0 & 0.0805 & 1.0000 & 0 \end{bmatrix};$$

$$B = \begin{bmatrix} 0.0073 & 0 \\ -0.4750 & 0.0077 \\ 0.1530 & 0.1430 \\ 0 & 0 \end{bmatrix};$$

$$C = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix};$$

$$D = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix};$$

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.

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

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 Interconnection Functions” 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

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

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

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

The Control System Toolbox responds with

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

      s + 2
```

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

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

Accessing I/O Pairs in MIMO Systems

Once you have defined 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 K_b and K_m 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);
```

(Note that K_b and K_m must be equal, so K represents both parameters in the state-space equations.) 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

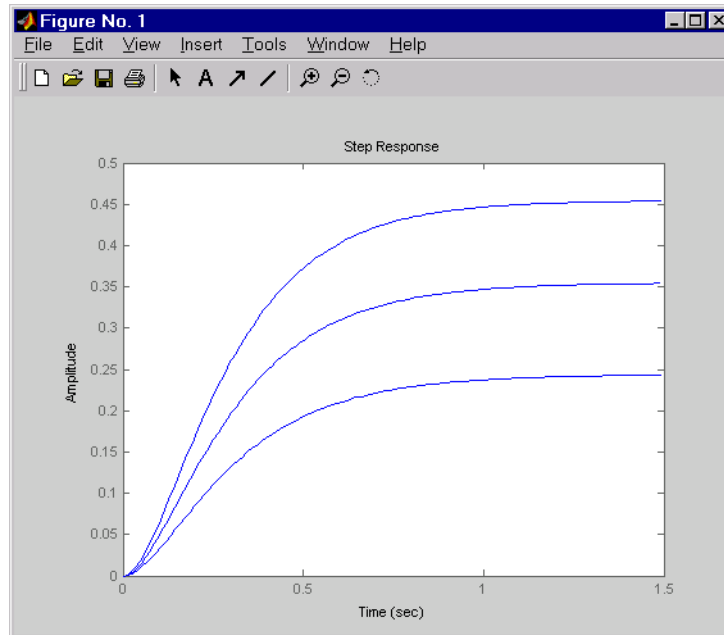
```
.  
. .  
. .
```

3x1 array of continuous-time state-space models.

You can manipulate the LTI array like any other object in the Control System Toolbox. 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 “Arrays of LTI Models”.

Model Characteristics

The Control System Toolbox contains 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
<code>size(model_name)</code>	Number of inputs and outputs
<code>ndims(model_name)</code>	Number of dimensions
<code>isct(model_name)</code>	Returns 1 for continuous systems
<code>isdt(model_name)</code>	Returns 1 for discrete systems
<code>hasdelay(model_name)</code>	True if system has delays
<code>pole(model_name)</code>	System poles
<code>zero(model_name)</code>	System (transmission) zeros
<code>dcgain(model_name)</code>	DC gain
<code>norm(model_name)</code>	System norms (H_2 and L_∞)
<code>covar(model_name,W)</code>	Covariance of response to white noise

Interconnecting Linear Models

You can perform simple operations 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” for more information. State-space models have the highest precedence while transfer functions have the lowest precedence. Hence the sum of a transfer function and a state-space model is always a state-space model.

Other available operations include system inversion, transposition, and pertransposition; see “Arithmetic Operations”. The Control System Toolbox also supports matrix-like indexing for extracting subsystems; see “Extracting and Modifying Subsystems” for more information.

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
<code>sys1 + sys2</code>	<code>parallel(sys1,sys2)</code>	Systems in parallel
<code>sys1 * sys2</code>	<code>series(sys2,sys1)</code>	Cascaded systems

Feedback Interconnection

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

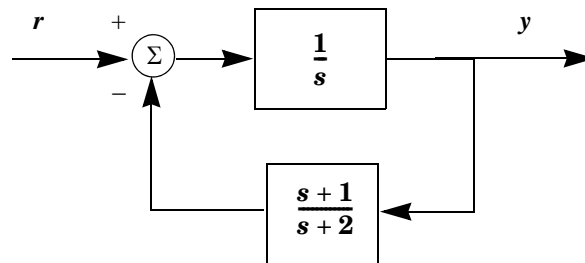
```
sys_f = feedback(tf(1,[1 0]), tf([1 1],[1 2]))
```

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

Transfer function:

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

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.

Continuous/Discrete Conversions

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
```

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
```

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 =  
      y1      x1      x2  
           0      1
```

```
d =  
      y1      u1  
           0
```

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

To see the discrete-time transfer function for the digital DC motor, use `tf` to convert the model.

```
fd=tf(sysd)
```

```
Transfer function:  
7.16e-005 z + 6.833e-005  
-----  
z^2 - 1.866 z + 0.8694
```

```
Sampling time: 0.01
```

Model Order Reduction

You can derive reduced-order models with the following commands.

Model Order Reduction	
<code>balreal</code>	Input/output balancing
<code>minreal</code>	Minimal realization (pole/zero cancellation)
<code>modred</code>	State deletion in I/O balanced realization
<code>sminreal</code>	Structurally minimal realization

Use `minreal` to delete uncontrollable or unobservable state dynamics in state-space models, or to cancel pole/zero pairs in transfer functions or zero-pole-gain models. Use `sminreal` to remove any states that are structurally decoupled from the inputs or outputs. For already minimal models, you can further reduce the model order using a combination of `balreal` and `modred`.

Example: Gasifier Model

This example presents a model of a gasifier, a device that converts solid materials into gases. The original model is nonlinear. To load a linearized version of the model, type

```
load ltexamples
```

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

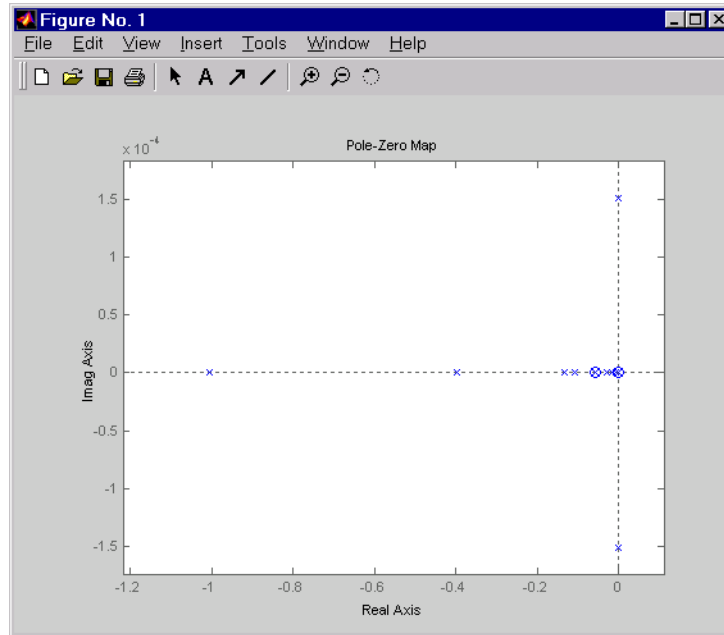
```
size(gasf)
```

MATLAB responds with

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

Before attempting model order reduction, inspect the pole and zero locations by using `pzmap(gasf)` and then zooming in near the origin. If you don't know how to use the zoom feature on plots, see "Zooming" on page 4-45 for an example.

This figure shows the results.



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

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

SISO Model Order Reduction

As an illustration of the model order reduction tools, this example focuses on a single input/output pair of the gasifier, input 5 to output 3.

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

To enhance the numerical stability, first scale the system realization with `ssbal`.

```
sys1 = ssbal(sys35);
```

Then use `minreal` to eliminate uncontrollable or unobservable states.

```
sys1 = minreal(sys1);
```



```
size(sys1)
```

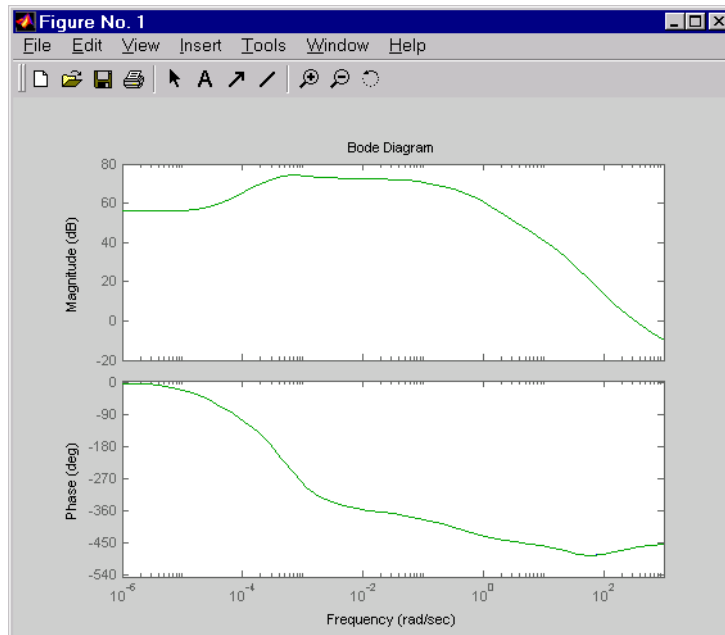
MATLAB responds with

```
State-space model with 1 output, 1 input, and 15 states.
```

The result is a 15th order system. Use this command

```
bode(sys35,sys1);
```

to compare the Bode magnitude and phase of the 25th order model to the reduced-order model. This figure shows the result.



Comparison of Full 25-State Model to the 15-State Reduced Order Models

As the figure shows, there is very little difference in the responses.

Finally, try eliminating states that are weakly affecting the I/O map by using the `balreal` and `modred` functions. First, attempt a balanced realization.

```
[sys1,G] = balreal(sys1);
```

Use `format short e` to view the Hankel singular values stored in variable `G`.

```
G =  
  
    4.5468e+003  
    2.6009e+003  
    1.8601e+003  
    2.5140e+002  
    1.5081e+002  
    1.1993e+001  
    1.1524e+001  
    1.0940e+001  
    2.8766e+000  
    1.3706e+000  
    3.5426e-001  
    2.2556e-002  
    1.2496e-002  
    1.0725e-002  
    6.2703e-005
```

Small Hankel singular values indicate that the associated states are weakly coupled. You can try discarding the last five states (associated with the five smallest Hankel singular values).

```
sys2 = modred(sys1,11:15);      % Down to 10 states  
size(sys2)
```

MATLAB responds with

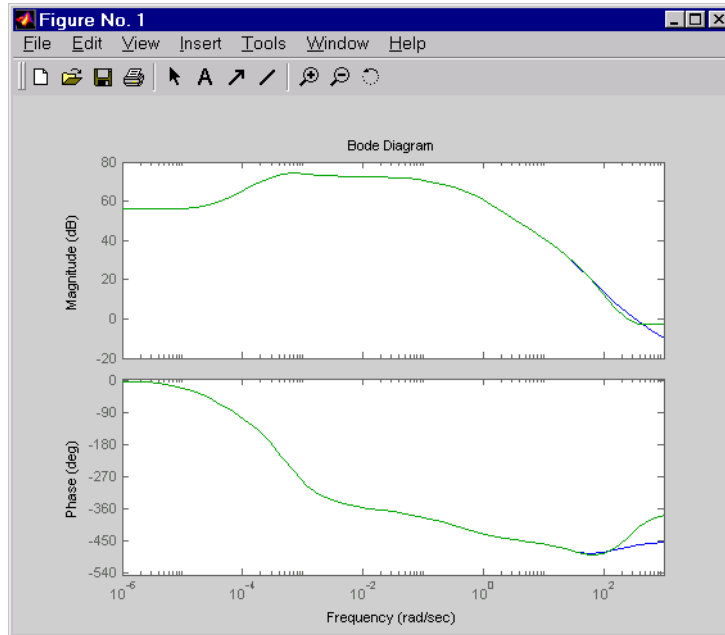
```
State-space model with 1 output, 1 input, and 10 states.
```

Type

```
bode(sys35,sys2);
```

to compare the magnitude and phase of the 10th order model to the 25th order model.

This figure shows the result.



Comparison of 25- and 10-State Models

The figure shows good agreement until the frequency reaches 5 rad/s. This may be acceptable since gasifiers are often low bandwidth systems, and since the models still agree at smaller frequencies. Try experimenting with discarding more Hankel values. With each further reduction, the match to the 25th order model will continue to degrade.

MIMO Model Order Reduction

You can choose not to restrict yourself to individual input/output pairs. The following code does model order reduction on the full MIMO gasifier model.

```
sys1 = ssbal(gasf)      % Scaling

% Compute the minimal realization and balance the model
sys2 = minreal(sys1);  % Down to 17 states
[sys3,G] = balreal(sys2);
% Discard smallest entry of G by using modred
```

```
sys3 = modred(sys3,17);    % Down to 16 states
```

After you get to 16 states, the reduced-order MIMO model begins to deteriorate when compared to the full 25-state MIMO model. Try reducing the model further to see which channels suffer the most degradation.

Acknowledgment

The MathWorks would like to thank ALSTOM Power UK for kindly permitting us to use 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.

Analyzing Models

LTI Viewer (p. 3-2)

How to use the LTI Viewer, a graphical user interface (GUI) that simplifies the analysis of linear, time-invariant systems.

Simulating Models with Arbitrary
Inputs and Initial Conditions (p. 3-17)

How to use the Linear Simulation Tool, a GUI that can generate or import arbitrary input signals and initial conditions for simulation.

Functions for Time and Frequency
Response (p. 3-33)

Plotting time and frequency responses of linear models. You can use these commands when you need more flexibility than the LTI Viewer provides.

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 available only for transfer function, state-space, and zero/pole/gain 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.

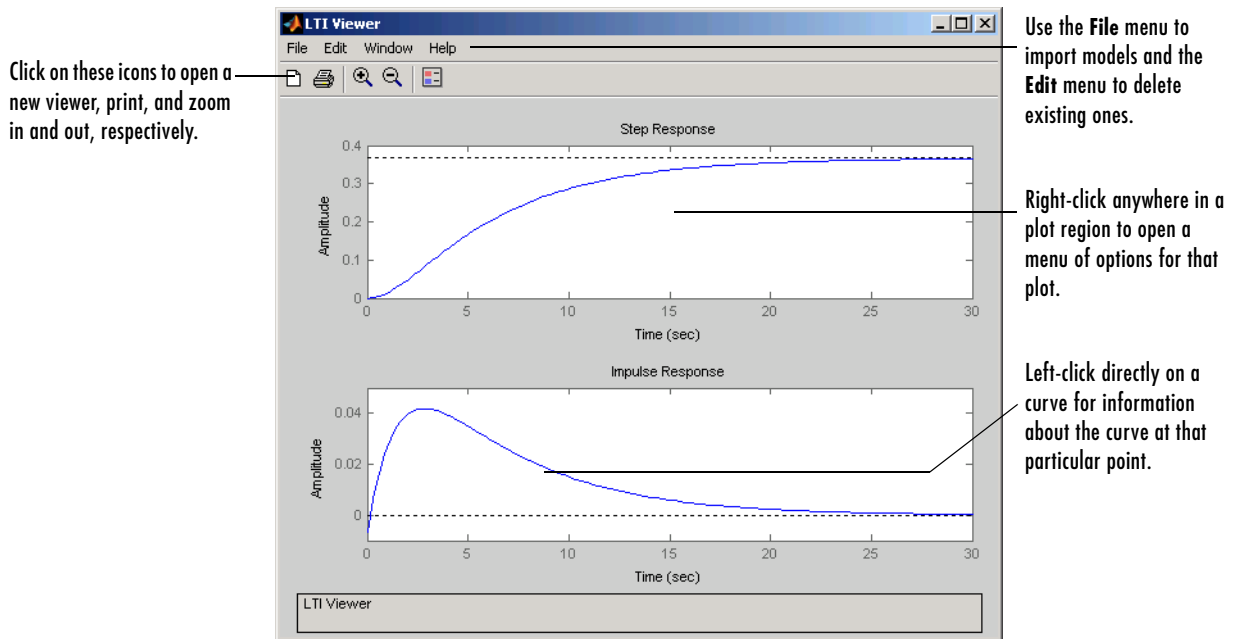


Figure 3-1: The LTI Viewer with Step and Impulse Response Plots

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.

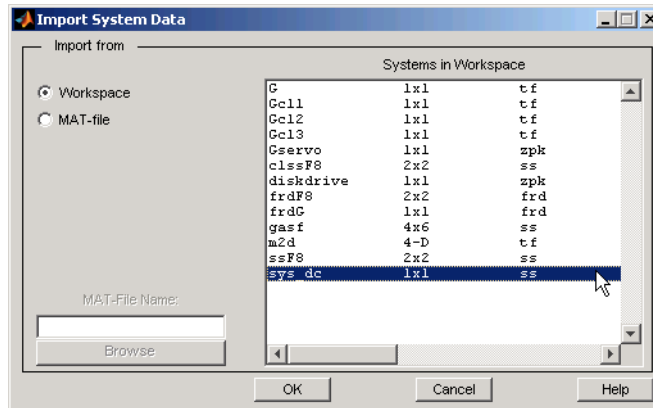


Figure 3-2: Import System Data Dialog Box with the DC Motor Model Selected

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.

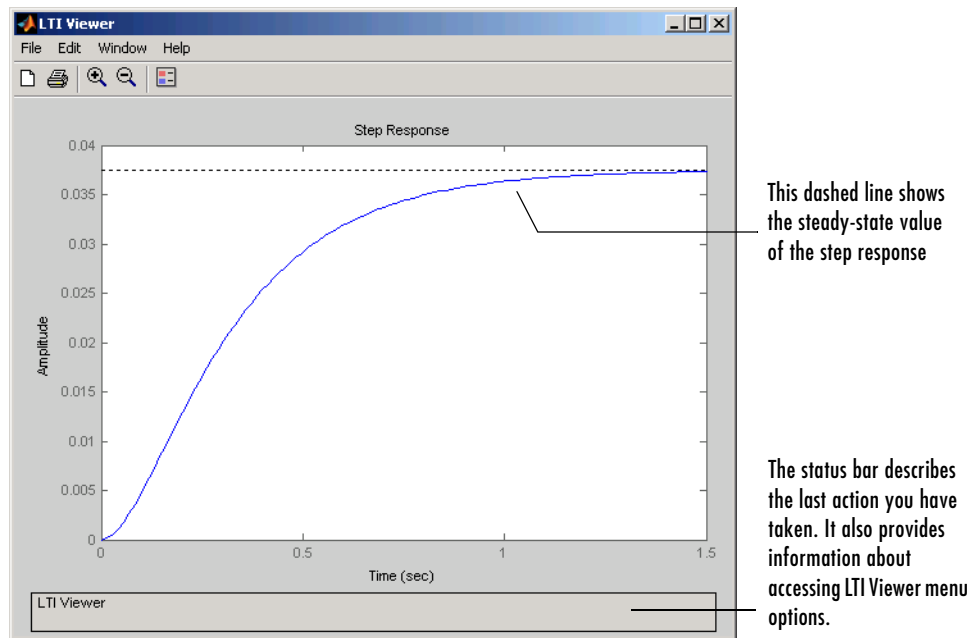


Figure 3-3: Step Response for the DC Motor Example in the LTI Viewer

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

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

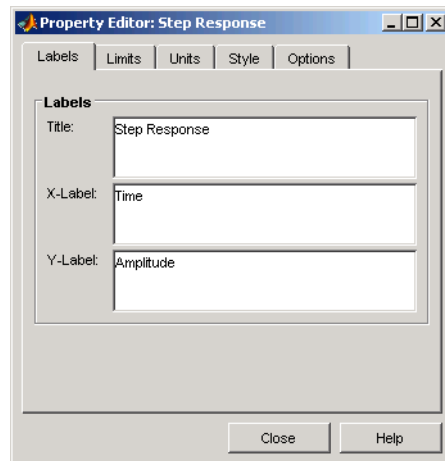
See the `ltiview` reference page for a complete list of options.

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 Type** — 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 your mouse and select **Rise Time** under **Characteristics**, as this figure illustrates.

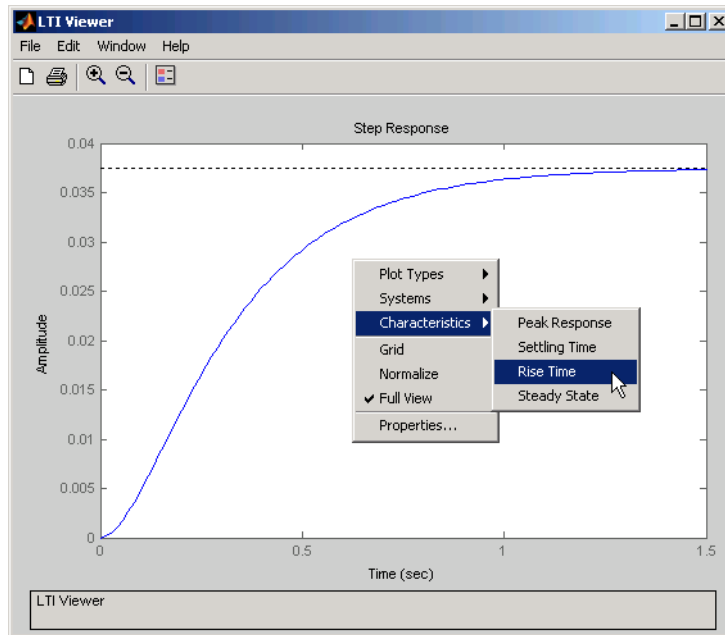


Figure 3-4: Using Right-Click Menus to Display the Rise Time for a Step Response

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.

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

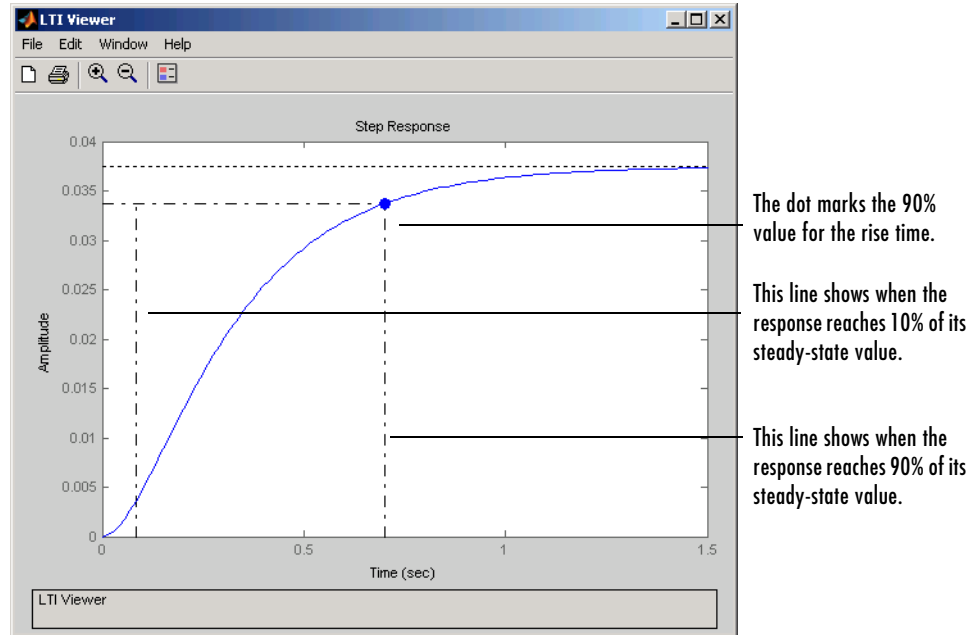


Figure 3-5: DC Motor Step Response with the Rise Time Displayed

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.

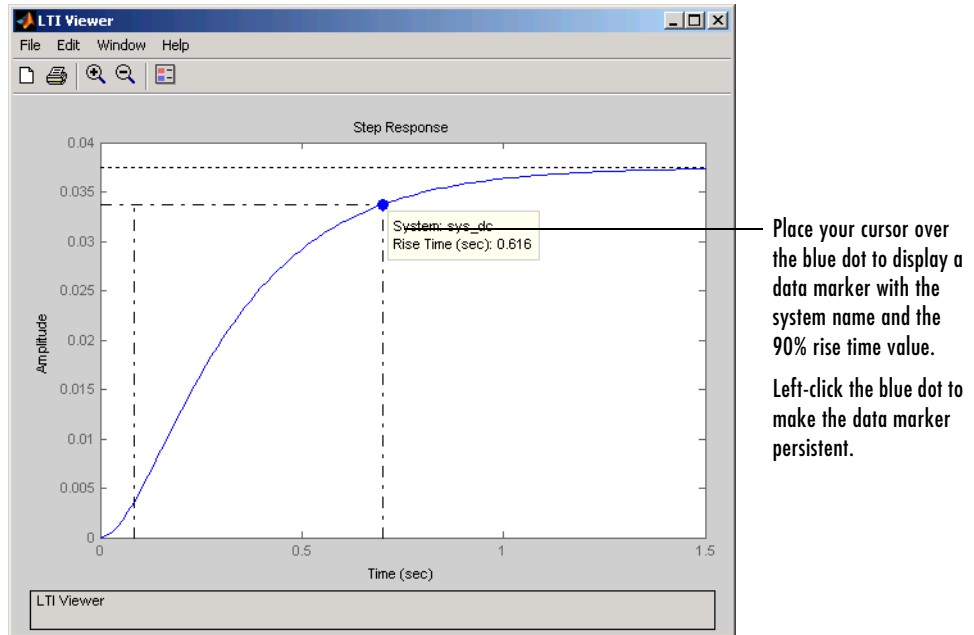


Figure 3-6: Using Your Mouse to Get the Rise Time Values

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 3-38.

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.

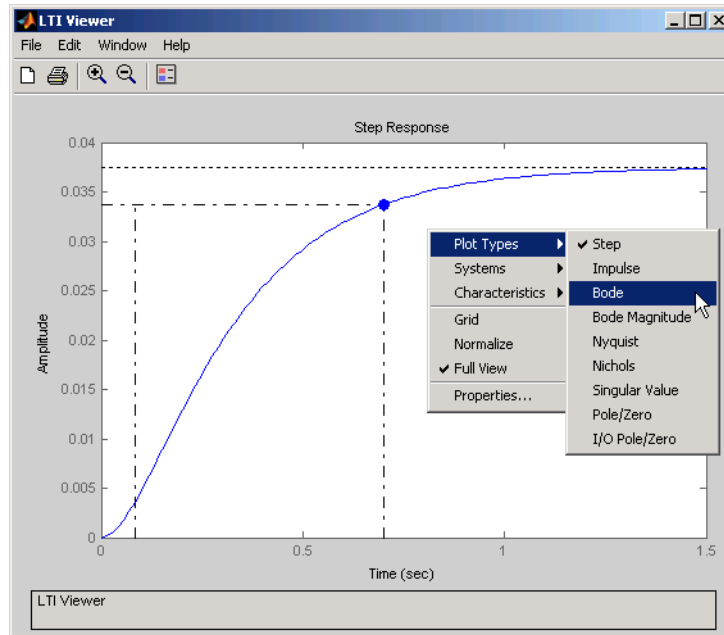


Figure 3-7: Changing the Step Response to a Bode Plot

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

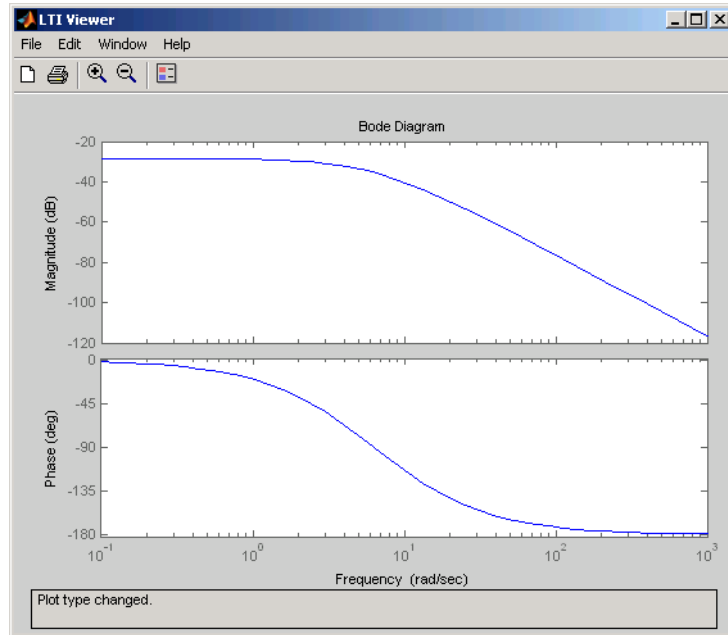


Figure 3-8: Bode Plot for the DC Motor Model

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.

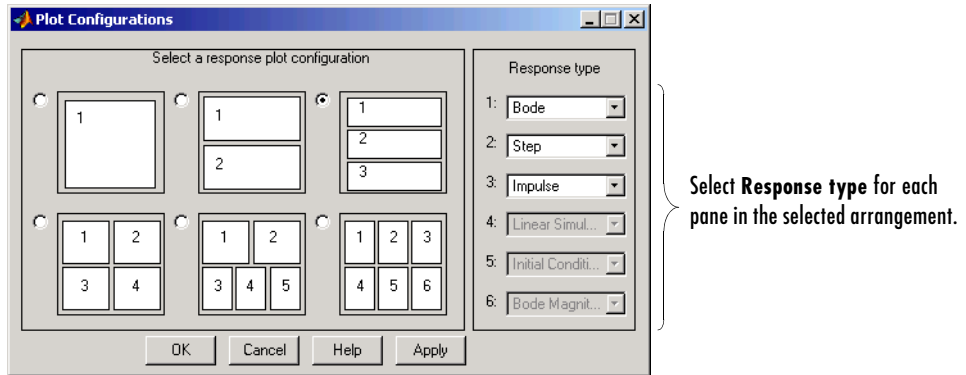


Figure 3-9: Using the Plot Configurations Dialog Box to Reconfigure the LTI Viewer

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
- Bode (magnitude and phase)
- Bode Magnitude (only)
- Nyquist
- Nichols
- Sigma
- 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, `Gc11`, `Gc12`, and `Gc13`, are for a satellite attitude controller.

In this example, you analyze the response plots of the `Gc11` and `Gc12` transfer functions.

Initializing the LTI Viewer with Multiple Plots

To load the two models `Gc11` and `Gc12` 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 3-4 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 3-12 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'},Gc11,Gc12)
```

Either approach opens the following LTI Viewer.

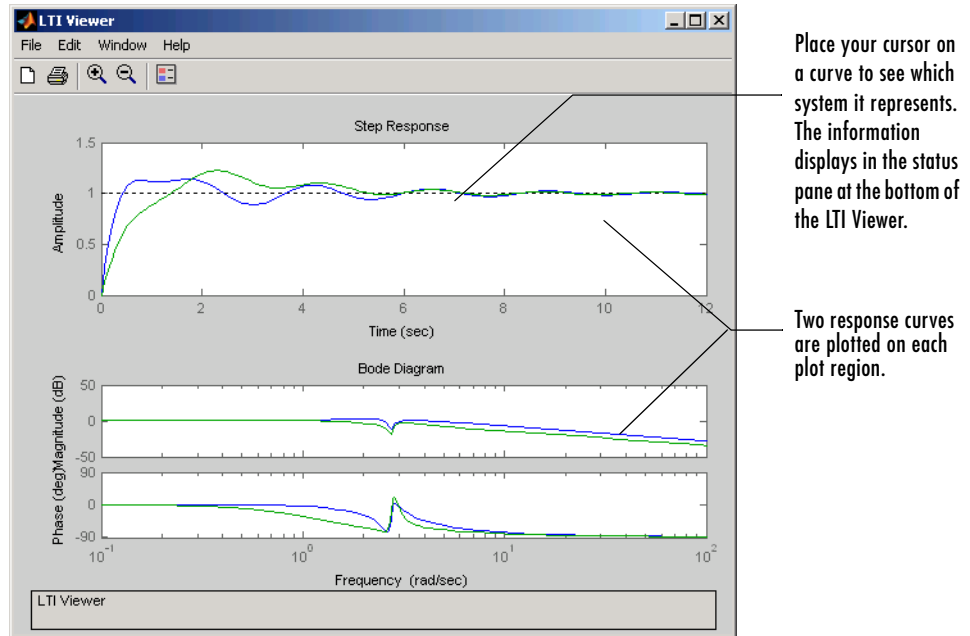


Figure 3-10: Multiple Response Plots in a Single LTI Viewer

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.
- Place your mouse pointer on the **Characteristics** menu item, and select **Settling Time** with your left mouse button.

To mark the stability margins of the Bode plot in this example, open the right-click menu and select **Stability Margins (Minimum Crossing)** under the **Characteristics** menu.

Your LTI Viewer should now look like this.

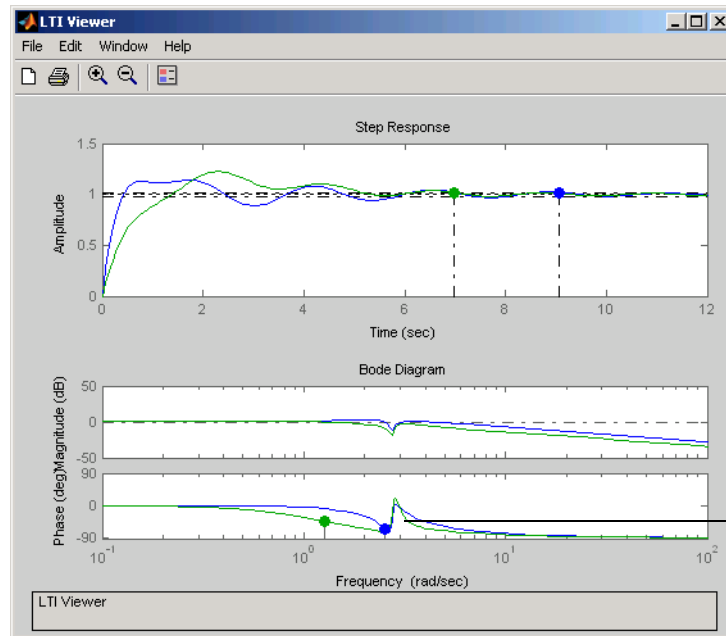


Figure 3-11: Multiple Plots with Response Characteristics Added

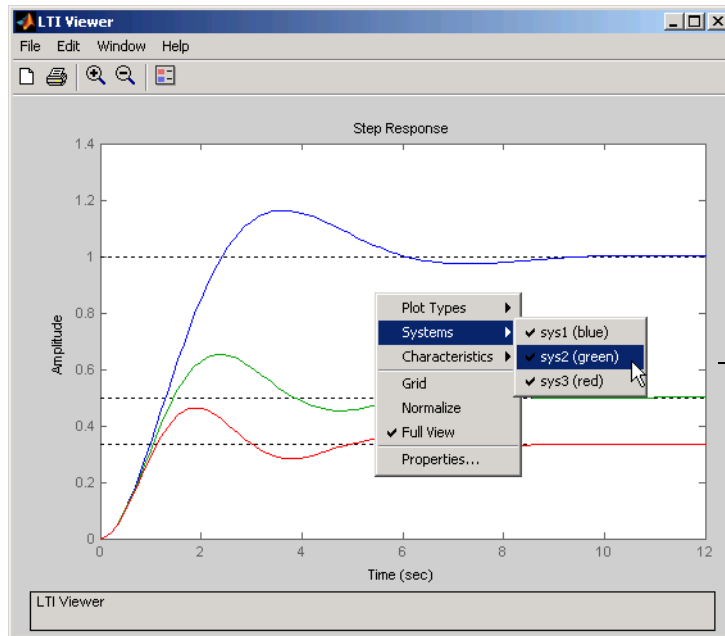
The minimum stability margins, meaning the smallest magnitude phase and gain margins, are displayed as green and blue markers on the Bode phase diagram. If you want to see all the gain and phase margins of a system, select **Stability Margins (All Crossings)** under the **Characteristics** menu item.

Toggleing 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.



Select/clear the systems you want to add or remove from the LTI Viewer under **Systems** in the right-click menu. The menu lists only systems that you have imported into the LTI Viewer.

Figure 3-12: 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

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:

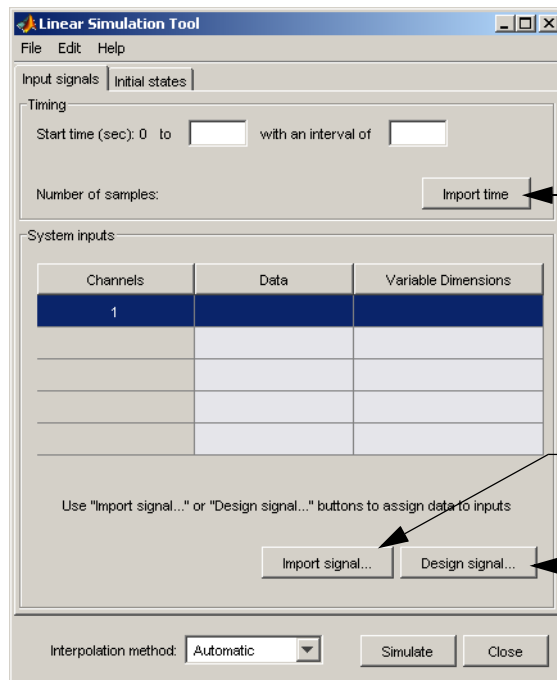
- In the LTI Viewer, right-click the plot area and select **Plot Types > Linear Simulation**.
- Use the `lsim` function at the MATLAB prompt:
`lsim(modelname)`
- In the MATLAB Figure window, right-click a response plot and select **Input data**.

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 3-17), follow these steps to simulate your model:

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



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

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

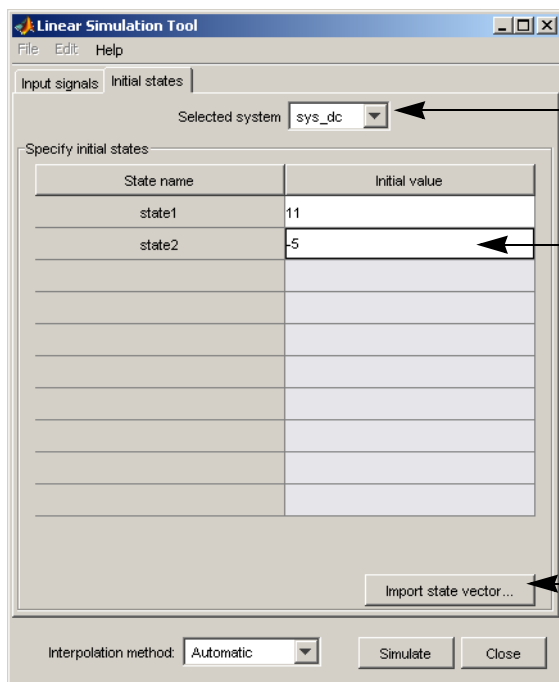
Click **Import signal** to import data from a file.

Click **Design signal** to create an input signal.

- 2 In the **Timing** area, specify the simulation time vector by doing one of the following:
 - Import the time vector by clicking **Import time**.
 - Enter the end time and the time interval in seconds. The start time is set to 0 seconds.

- 3** Specify the input signal by doing one of the following:
- Click **Import signal** to import it from the MATLAB workspace or a file. For more information, see “Importing Input Signals” on page 3-20.
 - Click **Design signal** to create your own inputs. For more information, see “Designing Input Signals” on page 3-29.
- 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 3-32.



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

This list contains names of state-space systems.

Enter initial-state values. Default values are zero.

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

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

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

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

6 Click **Simulate**.

Importing Input Signals

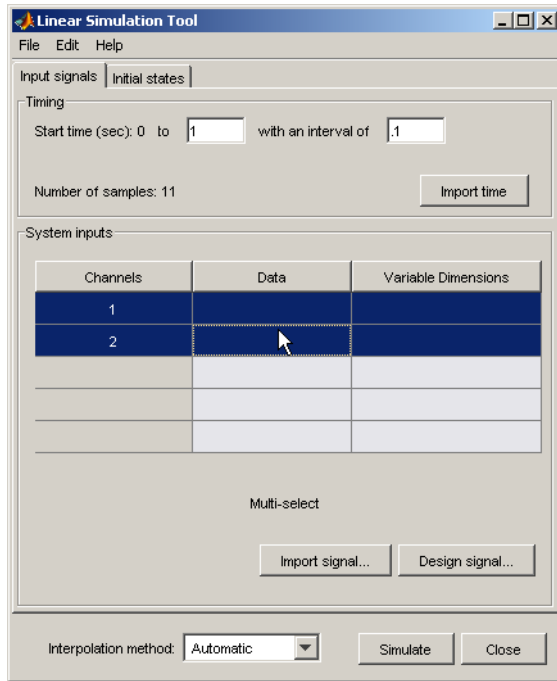
You can import input signals from the MATLAB workspace after opening the Linear Simulation Tool (see “Opening the Linear Simulation Tool” on page 3-17). You can also import inputs from a MAT-file, Microsoft Excel spreadsheet, ASCII flat-file, comma-separated variable file (CSV), or text file.

For information about creating your own inputs, see “Designing Input Signals” on page 3-29. For an overview of working with the Linear Simulation Tool, see “Working with the Linear Simulation Tool” on page 3-18.

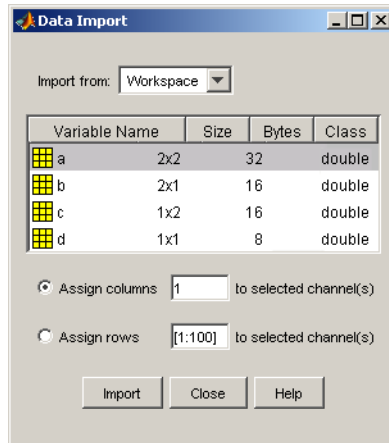
To import one or more input signals:

- 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**.

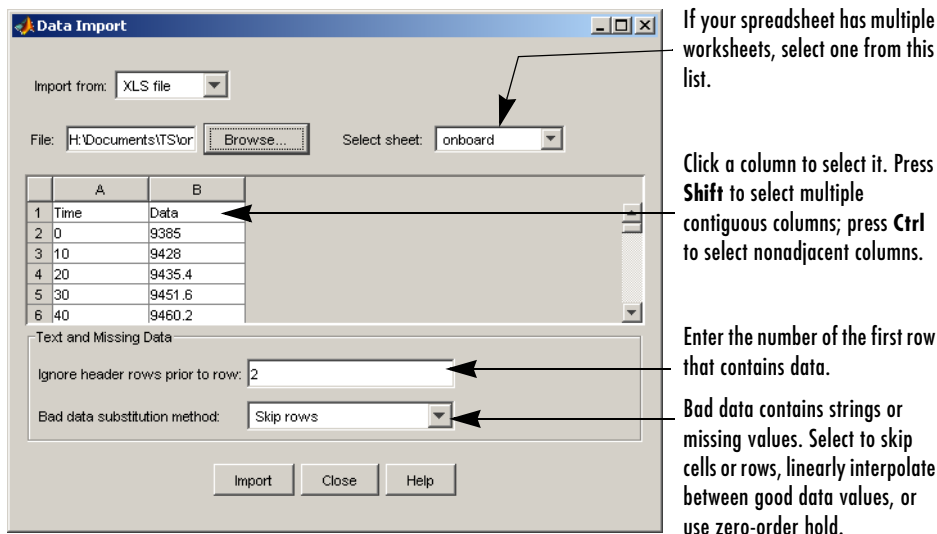
For an example of importing input signals, see the following:

- “Example: Loading Inputs from Excel” on page 3-23
- “Example: Importing Inputs from the Workspace” on page 3-23

Example: Loading Inputs from Excel

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.



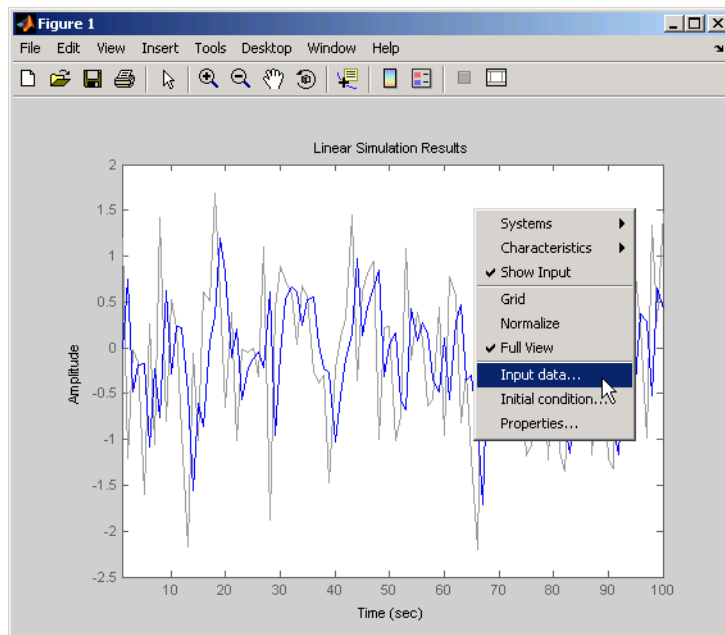
Example: Importing Inputs from the Workspace

To load an input signal from the MATLAB workspace:

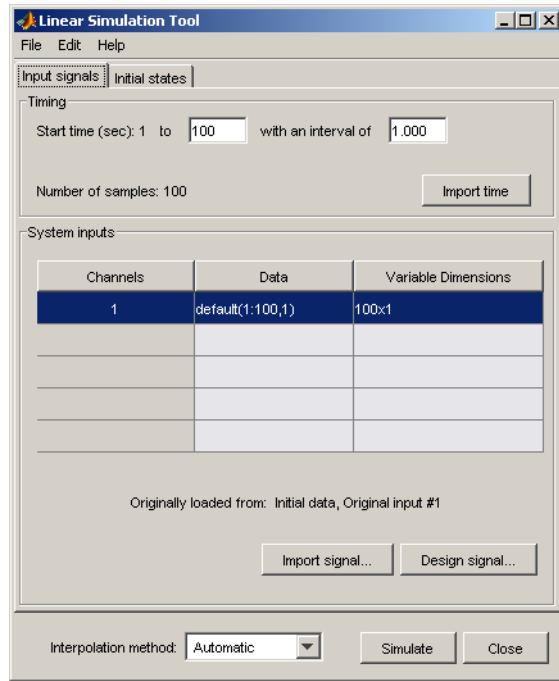
- 1 Enter this code to open a response plot with a second-order system:

```
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.

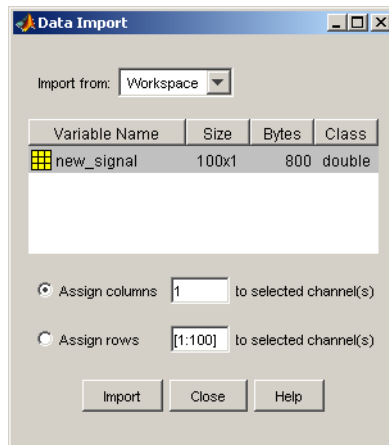


- 3 Create an input signal for your system in the MATLAB Command Window, such as the following:

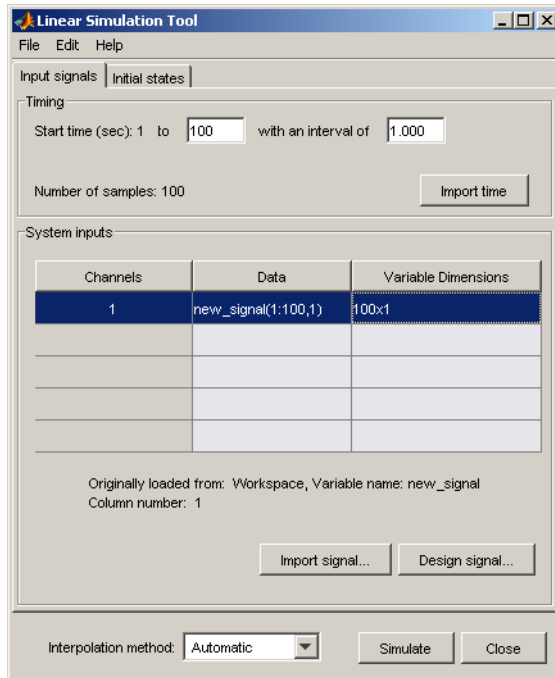
```
new_signal=[ -3*ones(1,20) 2*ones(1,30) 0.5*ones(1,50) ]';
```

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

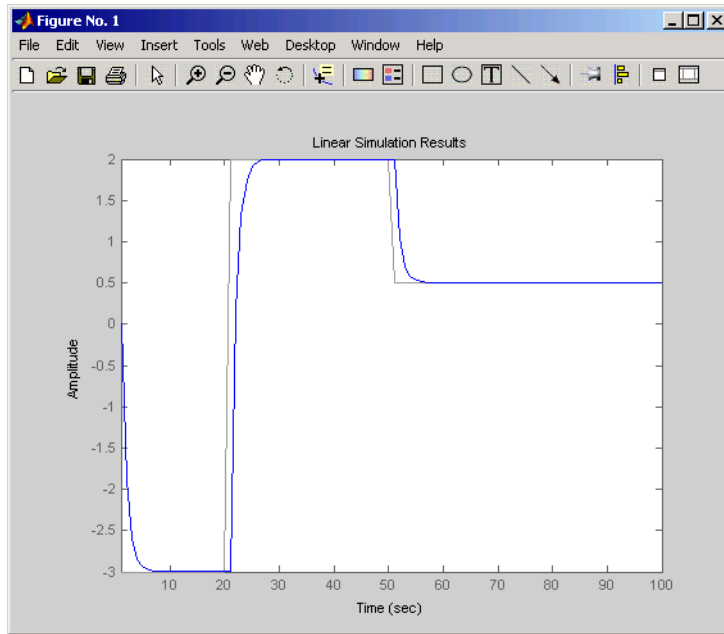
- 5 In the Data Import dialog box, click, **Assign columns** to assign the first column of the input signal to the selected channel.



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

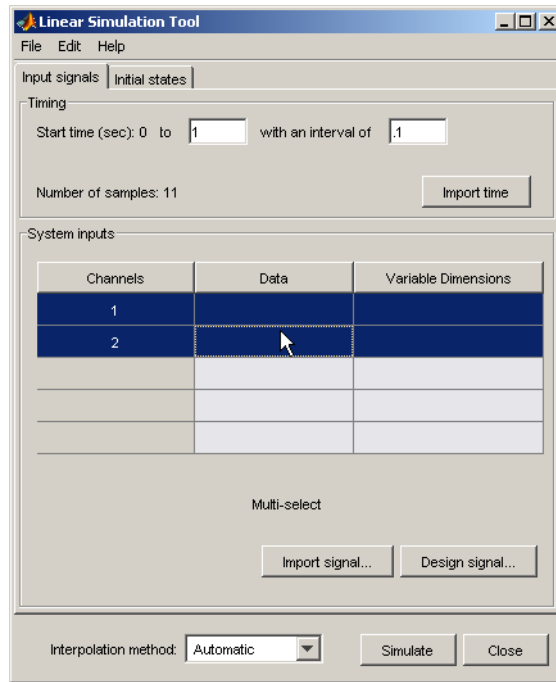
You can generate arbitrary input signals in the form of a sine wave, square wave, step function, or white noise after opening the Linear Simulation Tool (see “Opening the Linear Simulation Tool” on page 3-17).

For information about importing inputs from the MATLAB workspace or from a file, see “Importing Input Signals” on page 3-20. For an overview of working with the Linear Simulation Tool, see “Working with the Linear Simulation Tool” on page 3-18.

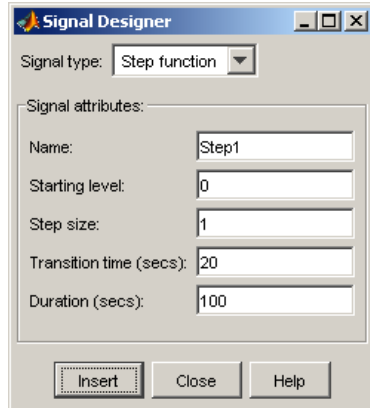
To design one or more input signals:

- 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

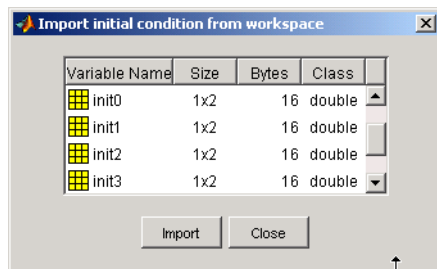
If your system is in state-space form, you can enter or import initial states after opening the Linear Simulation Tool (see “Opening the Linear Simulation Tool” on page 3-17).

For an overview of working with the Linear Simulation Tool, see “Working with the Linear Simulation Tool” on page 3-18.

You can also import initial states from the MATLAB workspace.

To import one or more input signals:

- 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

The Control System Toolbox provides the LTI Viewer, a GUI that is suitable for a wide range of applications. There are situations, however, where you may want a more open and extensible environment. The Control System Toolbox provides a set of functions that provide the 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 Responses

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. The Control System Toolbox provides 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, the Control System Toolbox provides 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.

Table 3-1: 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

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. Their syntax is

```
step(model_name)
impulse(model_name)
initial(model_name,x0)    % x0 = initial state vector
```

where `model_name` is any continuous or discrete LTI model or LTI array.

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.

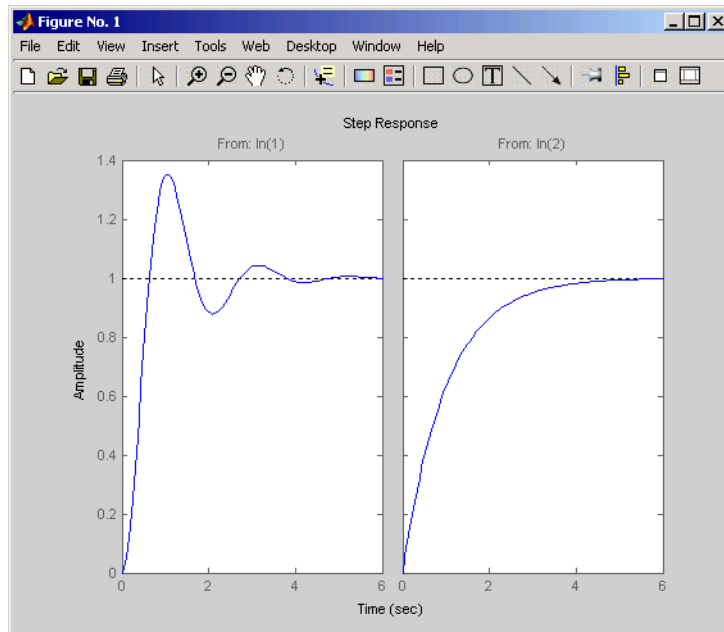


Figure 3-13: Step Responses for a MIMO Model

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. This figure shows the plots from Figure 3-13, Step Responses for a MIMO Model, on page 3-35, with the right-click menu open.

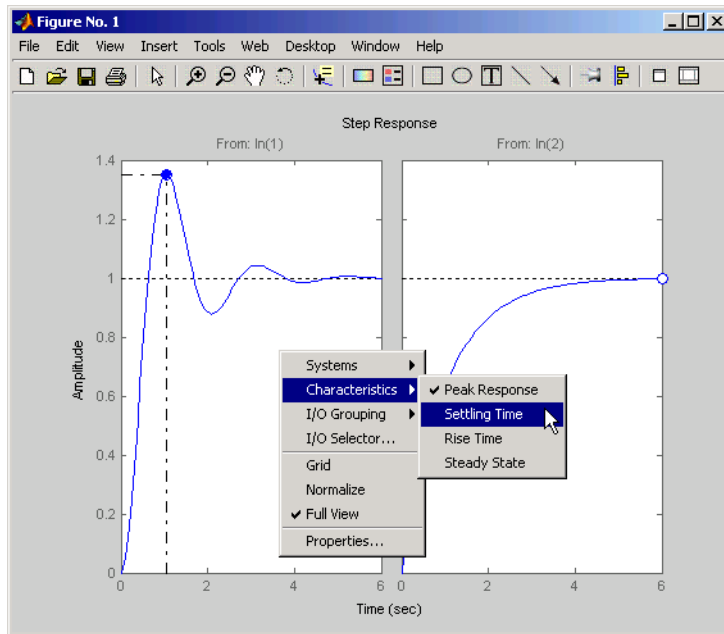


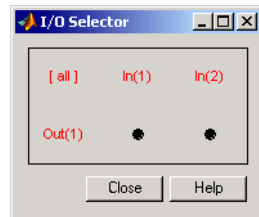
Figure 3-14: Using the Right-Click Menu in a Step Response Plot

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.



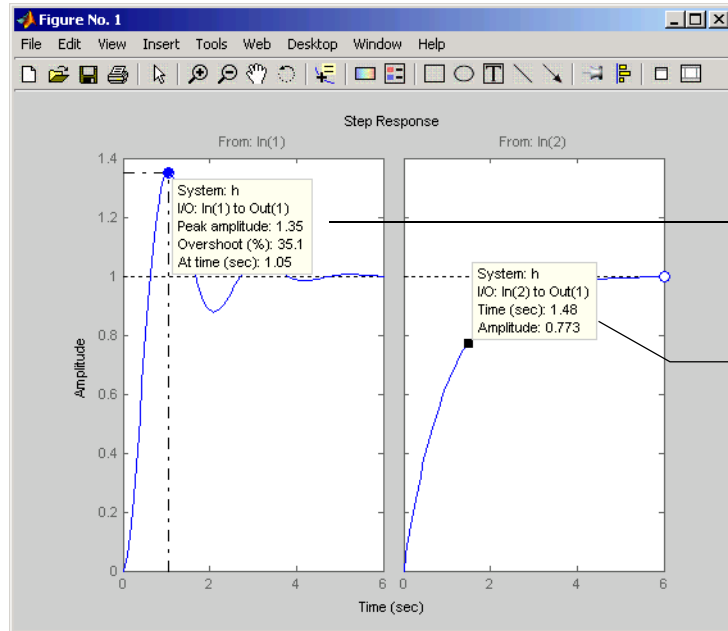
Use this dialog box to select/clear which inputs and outputs to plot.

- **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, the Control System Toolbox provides plot data markers. These allow you to identify key data points on your plots. This figure, using the same plot as Figure 3-13, shows markers on the plots.



Place your cursor on an active characteristic, shown as a blue dot here, to see its name and numerical value.

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

Figure 3-15: 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

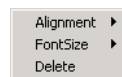
Right-click on any data marker to open a property menu for the marker.



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.
- **Track Mode** — The default is to track x - and y -values. You can choose to track only x - or y -values as well.

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.

```
step(sys1,sys2,...,sysN)
impulse(sys1,sys2,...,sysN)
...
bode(sys1,sys2,...,sysN)
nichols(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,

```
bode(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

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

These commands produce the plot shown below.

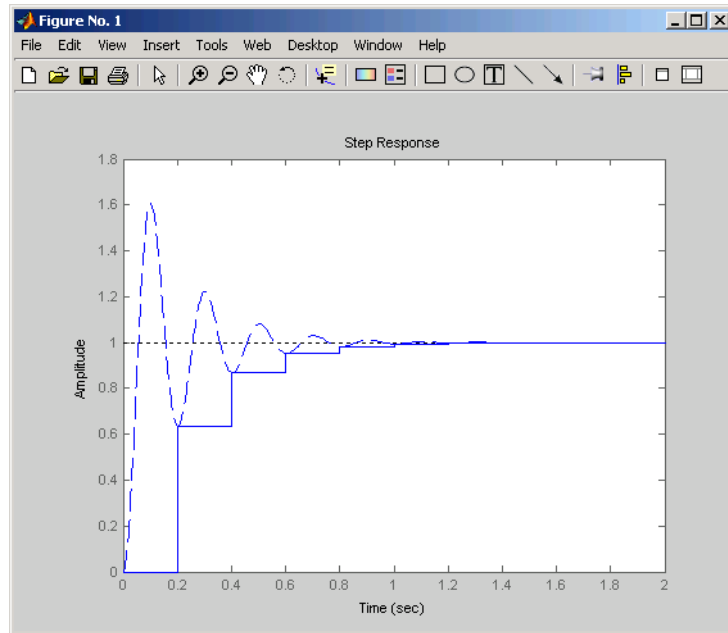


Figure 3-16: Comparison of a Continuous Model to Its Discretized Version

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

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

The Control System Toolbox creates this plot.

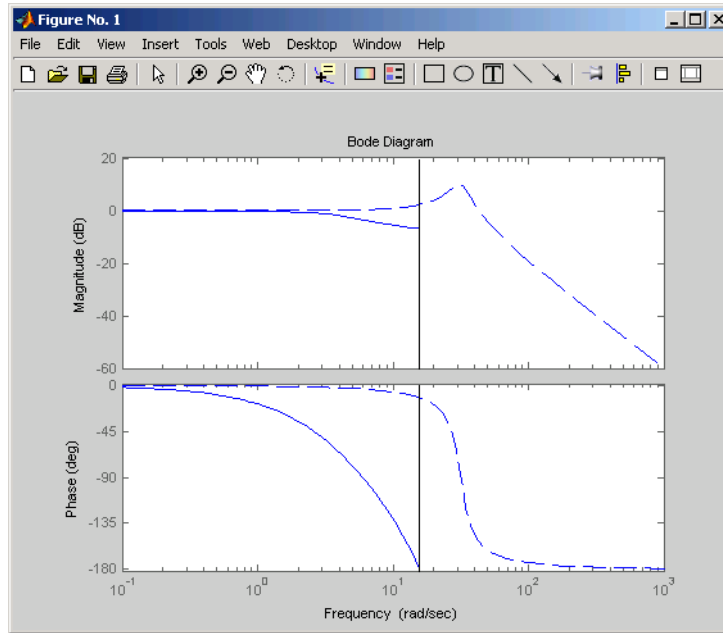


Figure 3-17: Comparison of Bode Plots for a Continuous Model and Its Discretized Version

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)
bode(h)
subplot(222)
step(h)
subplot(223)
pzmap(h)
subplot(224)
pzmap(h)
plot(rand(1, 100)) % Any data can go here
title('Some noise')
    
```

Your plot should look similar to this illustration.

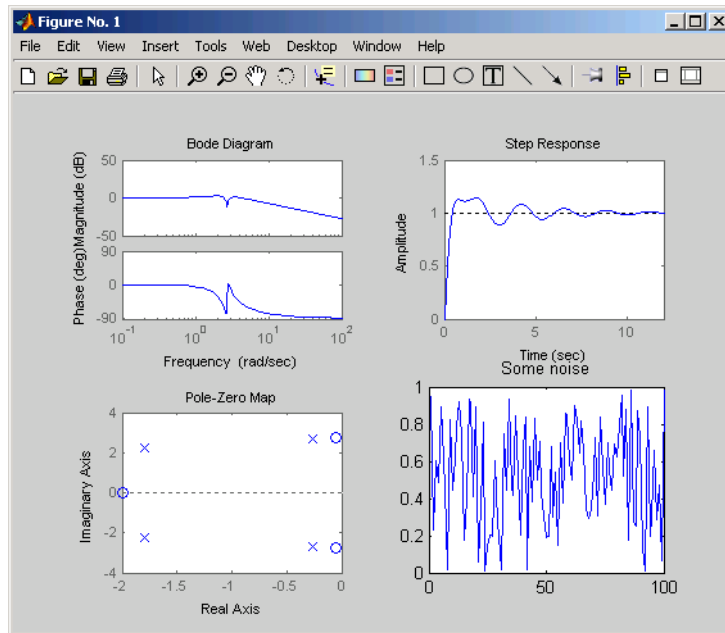


Figure 3-18: Example of Model and Nonmodel Data Plotted in One Window

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 Figure 3-18 (bode, step, and pzmap) 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.

Designing Compensators

SISO Design Tool (p. 4-2)	How to use the SISO Design Tool, which consists of the SISO Design Task in the Control and Estimation Tools Manager, a user interface (UI) that simplifies the task of designing controllers; the Graphical Tuning window, for displaying and manipulating the Bode, root locus, and Nichols plot designs; and a SISO Design Task LTI Viewer.
Bode Diagram Design (p. 4-17)	Designing compensators using Bode diagram techniques. This section presents a DC motor example.
Root Locus Design (p. 4-41)	Using an electrohydraulic servomechanism example, this section discusses the design of compensators using root locus diagram techniques.
Nichols Plot Design (p. 4-65)	Using a DC motor example, this section discusses the design of compensators using Nichols plot techniques.
Automated Tuning Design (p. 4-75)	Using a DC motor example, this section adds a PID controller with the Automated Tuning page on the SISO Design Task in the Control and Estimation Tools Manager.
Multi-Loop Compensator Design (p. 4-81)	Designing compensators with an inner loop. This section uses a position-controlled DC to show the steps needed to do the design.
Functions for Compensator Design (p. 4-92)	Functions for custom applications, MIMO design, pole placement, LQG design, and Kalman filtering. This section includes several examples of these more advanced techniques.

SISO Design Tool

The SISO Design Tool is made up of the following:

- The **SISO Design Task** in the Control and Estimation Tools Manager, a user interface (UI) that facilitates the design of compensators for single-input, single-output feedback loops through a series of interactive pages (referred to in this document as the **SISO Design Task** node).
- The Graphical Tuning window, 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” on page 3-2.

The SISO Design Tool facilitates the design of compensators for single-input, single-output feedback loops, and lets you iterate rapidly 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.
- Add and tune lead/lag networks and notch filters.
- Inspect closed-loop responses (using the LTI Viewer).
- Adjust phase and gain margins.
- Convert models between discrete and continuous time.
- Automate compensator design.

Opening the SISO Design Tool

This section shows how to open the SISO Design Tool with the DC motor example developed in Chapter 2, “Building Models.”

If you have not built the DC motor model, type

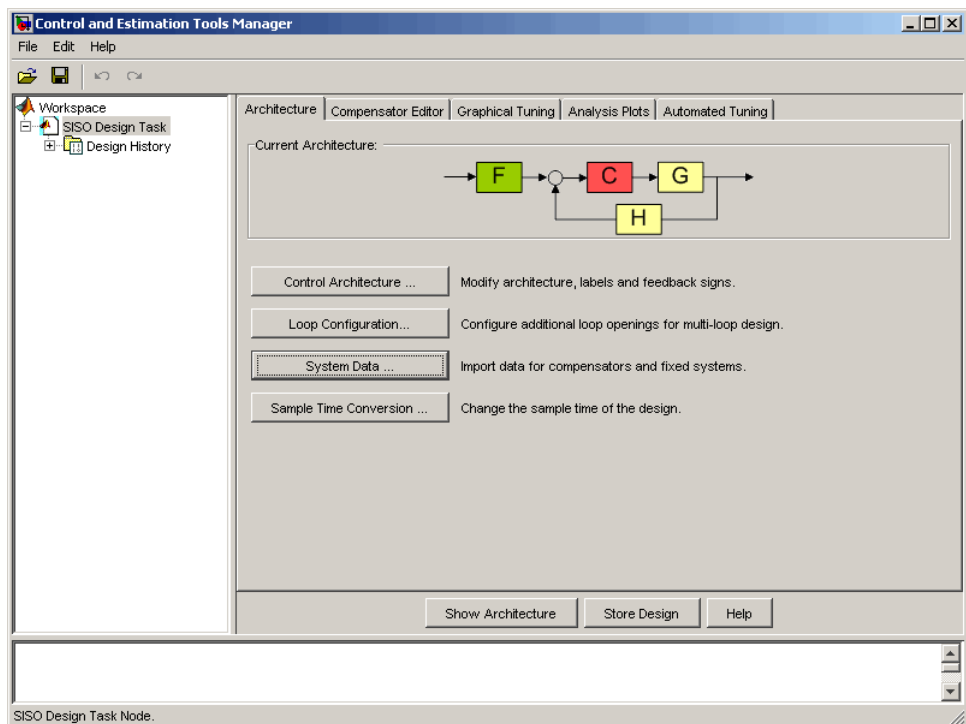
```
load ltiexamples
```

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

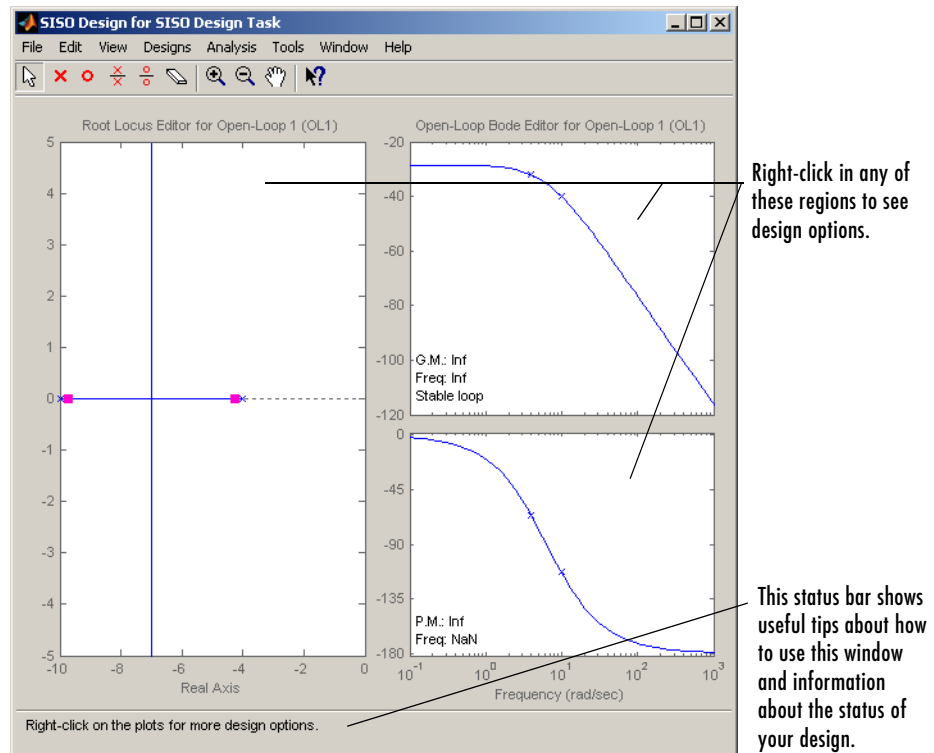
```
sisotool(sys_dc)
```

at the MATLAB prompt.

This command opens both the **SISO Design Task** node on the Control and Estimation Tools Manager and 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 Page View)



Graphical Tuning Window with the DC Motor Example

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

The **SISO Design Task** node in the Control and Estimation Tools Manager contains the following pages 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 models into your system.

- Convert the sample time of the system or switch between different sample times to design different compensators.
- **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 PID, internal model control (IMC), or linear-quadratic-Gaussian (LQG) methods (uses the Control System Toolbox).
 - Use a response optimization tool that automatically tunes the system to satisfy design requirements (available when you have the Simulink[®] Response Optimization product).

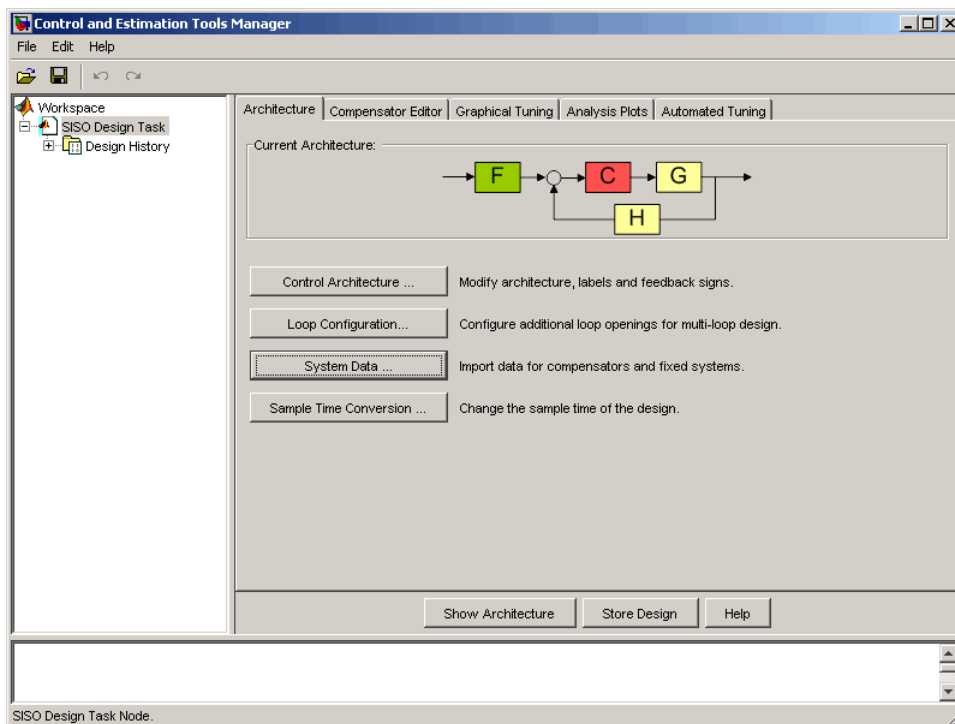
Importing Models into the SISO Design Tool

If you type

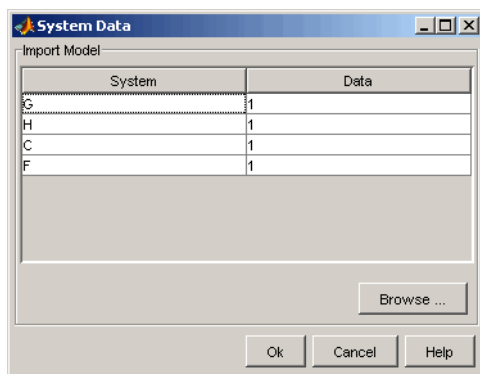
```
sisotool
```

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 the DC motor model by clicking **System Data** on the **Architecture** page, shown in the following figure.

4 Designing Compensators

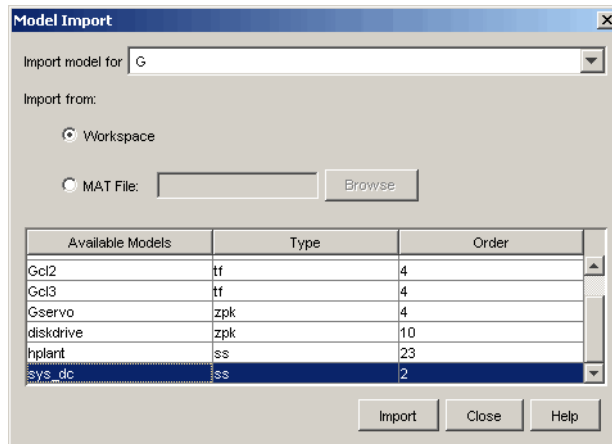


This opens the System Data dialog box, which is shown in the following figure.

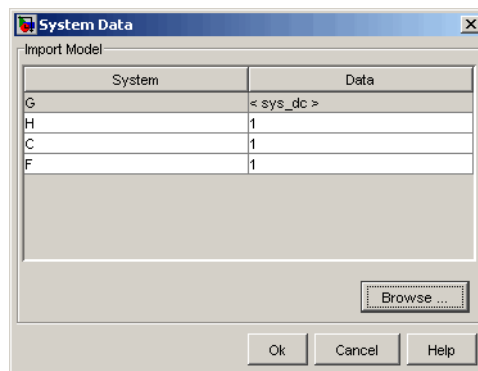


To import the DC motor model:

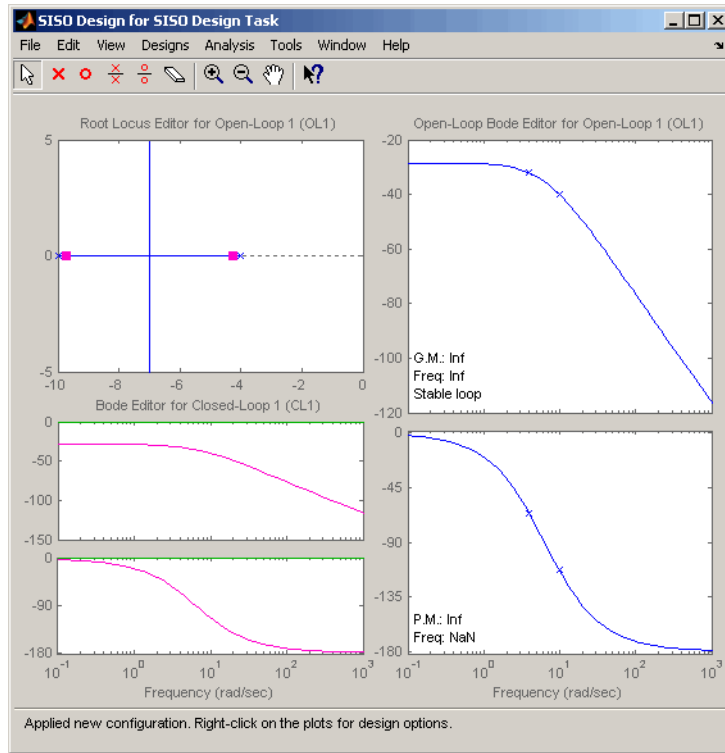
- 1 Select **G** and click **Browse**. The Model Import dialog box opens, as shown in the following figure.



- 2 Select **sys_dc** from the **Available Models** list. Click **Import**, and then click **Close**. You can now see **sys_dc** loaded into **G** in the System Data dialog box.

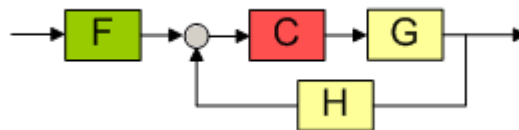


- 3 Click **OK**. The Graphical Tuning window is updated with the DC motor model, as shown in the following figure.



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, the lettered boxes represent the following:

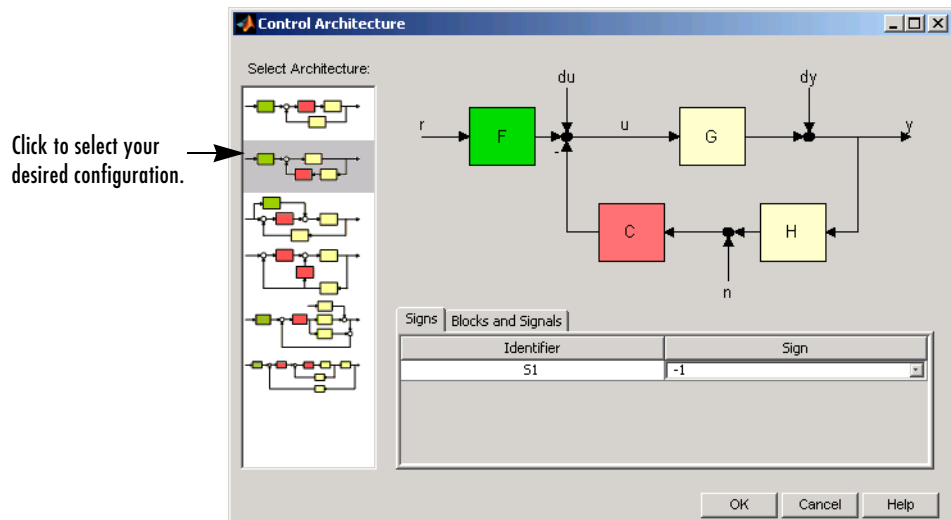
- G — plant

- **H** — sensor dynamics
- **F** — prefilter
- **C** — compensator

The default values for **F**, **H**, and **C** are all 1 (you can see this in the System Data dialog box). Note that this means that by default, the compensator has unity gain. **G** contains the DC motor model, `sys_dc`.

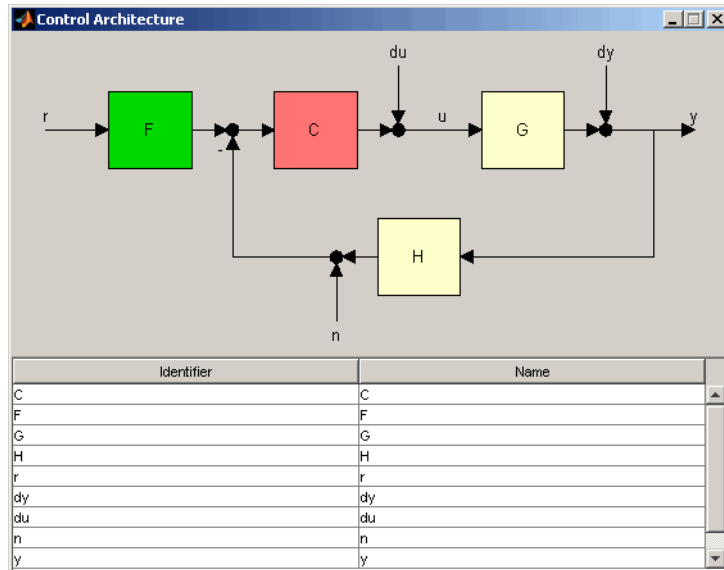
Alternative Feedback Structures

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



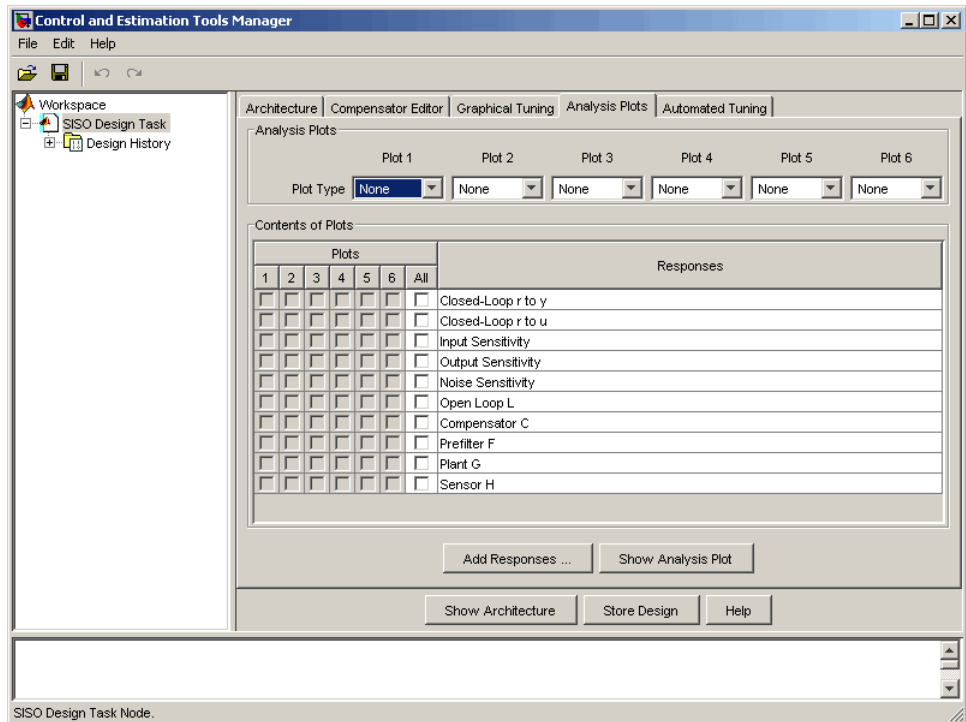
You can use the **Signs** and **Blocks and Signals** panes to change the sign of the feedback signal into a summing junction and rename blocks and signals in the diagram respectively. See “Modifying Block Diagram Structure” for more details.

On any page in the **SISO Design Task** node on the Control and Estimation Tools Manager, click **Show Architecture** to see the current architecture and a list of the identifiers and names associated with the components.

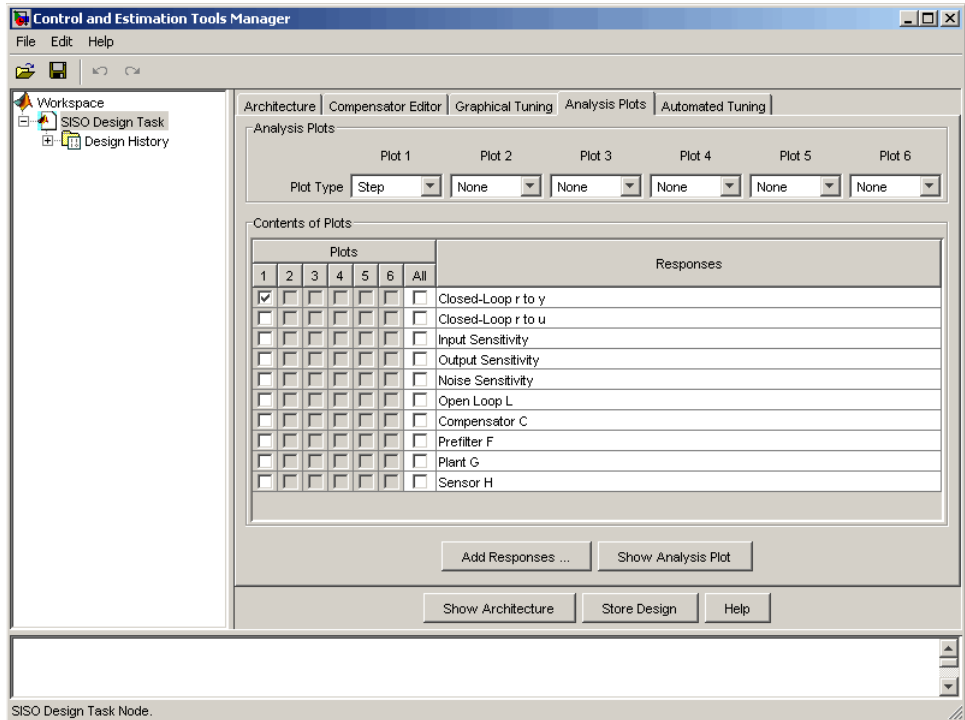


Loop Responses

As you select different compensator designs, you may find it convenient to be able to examine the various loop responses (for example, step or impulse responses) for a particular design. To view, for example, the closed-loop step response, click the **Analysis Plots** tab. This opens the **Analysis Plots** page containing the list of available responses, with none initially selected, as shown in the following figure.



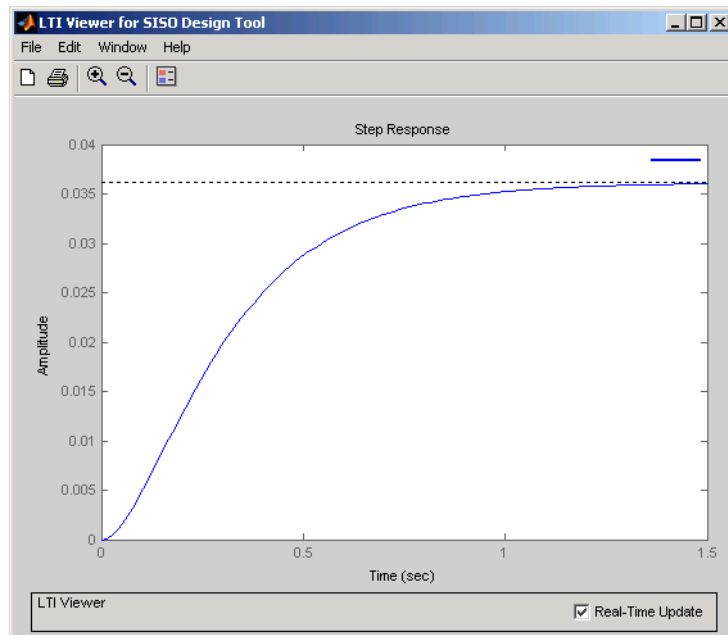
Select the plot types for each plot in the **Analysis Plots** group box, and then select the plots to appear in the **Plots** list in the **Contents of Plots** table, as shown in the following figure.



Analysis Plots Loop Response Selection

After you have selected a plot, the LTI Viewer with 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

As this plot shows, the step response of the DC motor is about 1.5 seconds, which is too slow for many applications. Also, there is a large steady-state error. The following sections show how to use Bode diagram techniques for improving the response time and steady-state error of the DC motor step response.

As you select different compensator designs, the LTI Viewer associated with your **SISO Design Task** will automatically update the response plots you have chosen for a particular design.

Using the Graphical Tuning Window

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. Most tasks can be accomplished using the pages in the **SISO Design Task** node on the Control and Estimation Tools Manager. Many of these tasks can also be done in the Graphical Tuning window, though you will find it easier to use the pages in the **SISO Design**

Task node on the Control and Estimation Tools Manager. However, there are a few tasks, such as adjusting the bandwidth, that can only be done using the Graphical Tuning window.

This section describes some of the methods for navigating in and manipulating the appearance of the Graphical Tuning window.

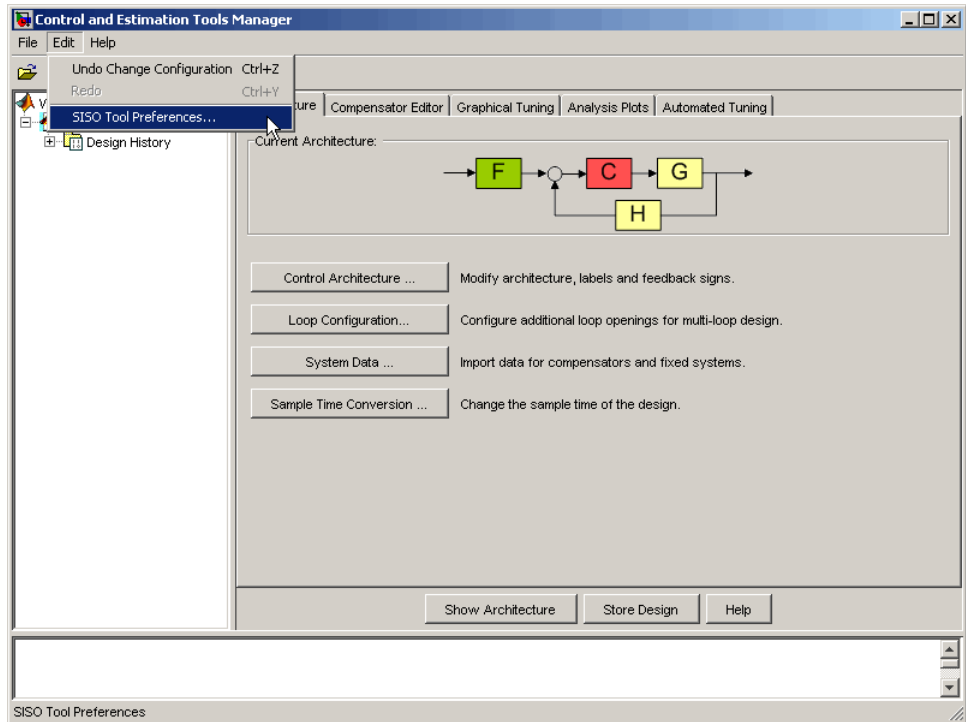
Graphical Tuning Window Display

The Graphical Tuning window shows

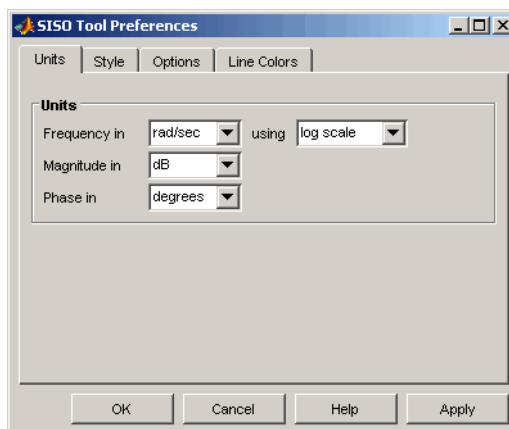
- Poles as x's
- Zeros as o's
- Gain and phase margins (by default) in the lower-left corners of the Bode magnitude and phase plots

Changing Units on a Plot

The SISO Design Tool provides editors for setting plot options in the Graphical Tuning window. If you want, for example, 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 next.



This opens the SISO Tool Preferences dialog box.

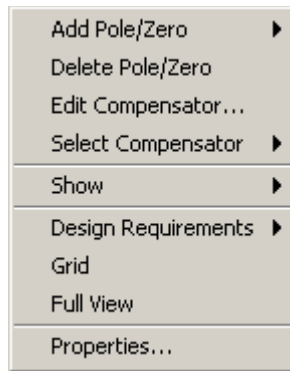


Use the options on the **Units** page to make the change. This unit change persists for the entire session.

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 your mouse in the white space of 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; for example, where the Closed-Loop Bode Editor right-click menu has a **Select Compensator** option, the Open-Loop Bode Editor right-click menu has a **Gain Target** option instead.

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 in “Using the SISO Design Tool and the LTI Viewer.”

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, and add notch filters for disturbance rejection.

Example: DC Motor

The following sections use the DC motor example to show how 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

$$\text{Transfer function:} \\ \frac{1.5}{s^2 + 14s + 40.02}$$

For this example, the design criteria are as follows:

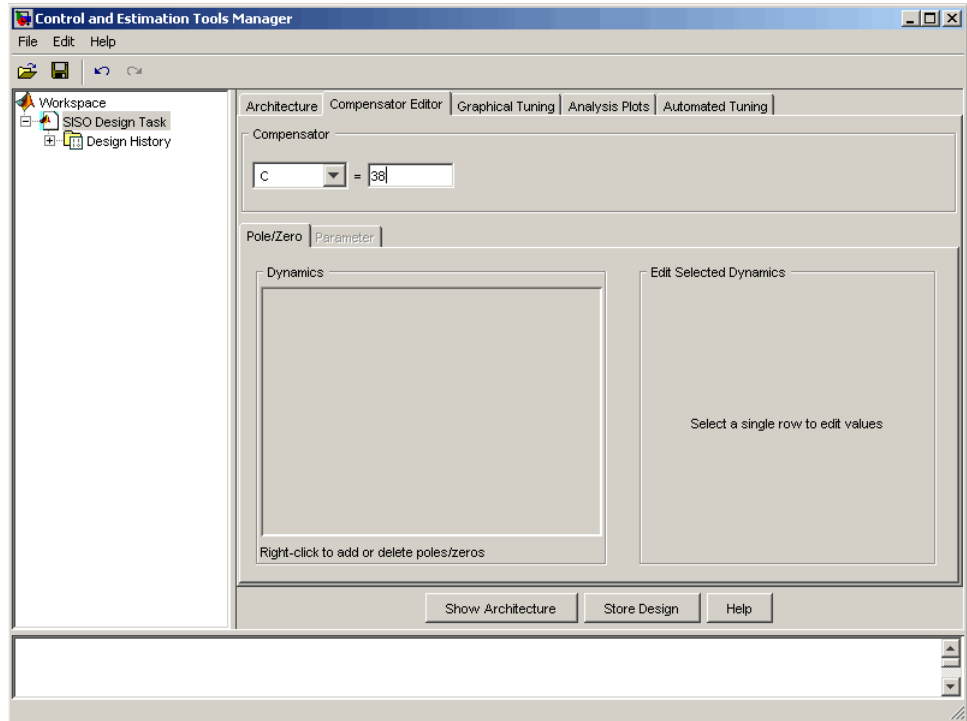
- Rise time of less than 0.5 second
- Steady-state error of less than 5%
- Overshoot of less than 10%
- Gain margin greater than 20 dB
- Phase margin greater than 40 degrees

Adjusting the Compensator Gain

The LTI Viewer Showing the Step Response for the DC Motor, on page 4-13, 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.

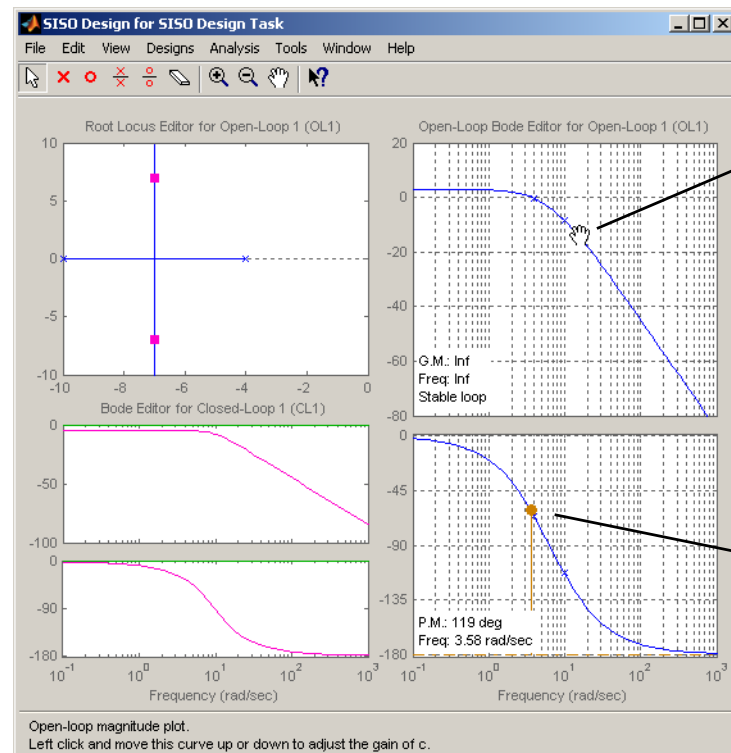
Adjusting the Bandwidth

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.

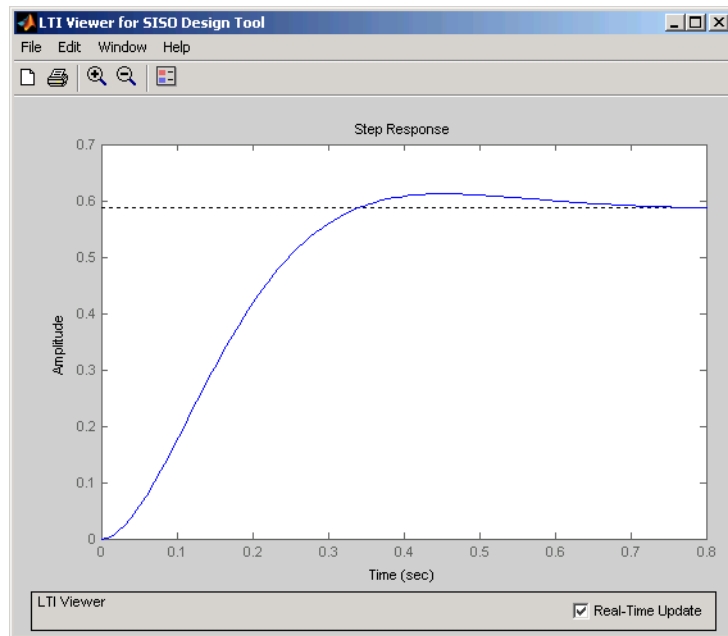
For a crossover at 3 rad/s, the compensator gain should be about 38. By default, the Graphical Tuning window shows gain and phase margin information in the lower-left corners of the Bode diagrams. In the Bode magnitude plot, it also tells you if your closed-loop system is stable or unstable.

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

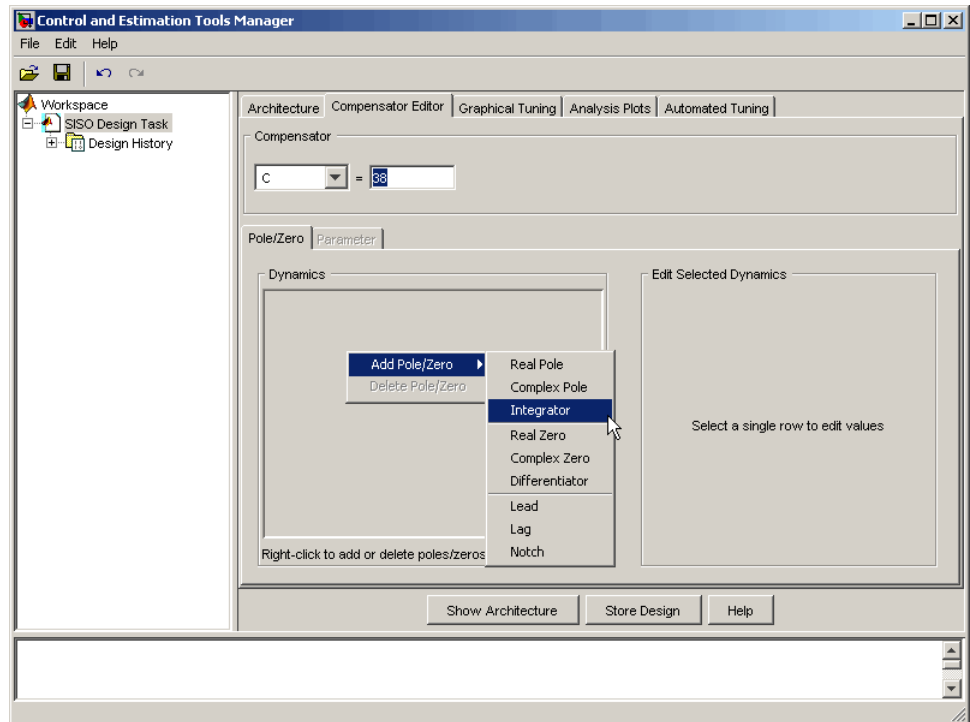
The step response shows that the steady-state error and rise time have improved somewhat, but you must design a more sophisticated controller to meet all the design specifications, in particular, the steady-state error requirement.

Adding an Integrator

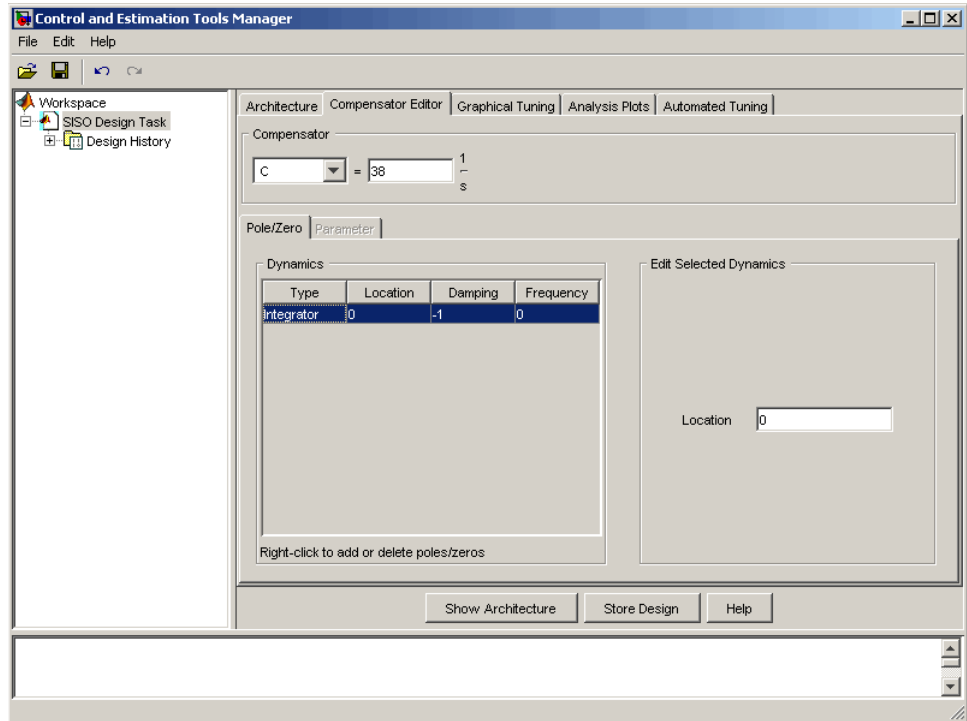
One way to eliminate steady-state error is to add an integrator. To add an integrator:

- 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.



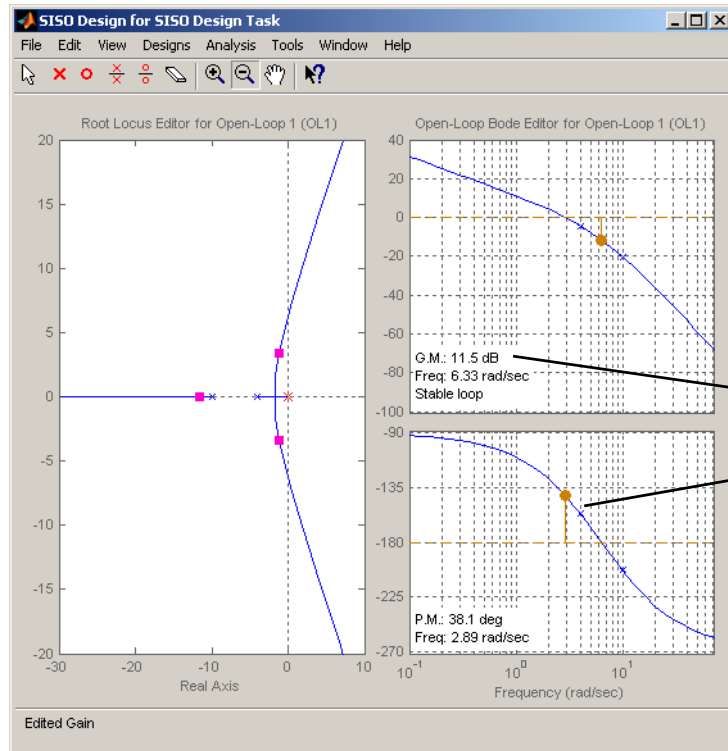
Adding an Integrator in the Dynamics Table



Editable Integrator Parameters

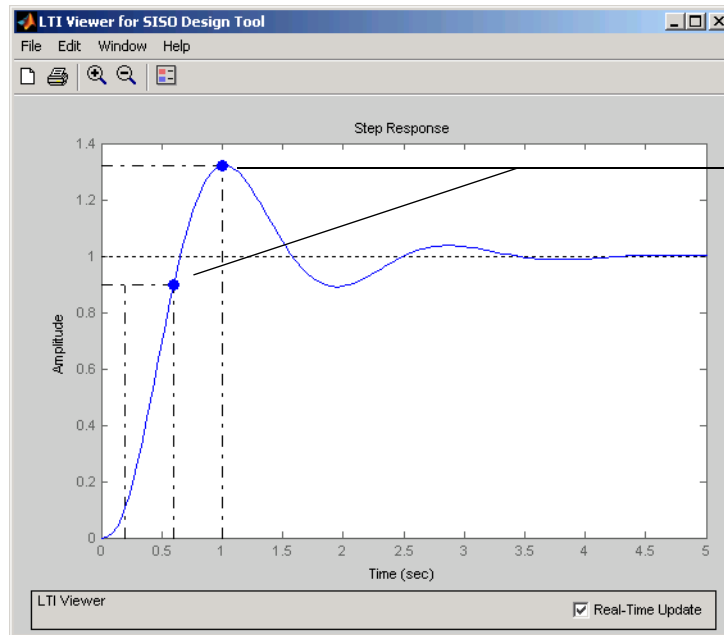
Notice adding the integrator changed the crossover frequency of the system. Readjust the compensator gain in the **Compensator Editor** page to bring the crossover back to 3 rad/s; the gain should be about 100.

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.



Use the right-click menu to show the **Peak Response and Rise Time** (listed under **Characteristics**).

Step Response for the DC Motor with an Integrator in the Compensator

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

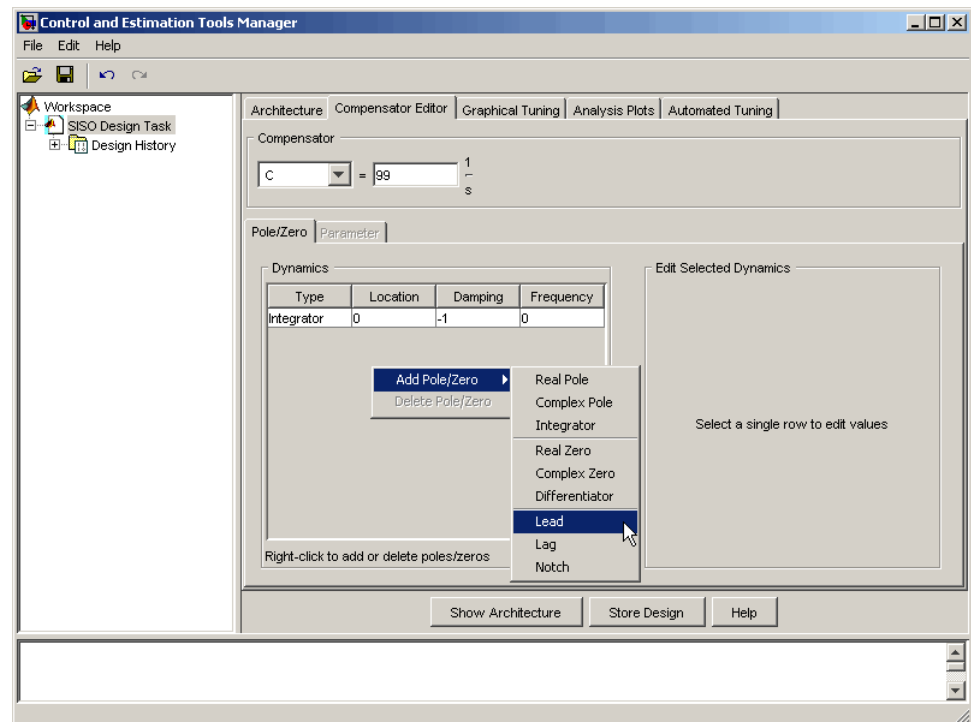
Part of the design requirements is a gain margin of 20 dB or greater and a phase margin of 40° or more. In the current compensator design, the gain margin is 11.5 dB and the phase margin is 38.1° , both of which fail to meet the design requirements. So two goals left are to shorten the rise time while 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.

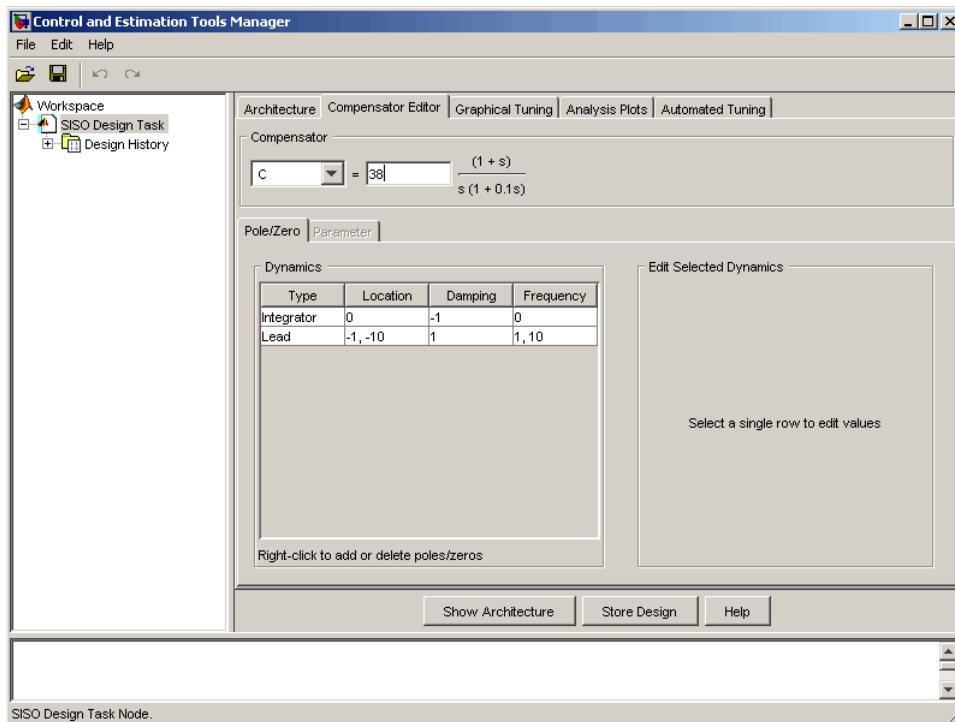
One possible solution is to add a lead network to the compensator. To add the lead network:

- 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.

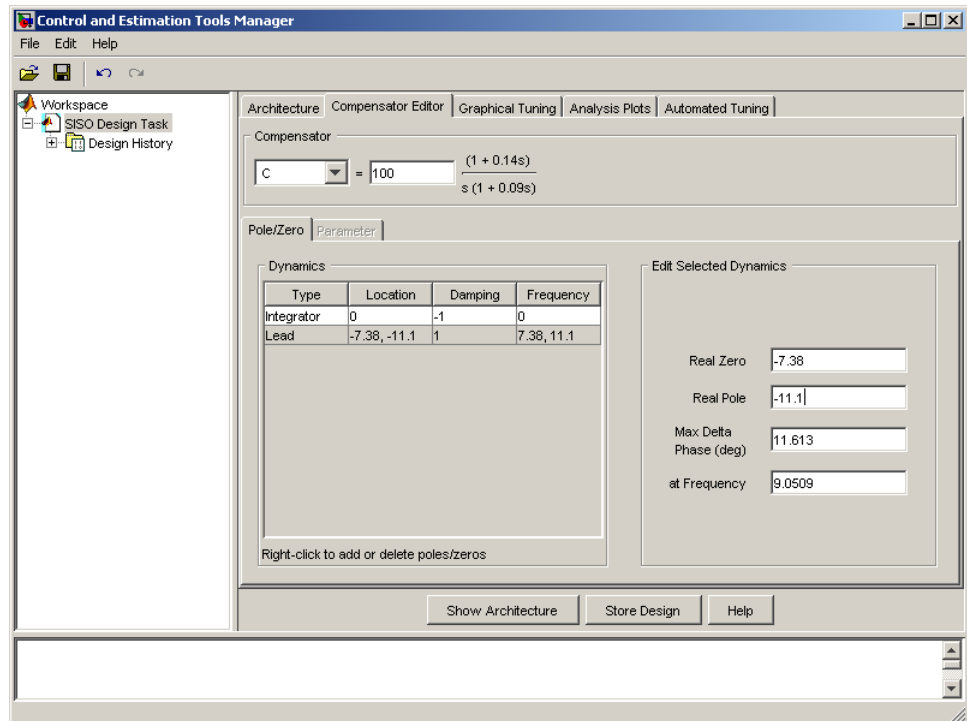


Adding a Lead Network to the DC Motor Compensator on the Compensator Editor Page



Lead Network Added

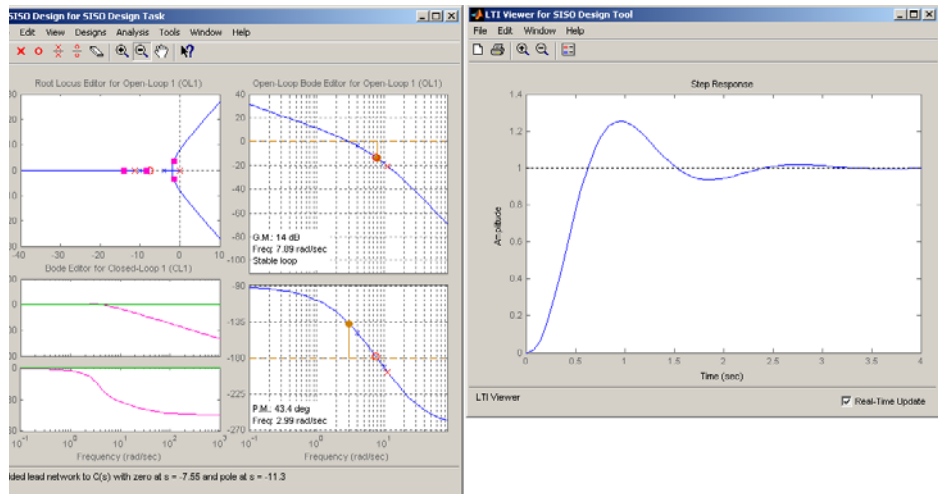
Editable fields are shown in the **Edit Selected Dynamics** group box (right side of page) when an item in the **Dynamics** table has been selected, as shown in the following figure.



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 > Integrator**, 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 1.25 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.

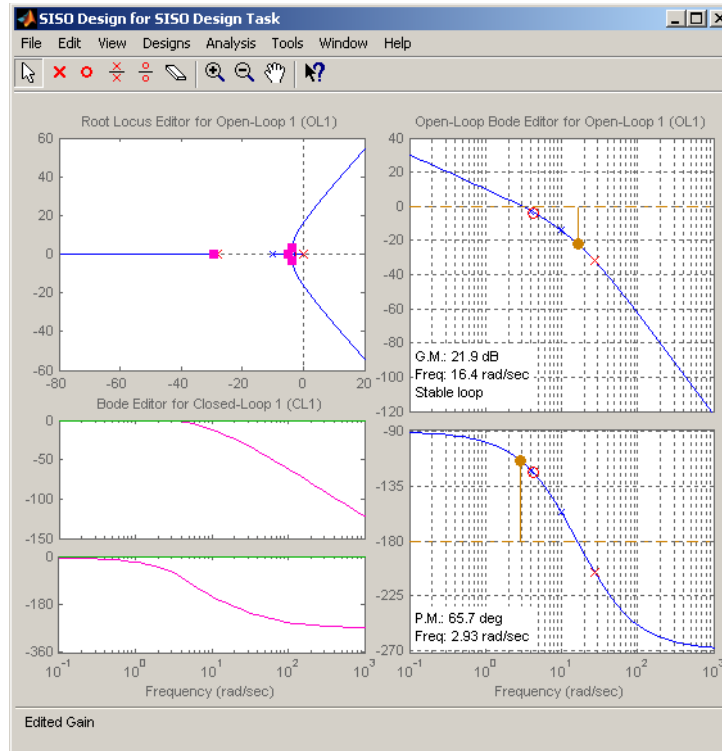
Moving Compensator Poles and Zeros

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.



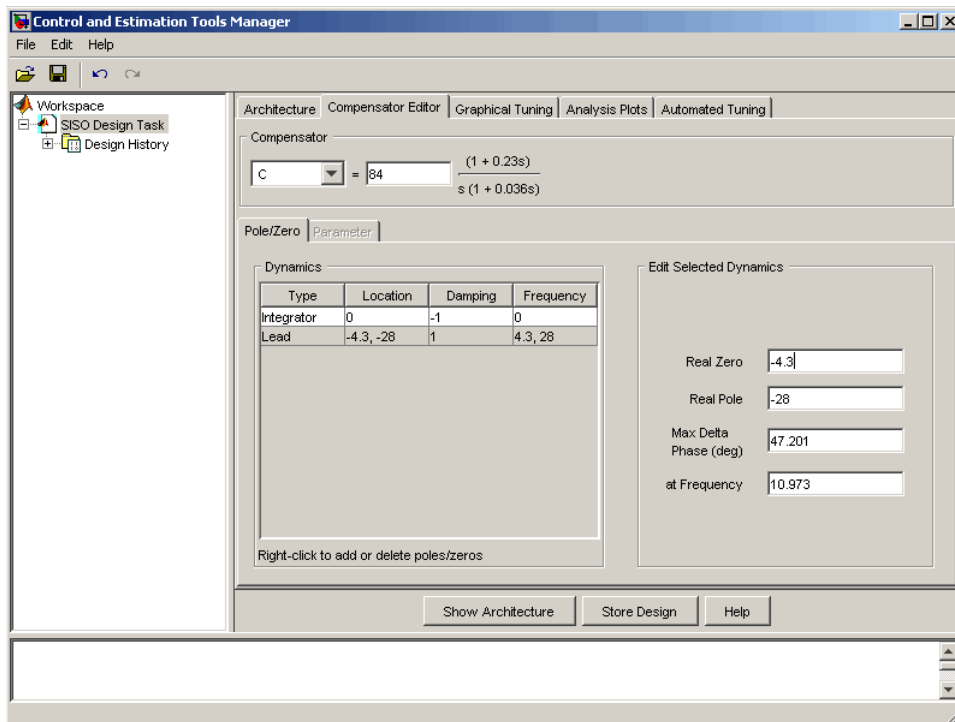
Graphical Tuning Window with Final Design Parameters for the DC Motor Compensator

The values for this final design are as follows:

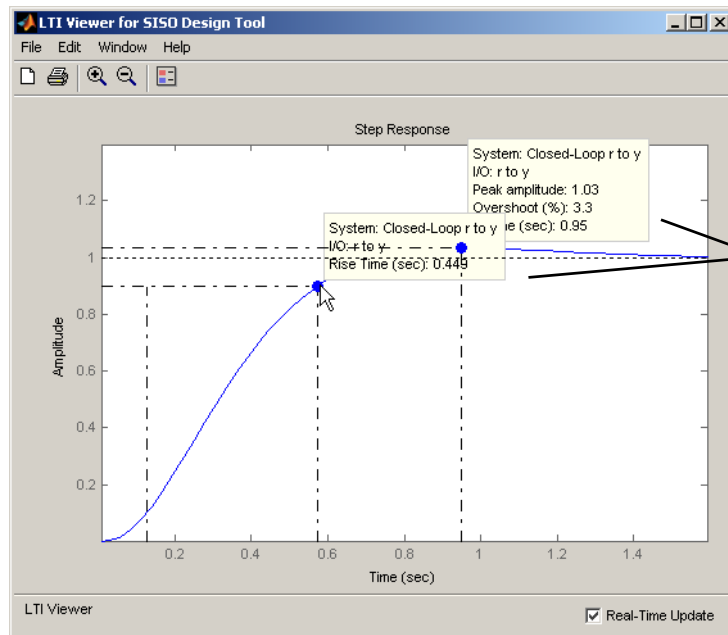
- Poles at 0 and -28
- Zero at -4.3
- Gain = 84

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).

4 Designing Compensators



Entering Final Design Parameters on the Compensator Editor Page

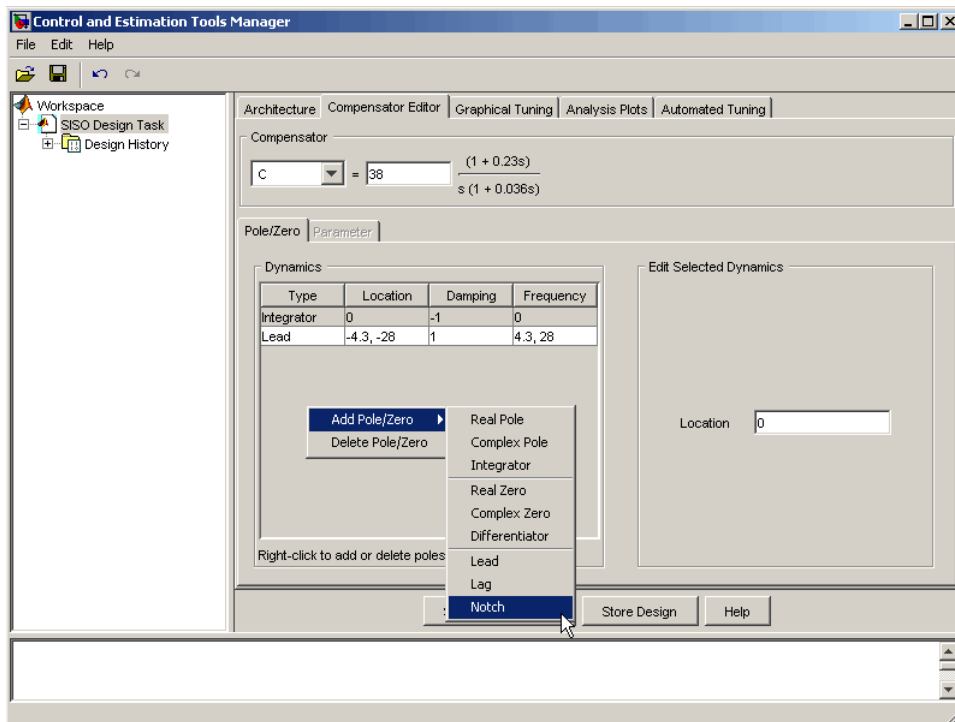


Step Response for the Final Compensator Design

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.

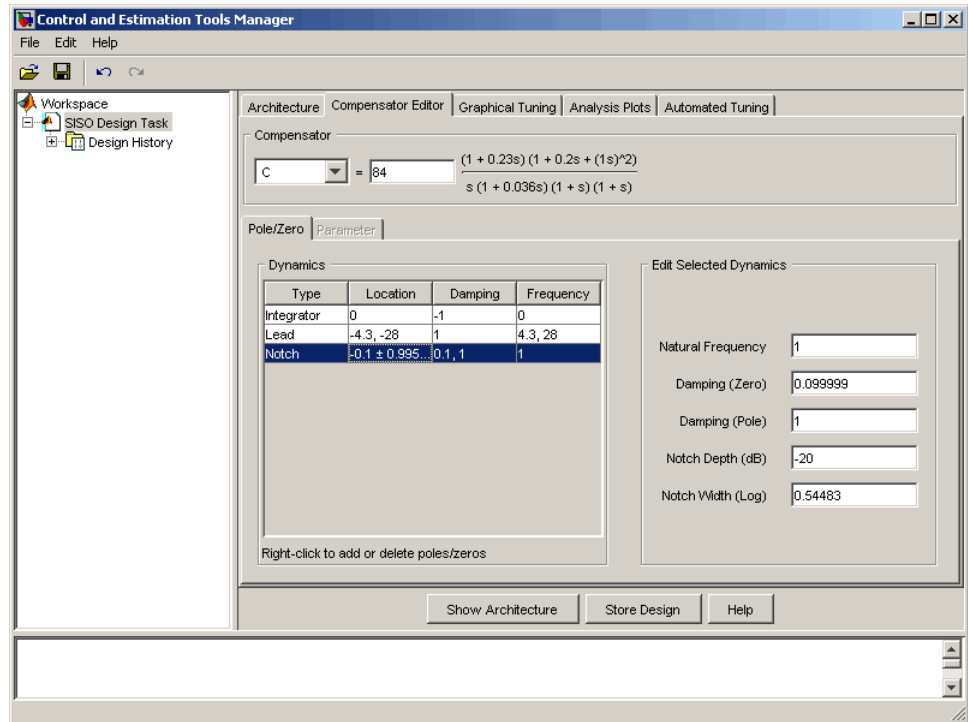
Adding a Notch Filter

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.



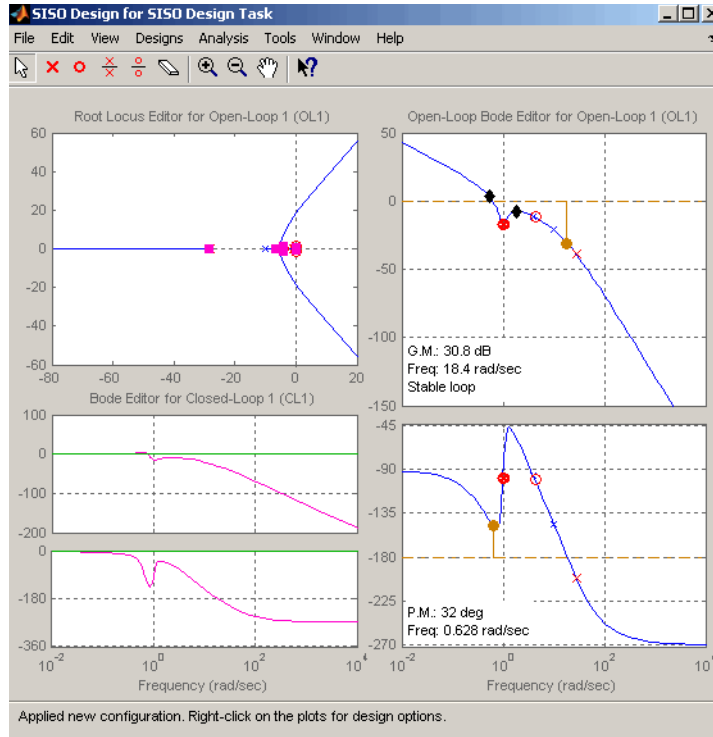
Adding a Notch Filter with the Dynamics Right-Click Menu

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

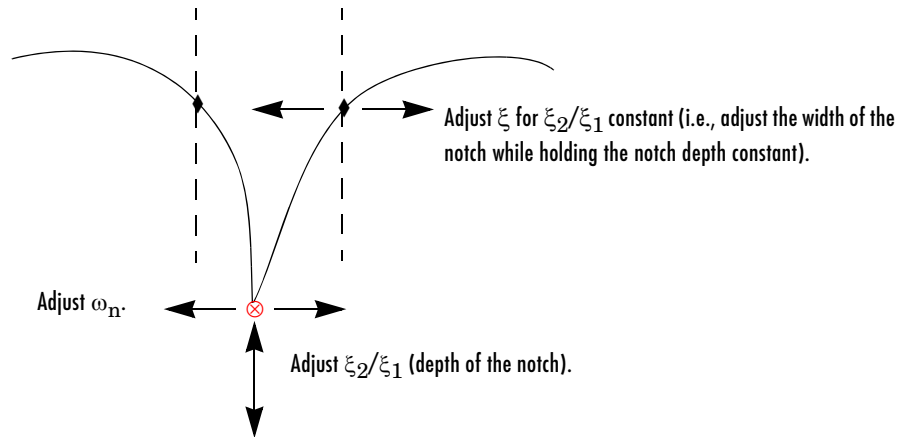
To see the notch filter parameters in more detail, click the Zoom In  icon on the Graphical Tuning window. In the Open-Loop Bode Editor, press the left mouse button and drag your mouse to draw a box around the notch filter. When you release the mouse, the Graphical Tuning window will zoom in on the selected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

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

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

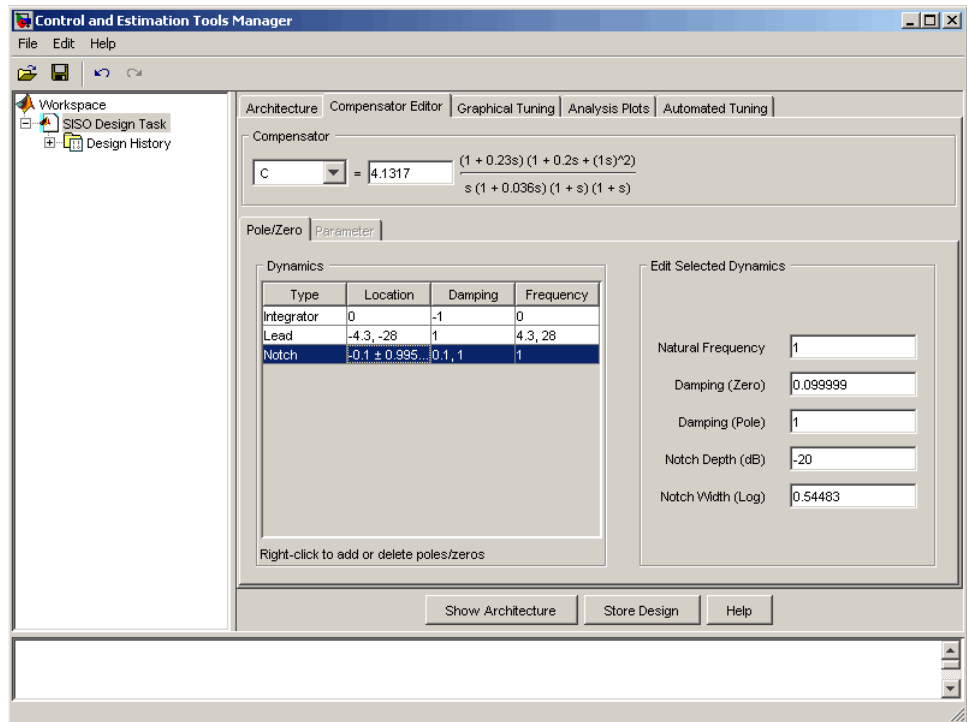
The three adjustable parameters are ξ_1 , ξ_2 , and ω_n . The ratio of ξ_2/ξ_1 sets the depth of the notch, and ω_n is the natural frequency of the notch.

This diagram shows how moving the red \otimes and black diamonds changes these parameters, and hence the transfer function of the notch filter.



A Close Look at Notch Filter Parameters

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



Editing Notch Filter Parameters

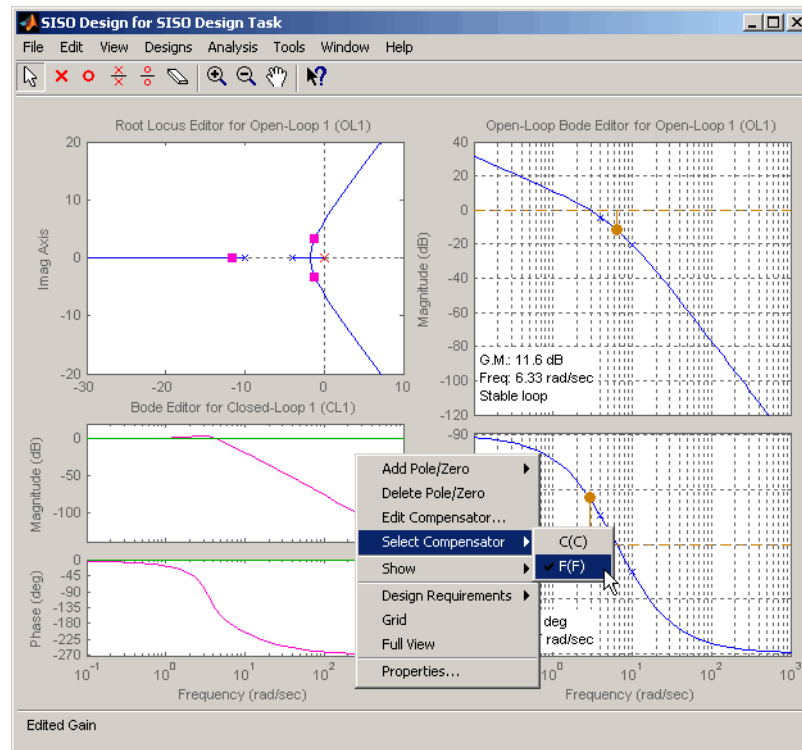
Modifying a Prefilter

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

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

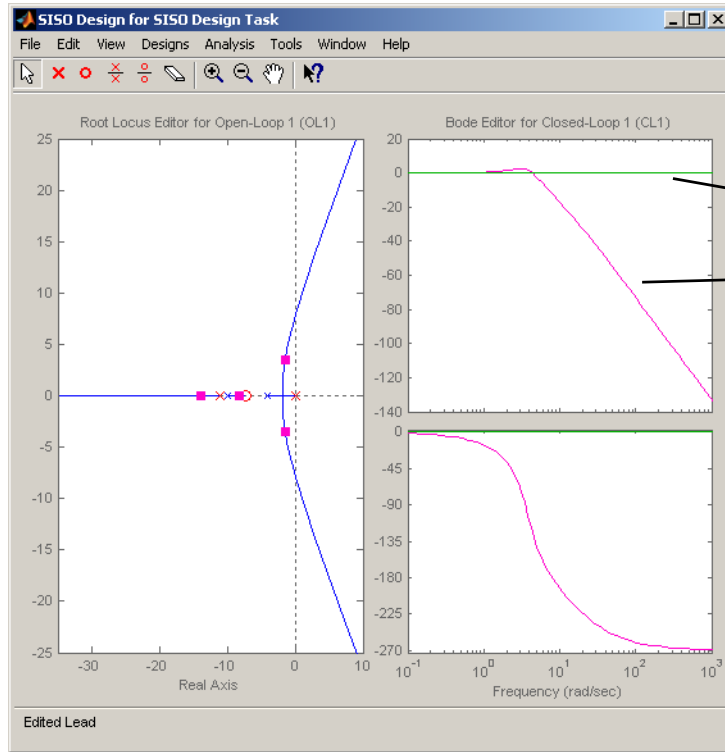
A common prefilter is a simple lowpass filter that reduces noise in the input signal.

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



Selecting the Prefilter in the Graphical Tuning Window

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

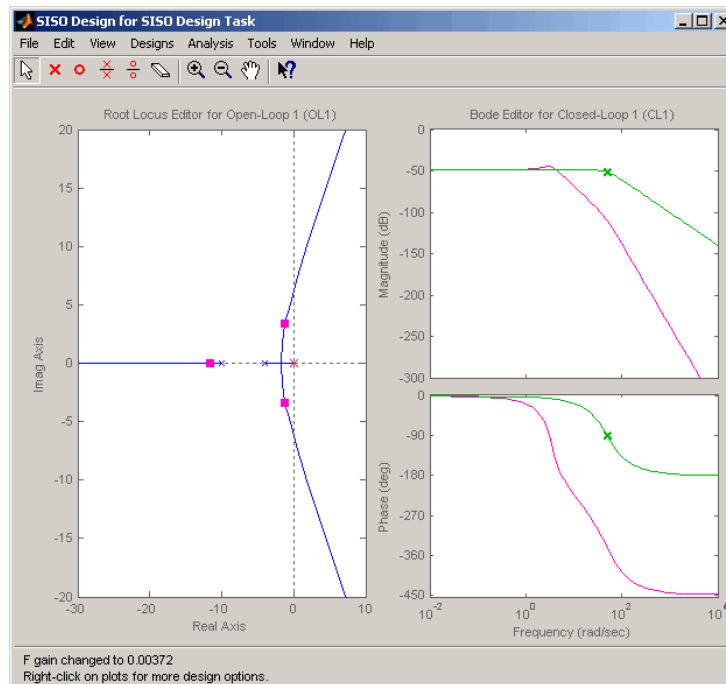


The green curve is the prefilter Bode magnitude.
The magenta curve is the closed-loop response from the prefilter input to the plant output.

Prefilter Bode Diagram

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

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



Adding a Complex Pair of Poles to the Prefilter Bode Diagram

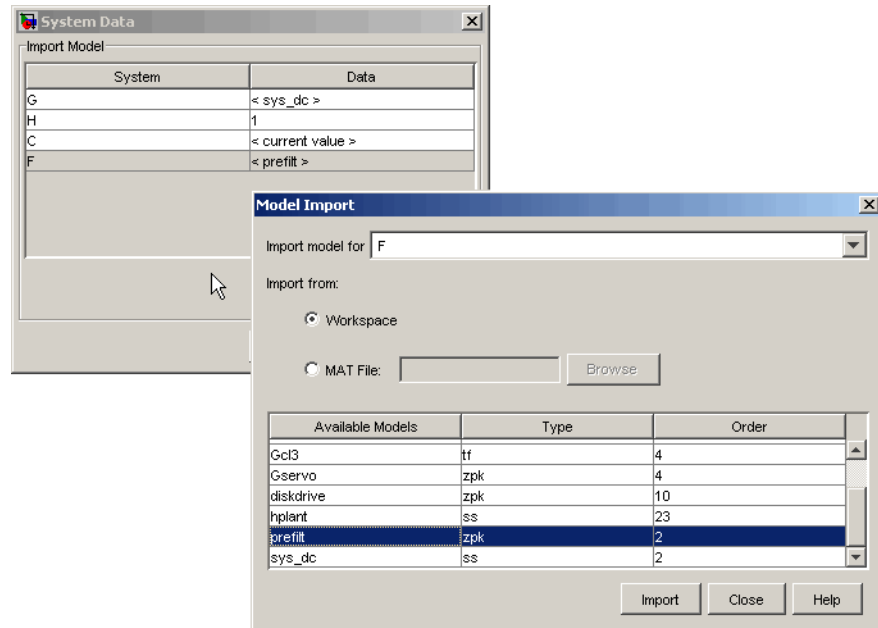
By default, the damping ratio of the complex pair is 1.0, which means that there are two real-valued poles at about -50 rad/s. The green curve, which represents the prefilter Bode response, shows the -3 dB point for the roll-off is at about 50 rad/s. The magenta curve, which represents the closed-loop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

Importing a Prefilter

An alternative approach is to design a prefilter using Control System Toolbox commands like `ss` or `tf` and importing 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

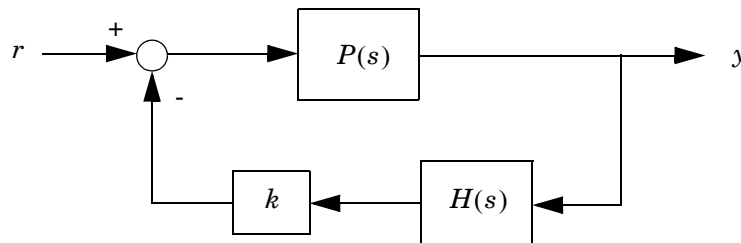
Select `prefilt` from the **Available Models** list and click **Import** to import the prefilter model. Click **Close** to close the Import Model dialog box. After you have imported the prefilter model, you can modify it using the same methods as described in this chapter for compensator design.

Root Locus Design

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.

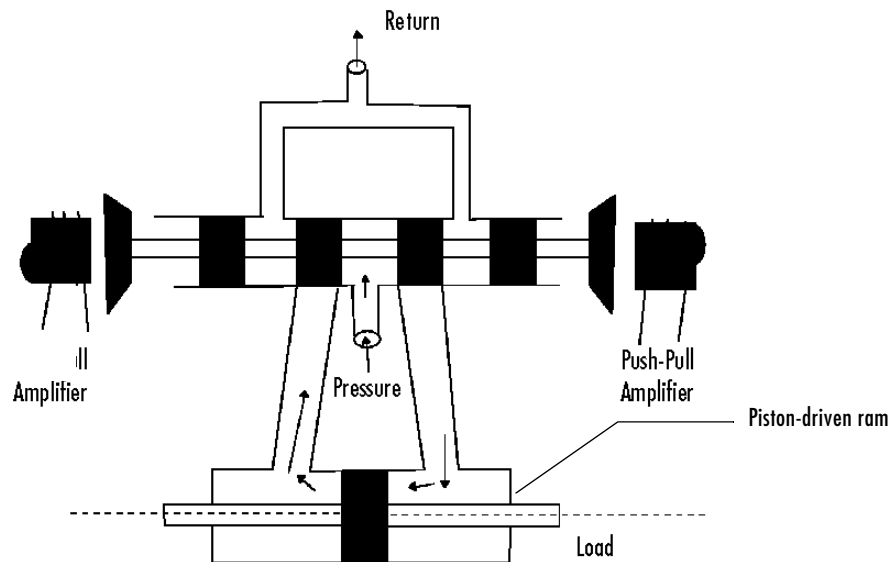
The SISO Design Tool's root locus and Bode diagram design tools provide complementary perspectives on the same design issues; each perspective offers insight into the design process. Because the SISO Design Tool shows both root locus and Bode diagrams, you can also choose to combine elements of both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

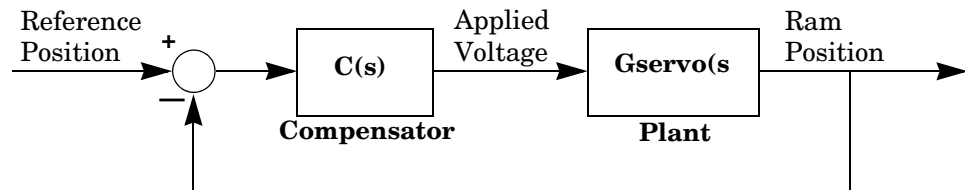
This figure shows a schematic of this servomechanism.



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%.

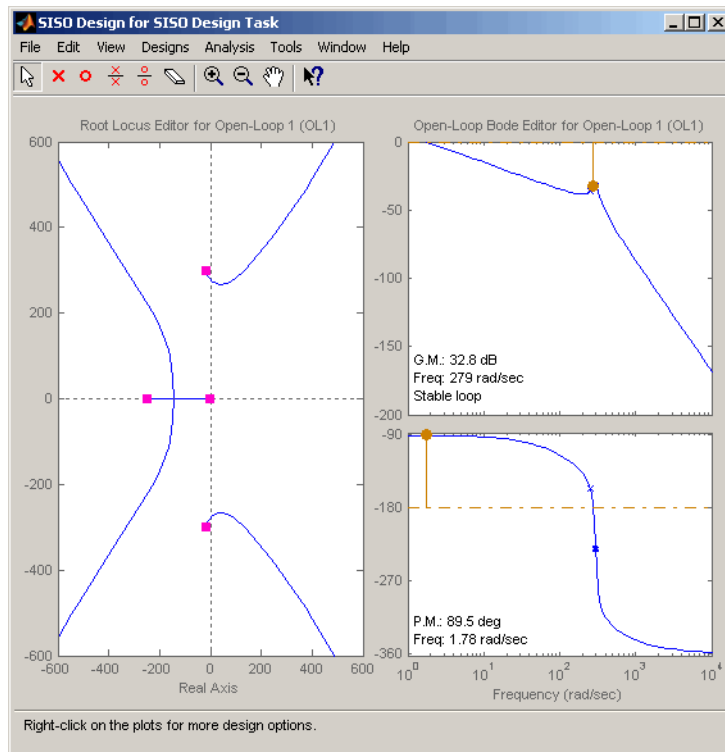
The remainder of this section discusses how to use the SISO Design Tool to design a controller to meet these specifications.

Opening the SISO Design Tool

Open the SISO Design Tool and import the model by typing


```
sisotool(Gservo)
```

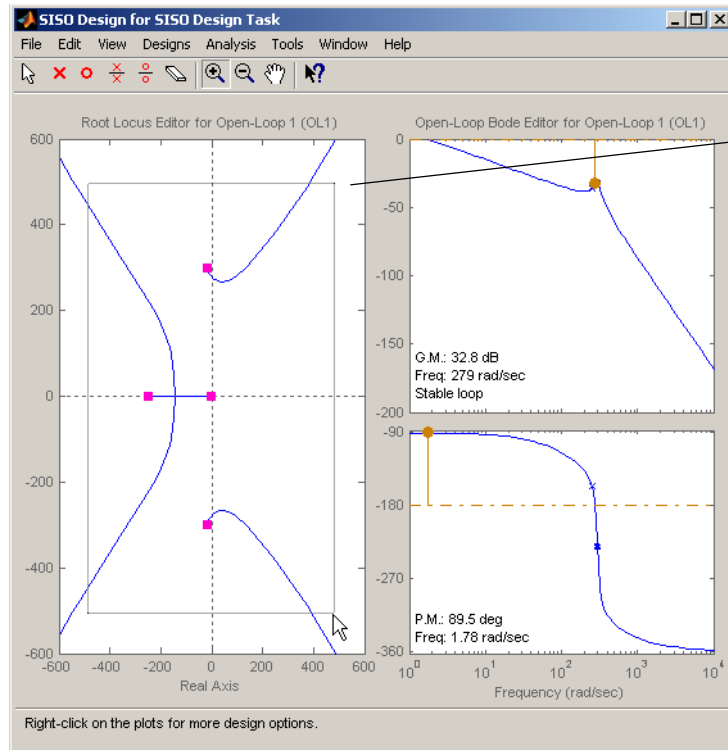
at the MATLAB prompt. This opens the **SISO Design Task** node in the Control and Estimation Tools Manager and the Graphical Tuning window with the servomechanism plant imported.



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

Zooming

Click the Zoom In  icon in the Graphical Tuning window. Press and hold the mouse's left button and drag the mouse to select a region for zooming. For this example, reduce the root locus region to about -500 to 500 in both the x - and y -axes. This figure illustrates the zooming in process.



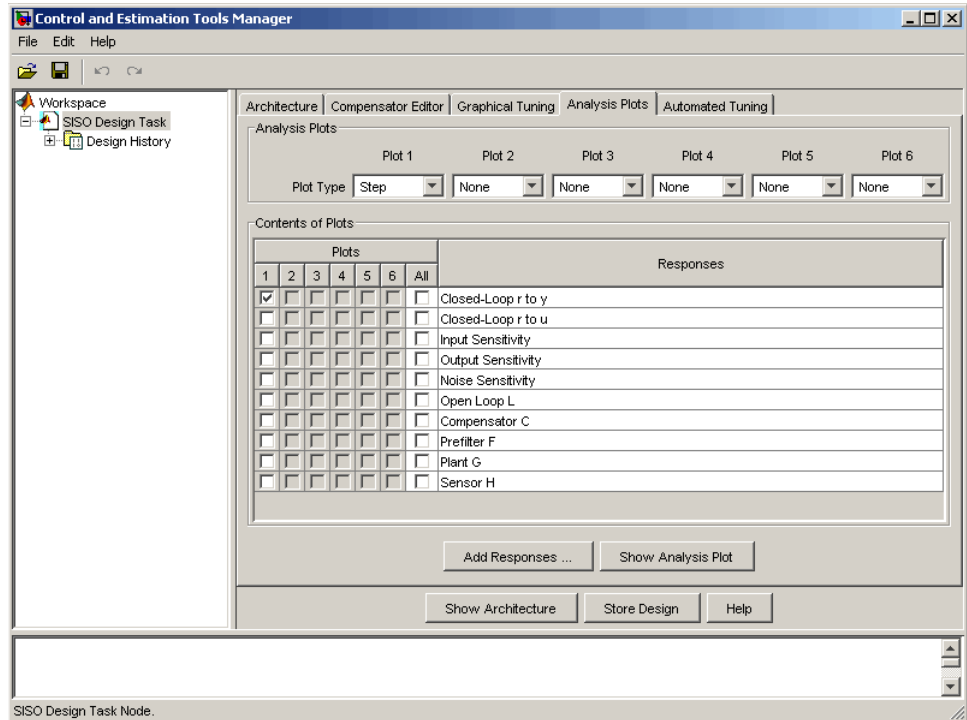
Hold the left mouse button down and drag the mouse to select a rectangular region for zooming in. When you release the mouse button, the Graphical Tuning window replots the root locus with the new axis boundaries.

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

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

Zooming In on a Region in the Root Locus Plot

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

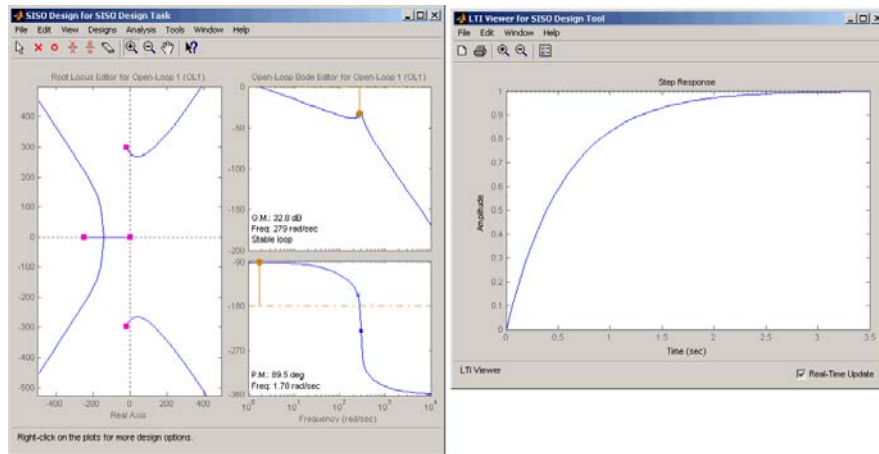


Analysis Plots Loop Response Selection

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

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

You should now see both the Graphical Tuning window and the LTI Viewer.



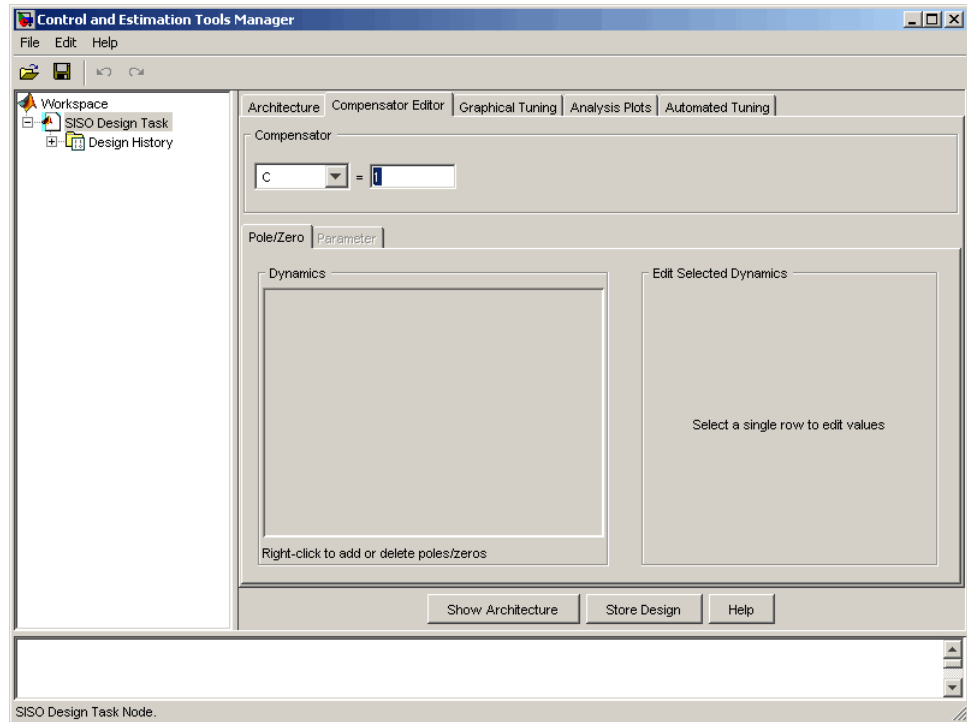
Graphical Tuning Window and Associated LTI Viewer for the Electrohydraulic Servomechanism

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 sections 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 4-43.

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.

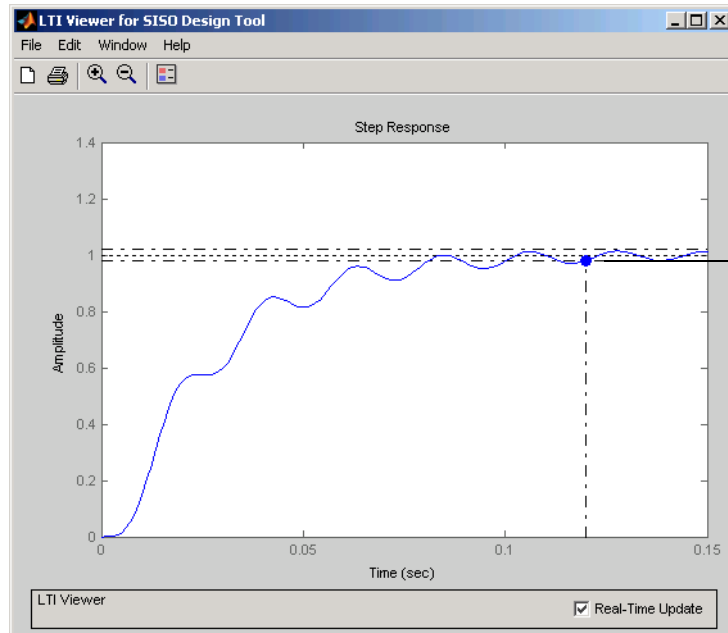
Experiment with different gains and view the closed-loop response in the associated LTI Viewer.

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 $C(s) = 20$

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 4-49 shows how to design a compensator so that you meet the required specifications.

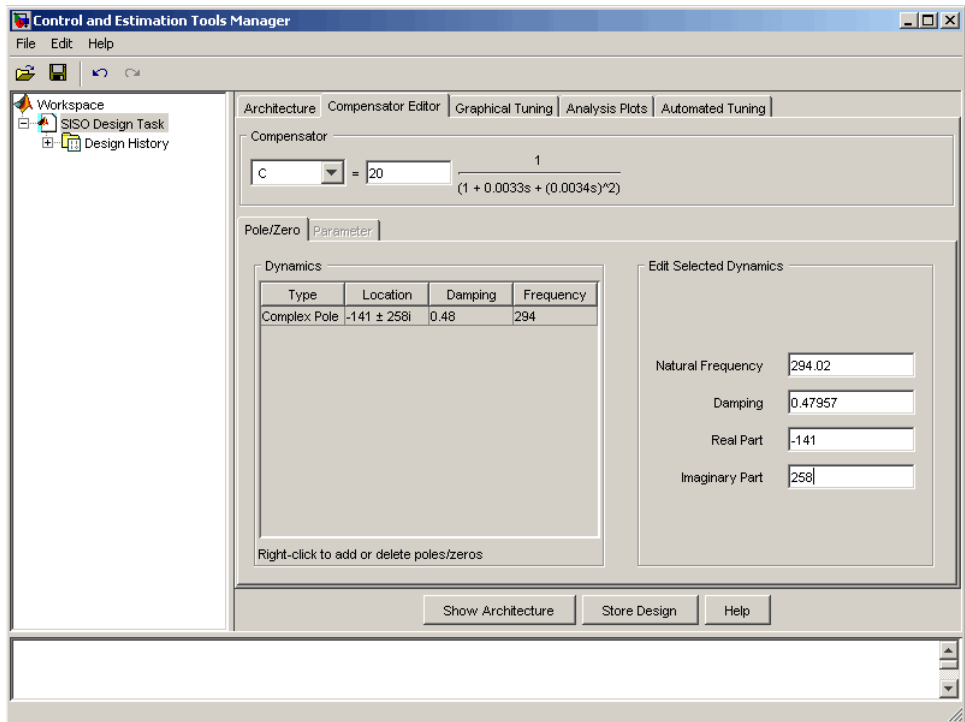
Adding Poles and Zeros to the Compensator

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 4-53.



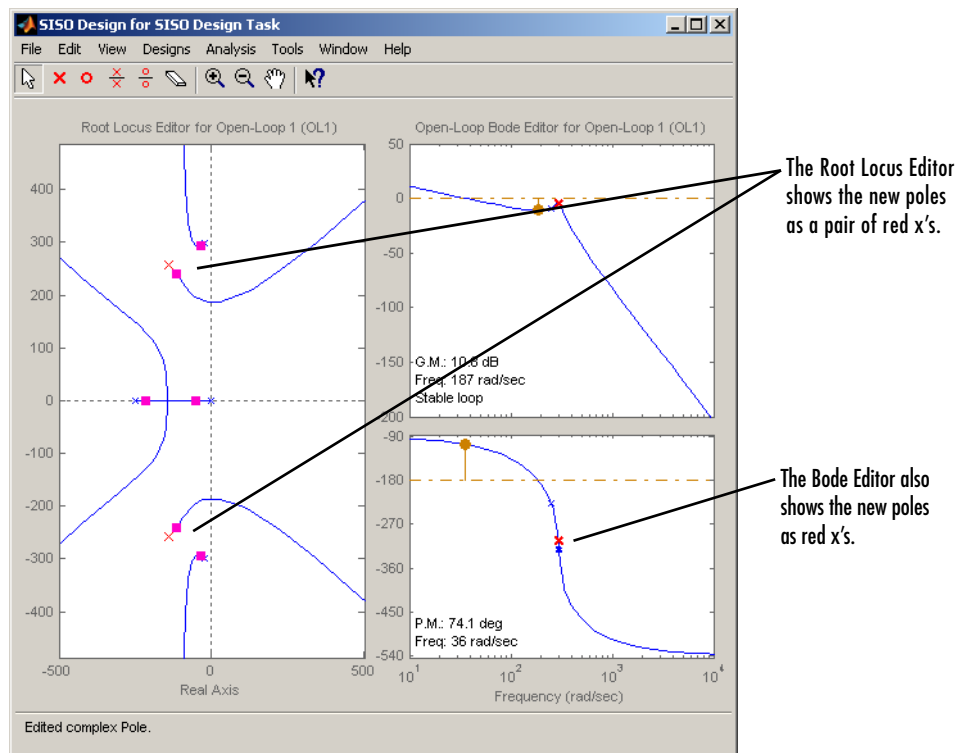
Adding a Complex Pair of Poles to the Compensator on the Compensator Editor Page

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.

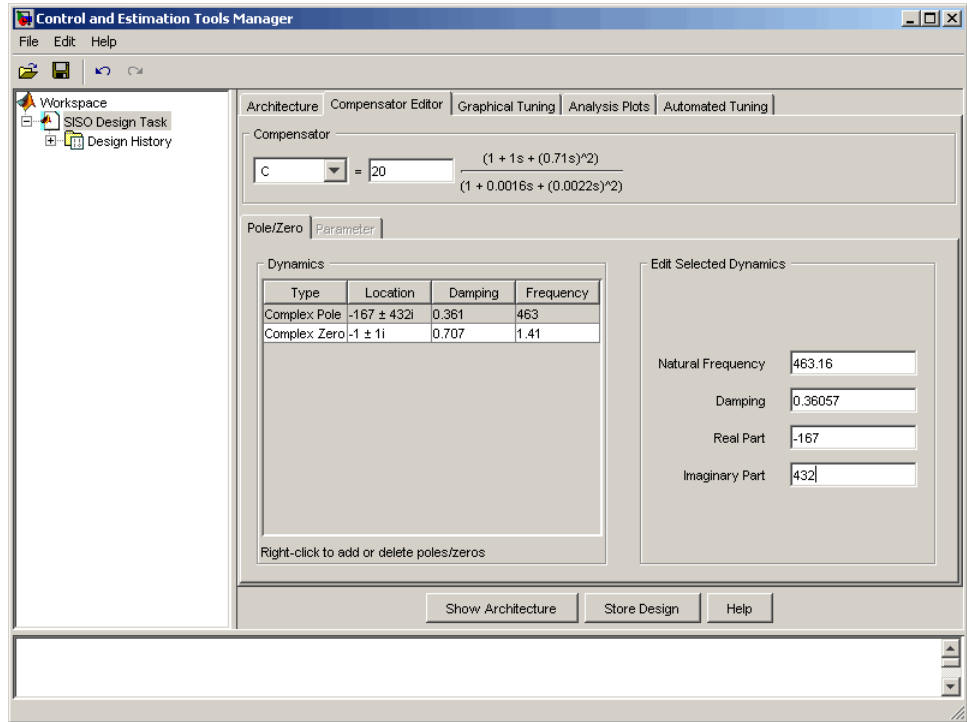


Result of Adding a Complex Pair of Poles to the Compensator

Adding Zeros to the Compensator

The procedure for adding zeros to the compensator is exactly the same. You 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 4-53.

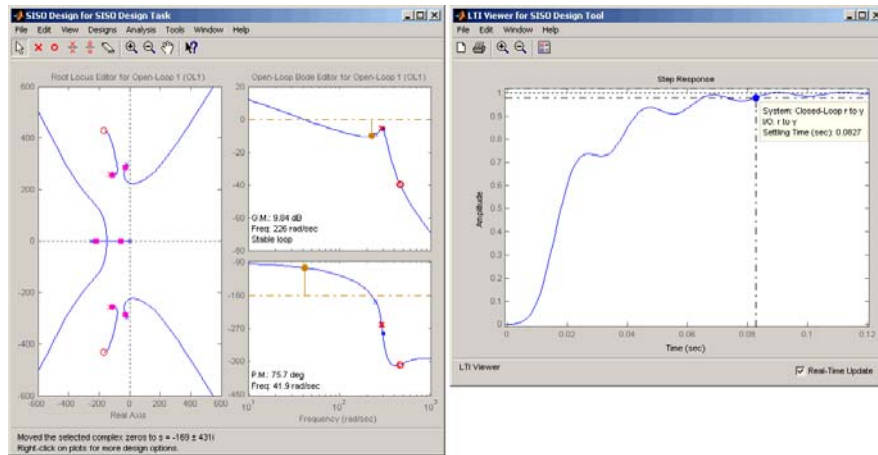


Adding Complex Zeros to the Compensator on the Compensator Editor Page

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

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

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



Electrohydraulic Servomechanism Example with Complex Zeros Added

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

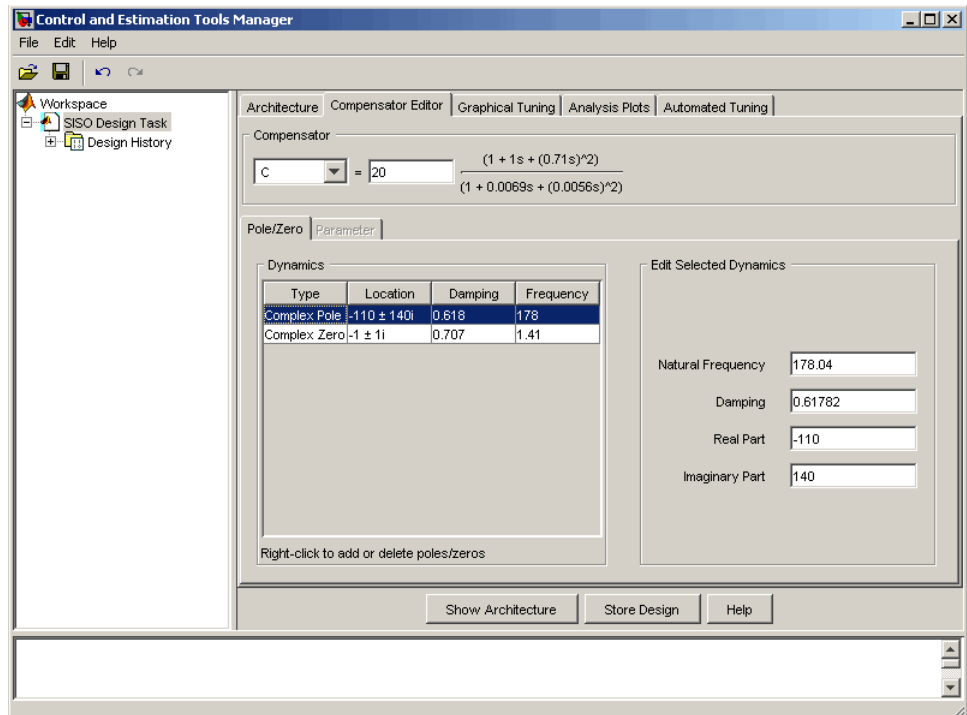
As you can see, the compensator design process can involve some trial and error. You can try dragging the compensator poles, compensator zeros, or the closed-loop poles around the root locus until you meet the design criteria.

“Editing Compensator Pole and Zero Locations” on page 4-53, 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 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:

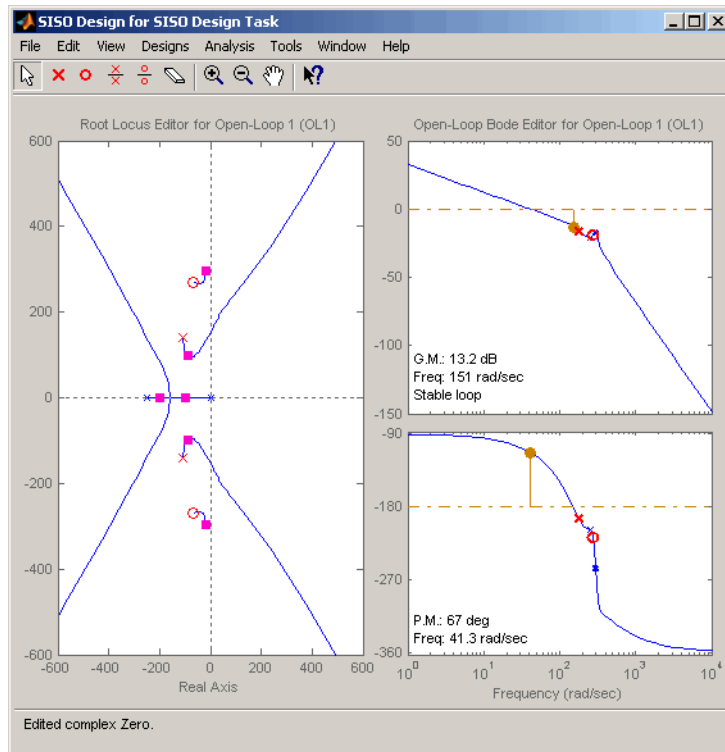
- 1 Select the compensator to edit in the list box to the left of the equal sign in the **Compensator** area.
- 2 Enter the gain value in the text box to the right of the equal sign in the **Compensator** area.

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.

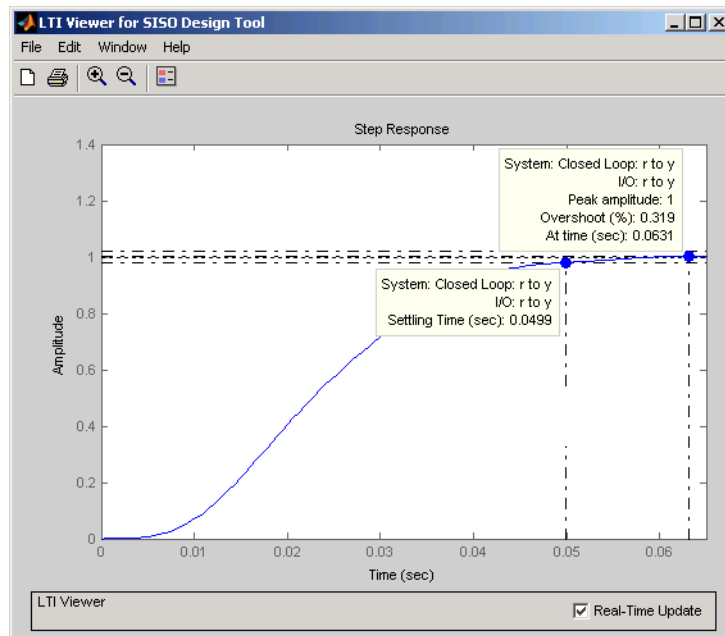
For this example, edit the poles to be at $-110 \pm 140i$ and the zeros at $-70 \pm 270i$. Set the compensator gain to 23.3.

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.



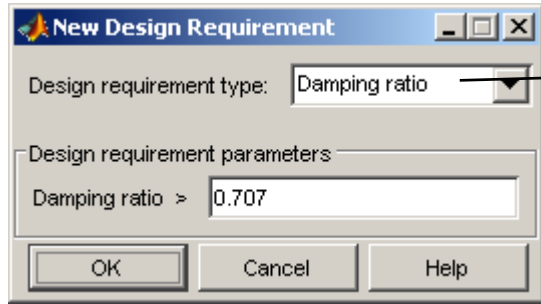
Closed-Loop Step Response for the Final Design of the Electrohydraulic Servomechanism Example

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

Viewing Damping Ratios

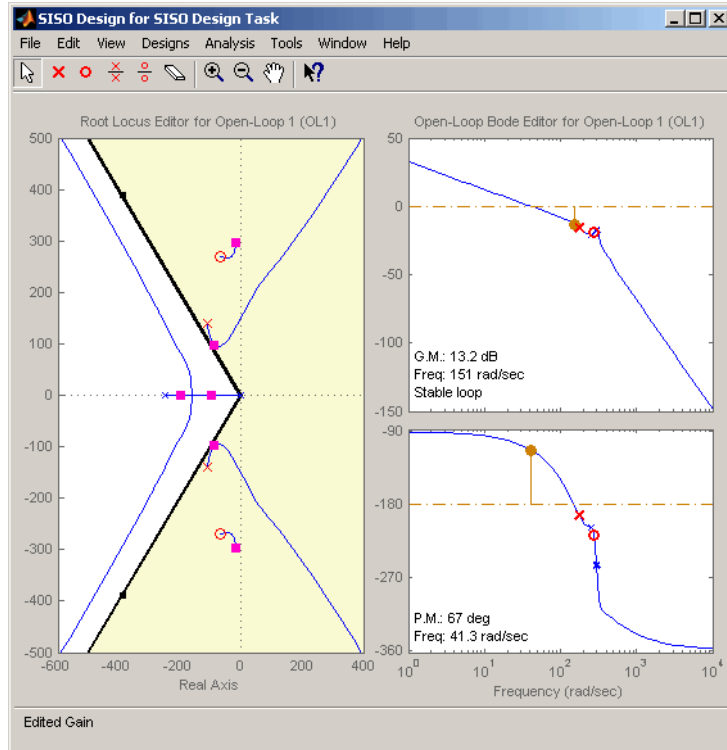
The Graphical Tuning window provides design requirements that can make it easier to meet design specifications. If you want to place, for example, a pair of complex poles on your diagram at a particular damping ratio, select **Design Requirements > New** from the right-click menu in the root locus graph.

This opens the **New Design Requirement** dialog box.



Select **Damping ratio** from this list. The default value 0.707 sets lines on the root locus corresponding to a 0.707 damping ratio.

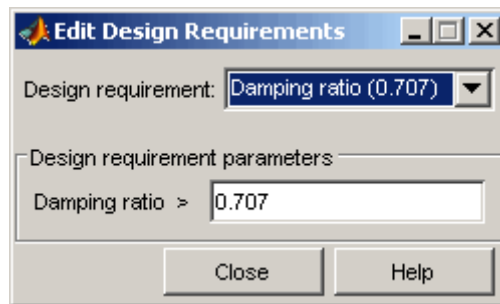
Applying damping ratio requirements to the root locus plot results in a pair of shaded rays at the desired slope, as this figure shows.



Root Locus with 0.707 Damping Ratio Lines

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

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



Specify the new damping ratio requirement in this dialog box.

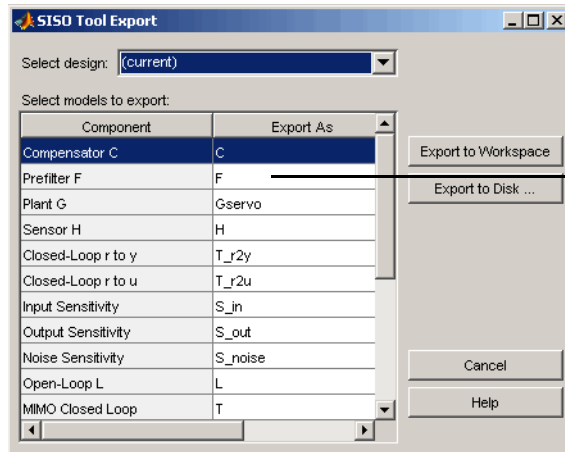
An alternate way to adjust a requirement is to left-click the requirement itself to select it. Two black squares appear on the requirement when it is selected. You can then drag it with your mouse anywhere in the plot region.

If you want to add a different set of requirements, for example, a settling time requirement, again select **Design Requirements > Edit** 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.

Exporting the Compensator and Models

Now that you have successfully designed your compensator, you may want to save your design parameters for future implementation. You can do this by selecting **Export** from the **File** menu on the **SISO Design Task** node on the Control and Estimation Tools Manager or from the **File** menu on the Graphical Tuning window. Both methods open the SISO Tool Export dialog box.



Double-click any cell in the **Export As** column to edit the name for export.

You may leave the model list as all components or you may select the components for a particular design by selecting the design name from the pull-down list.

The variables listed in the **Export As** column are either previously named by you (in the System Data dialog box) or have default names. To export your compensator to the workspace:

- 1 Select **Compensator C** in the **Component** column. If you want to change the export name, double-click in the cell for **Compensator C**.
- 2 Click **Export to Workspace**.

If you go to the MATLAB prompt and type

```
who
```

the compensator is now in the workspace, in the variable named **C**.

Type

```
C
```

to see that this variable is stored in **zpk** format.

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

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

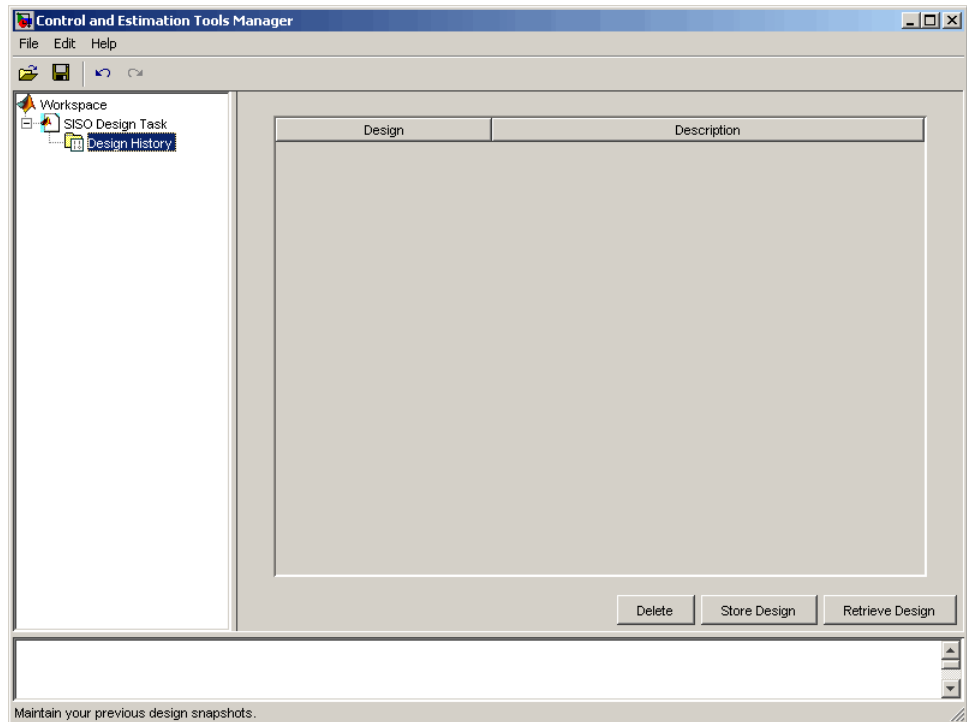


You can save your models as MAT-files in any directory you want. The default name for the MAT-file is the name of your original model; you can change the name to anything you want. If you save multiple components, they are stored in a single MAT-file.

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.

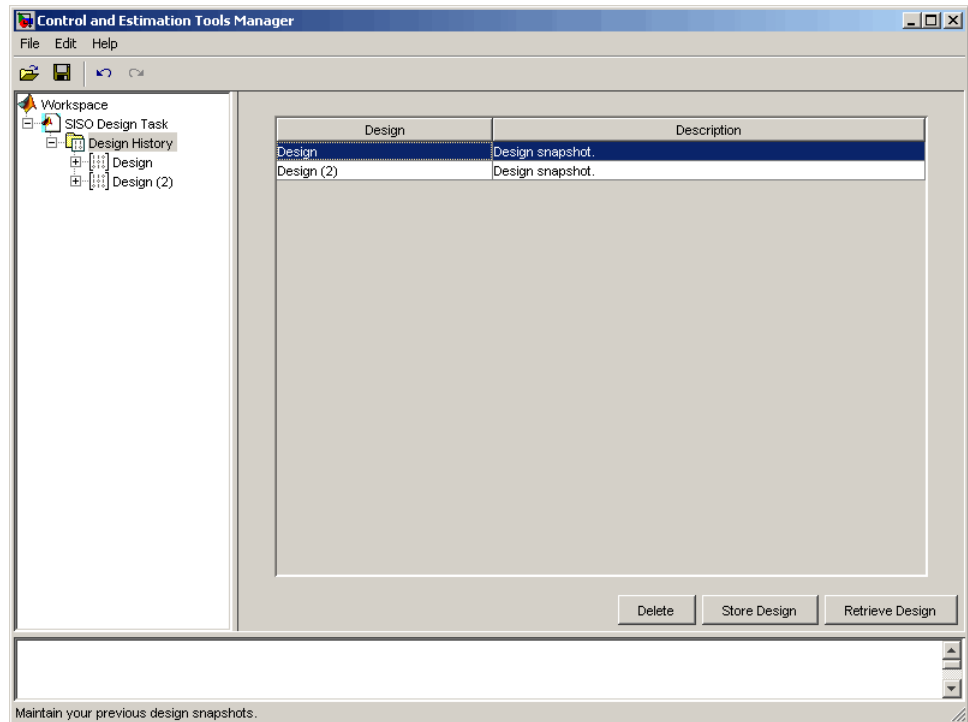
Alternatively, you can select **Store/Retrieve** from the **Designs** menu in the Graphical Tuning window. Using either method, the following **Design History** page opens.



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.

The screenshot shows the 'Control and Estimation Tools Manager' window. The workspace tree on the left contains 'SISO Design Task', 'Design History', and 'Design'. The main area displays the 'Design Snapshot Summary - Design' for 'Design Configuration #1'. It features a block diagram with blocks F (green), C (red), G (yellow), and H (yellow). Block F is the forward path, C is the controller, G is the plant, and H is the feedback path. Below the diagram, the 'Tunable Elements' section lists parameters for blocks C and F.

Design Configuration #1

Tunable Elements

C:

Parameter	Value
Gain	2.77541089206017
Zeros	$[-70+i*270; -70-i*270]$
Poles	$[-110+i*140; -110-i*140]$

F:

Parameter	Value
Gain	1

Retrieve Design

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**.

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 section repeats the DC motor compensator design presented in “Example: DC Motor” on page 4-17, 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

$$\begin{array}{l} \text{Transfer function:} \\ 1.5 \\ \text{-----} \\ s^2 + 14 s + 40.02 \end{array}$$

This example uses the design criteria specified in “Design Specifications” on page 4-43:

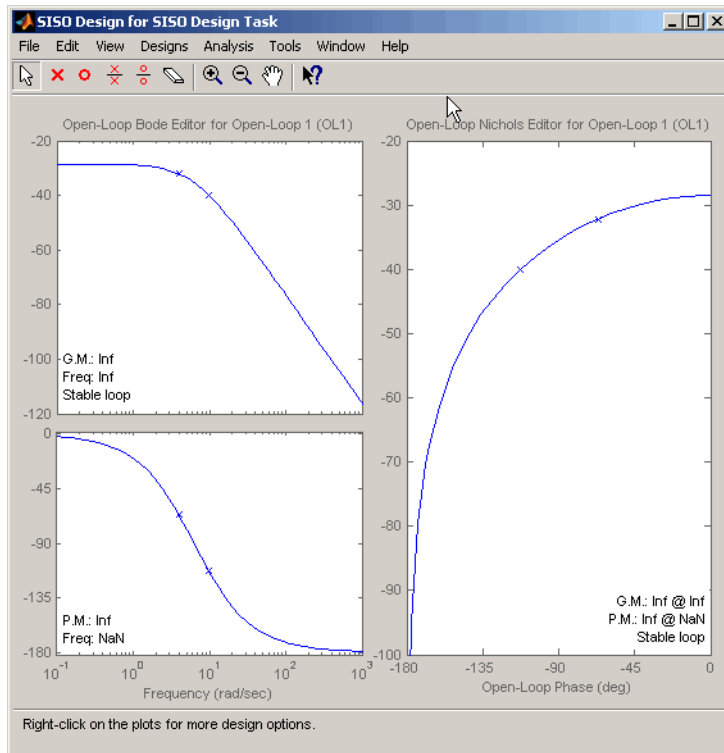
- Rise time of less than 0.5 second
- Steady-state error of less than 5%
- Overshoot of less than 10%
- Gain margin greater than 20 dB
- Phase margin greater than 40 degrees

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.

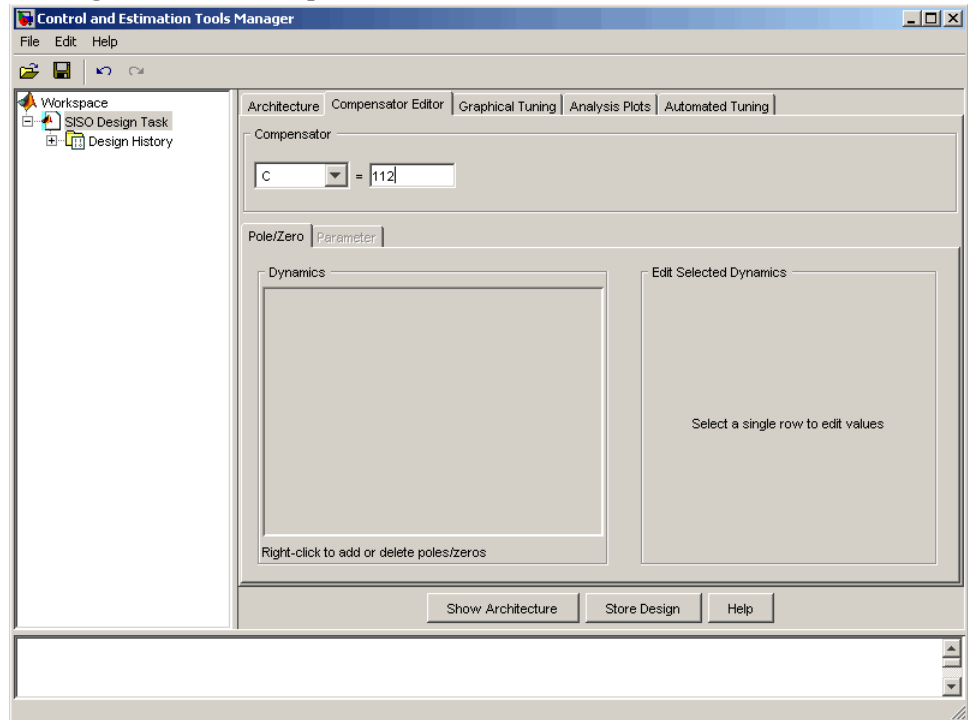


Graphical Tuning Window with a Bode Diagram and a Nichols Plot

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**.

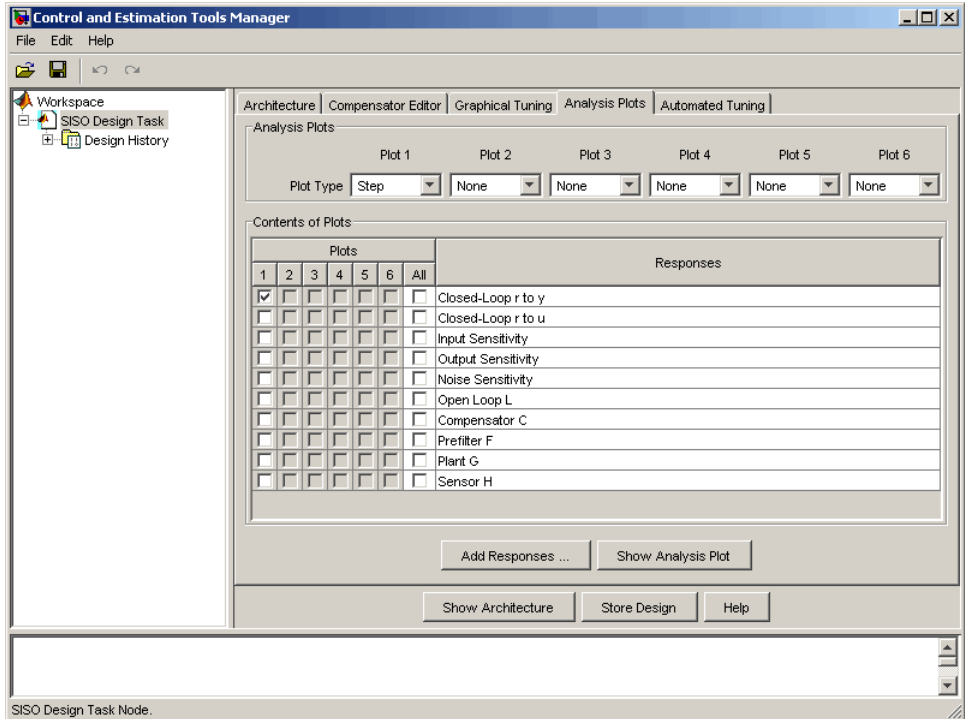


Adjusting Compensator Gain in the Compensator Editor Page

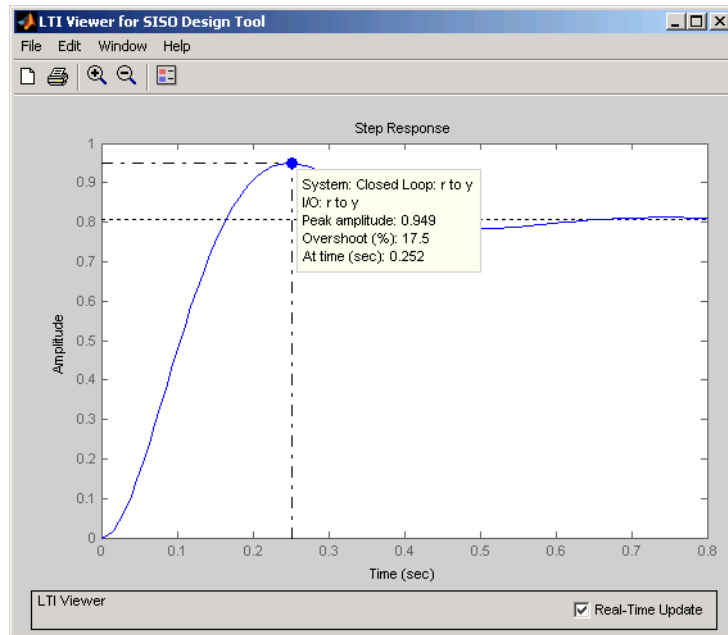
In this example, the new gain is 112.

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.

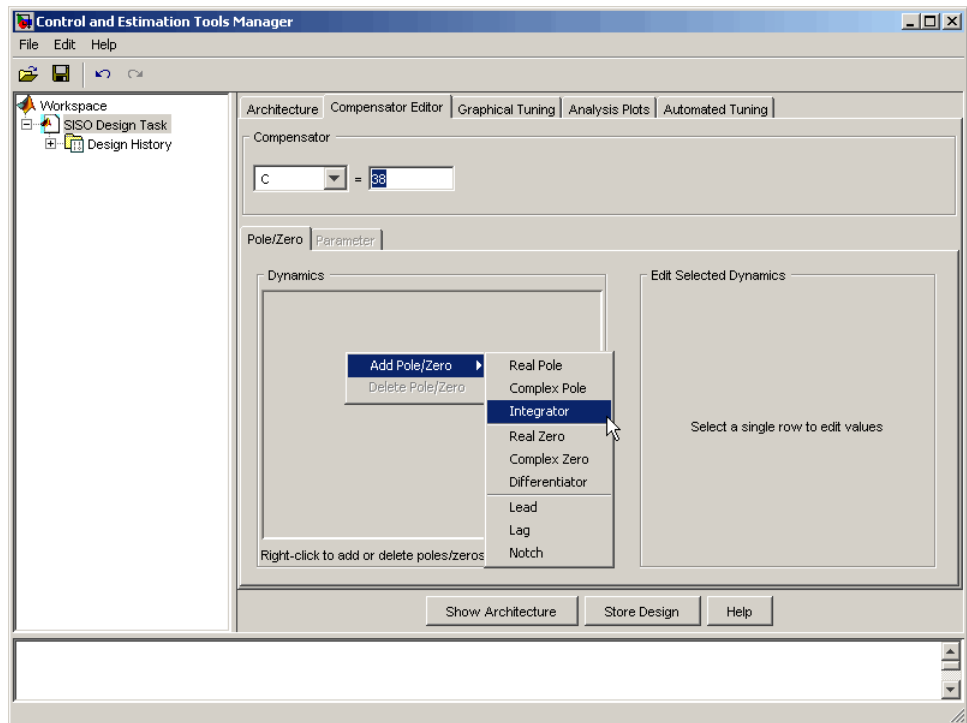
Adding an Integrator

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.



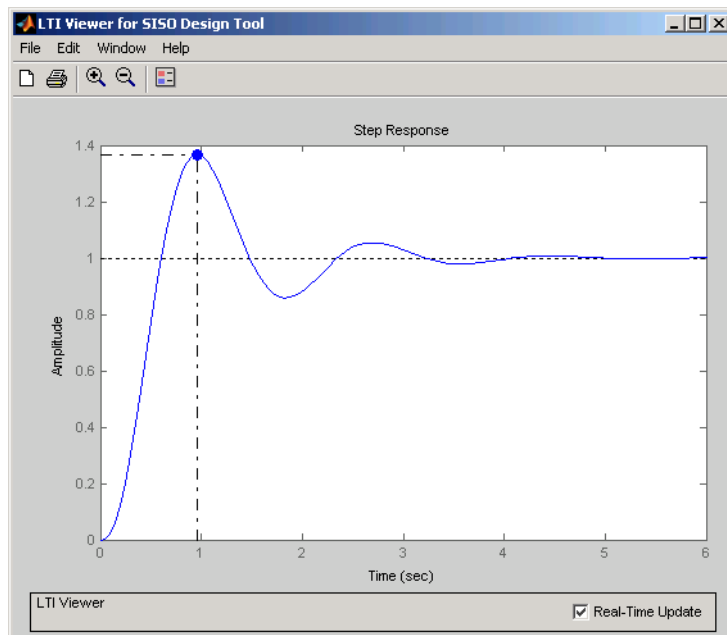
Adding an Integrator in the Dynamics Table

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

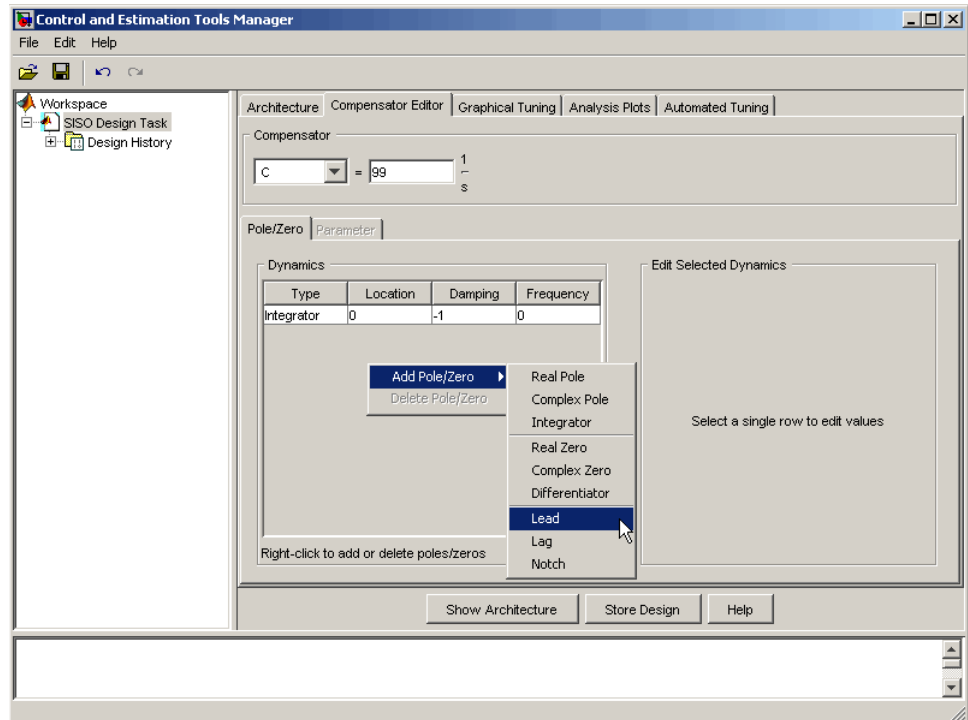
The steady-state value is now 1, which means the steady-state error has been eliminated, but the overshoot is 34% and the rise time is about 0.7 second. You must do more work to create a good design.

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:

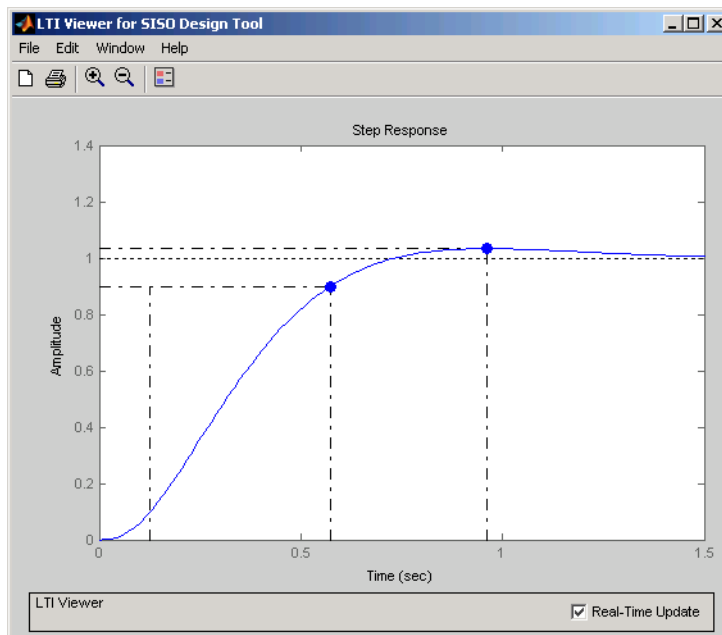
- 1 Click the **Compensator Editor** tab to open the **Compensator Editor** page.
- 2 Right-click in the **Dynamics** table and select **Add Pole/Zero > Lead**.

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.



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

The SISO Design Tool simplifies the task of designing and tuning compensators. There are four different 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
- LQC synthesis

This section adds a PID controller to a DC Motor example. For a discussion of the other methods, see “Automatic Tuning”.

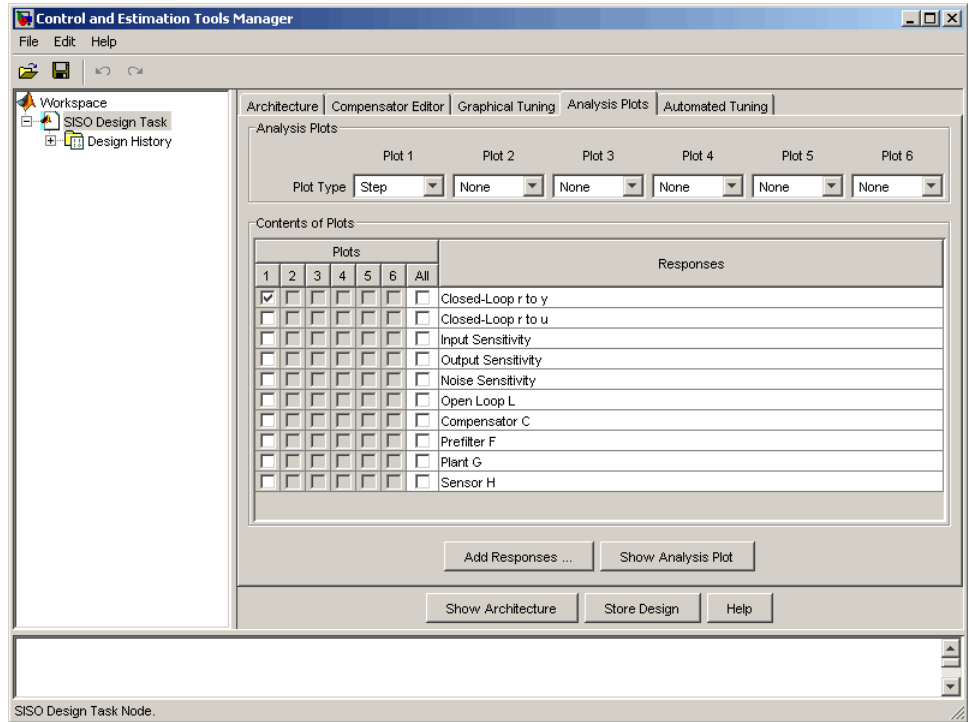
Loading and Displaying the DC Motor Example for Automated Tuning

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

```
sisotool(sys_dc)
```

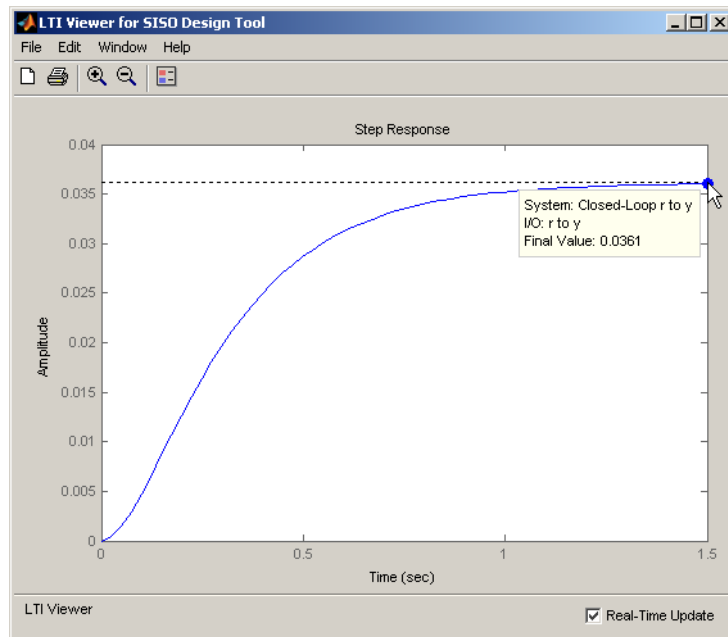
at the MATLAB prompt. This opens the **SISO Design Task** node on the Control and Estimation Tools Manager and the Graphical Tuning window with `sys_dc` loaded.

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 next, 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

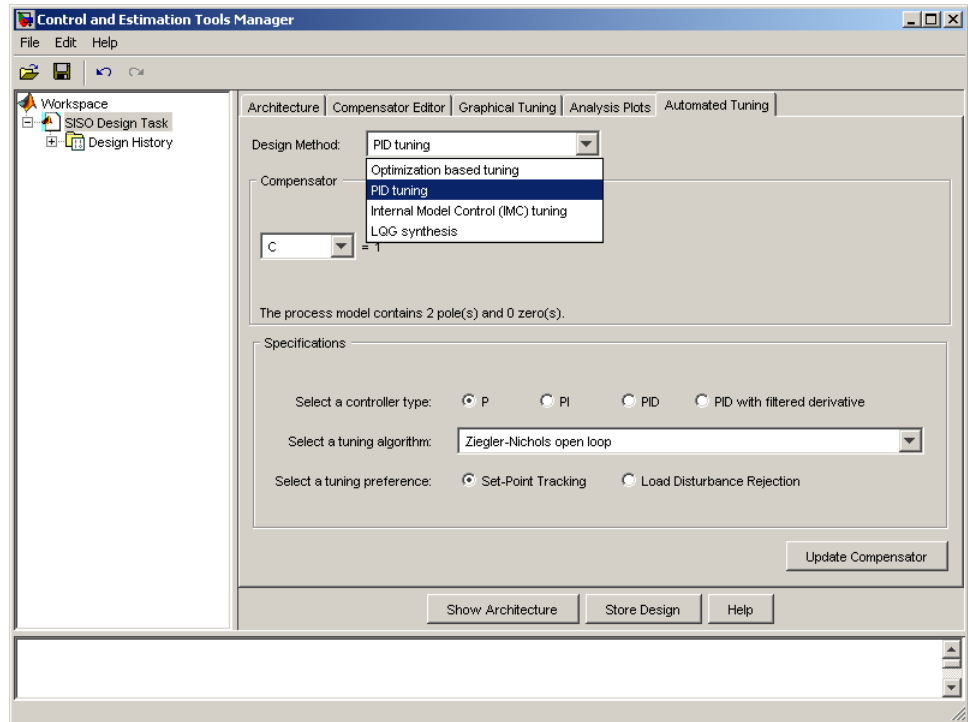
In the LTI Viewer, note that by default, the compensator is 1 and unit negative feedback is used (see “Architecture” in “Using the SISO Design Tool and LTI Viewer”). When a unit step is applied to the setpoint change, the steady state value of the system output is 0.0361, which is far from the setpoint, and its rise time is 0.589.



Step Response When Compensator = 1

Applying Automated PID Tuning

- 1 Click the **Automated Tuning** tab to open the **Automated Tuning** page.
- 2 Select PID tuning from the **Design Method** list.
- 3 Leave C as the default compensator.



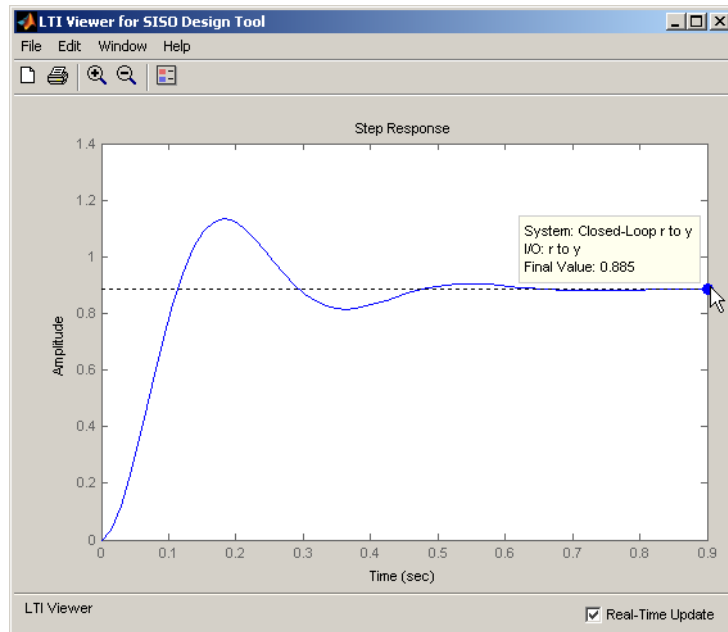
Automated Tuning Page – PID Tuning Selected

Designing a Proportional-Only Controller

- 1 Click the **P** option button for proportional-only control ($c = K_p$).
- 2 Select the Ziegler-Nichols open loop tuning algorithm from the **Select a tuning algorithm** list.
- 3 Click the **Set-Point Tracking** option button (tuning preference is available only when Ziegler-Nichols open loop is selected).
- 4 Click **Update Compensator**. The compensator value is now 205.49.

Look at the LTI Viewer and 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 4-79.



LTI Viewer with PID Tuning Set to P

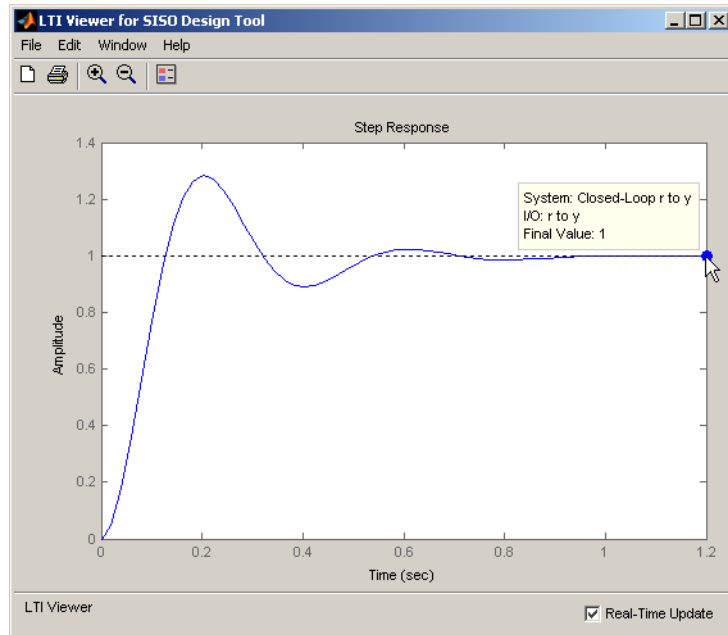
Designing a Proportional-Integral Controller

- 1 Click the **PI** option button for proportional-integral control

$$\left(c = K_p + \frac{K_I}{s} \right).$$
- 2 Select the Ziegler-Nichols open loop tuning algorithm from the **Select a tuning algorithm** list.
- 3 Click the **Set-Point Tracking** option button.
- 4 Click **Update Compensator**. Note the compensator value is now

$$487.64 \times \frac{(1 + 0.36s)}{s}.$$

The LTI Viewer is again updated with the application of PID automated tuning with PI, as shown here.



LTI Viewer with PID Tuning Set to PI

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.

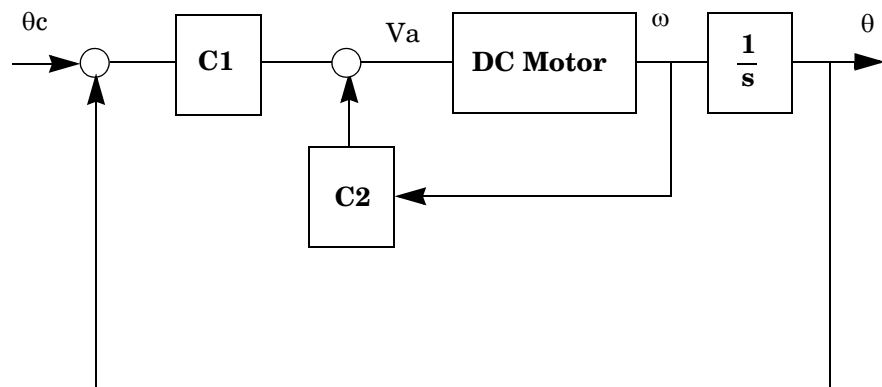
Multi-Loop Compensator Design

In 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.

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 4-81 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 to 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.

The steps in this process are:

- “Developing a Mathematical Model of the DC Motor” on page 4-82
- “Selecting the Architecture and Importing the Model” on page 4-83
- “Designing the Inner Loop” on page 4-86
- “Tuning the Outer Loop” on page 4-88
- “Validating the Design with the LTI Viewer for SISO Design” on page 4-91

Developing a Mathematical Model of the DC Motor

These are the relevant physical constants:

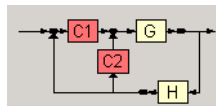
```
R=2.0           % Ohms
L = 0.5         % Henrys
Km=0.1; Kb = 0.1 % Torque and back emf constants
Kf= 0.2;       % Nms
J = 0.02       % kg.m^2/s^2
```

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
```

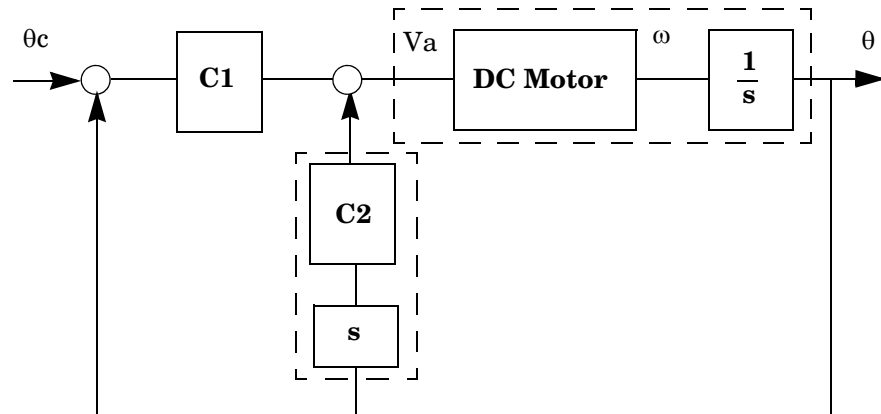
Adapting the Model to SISO Tool Architecture

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 4-81, 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 V_a 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
```

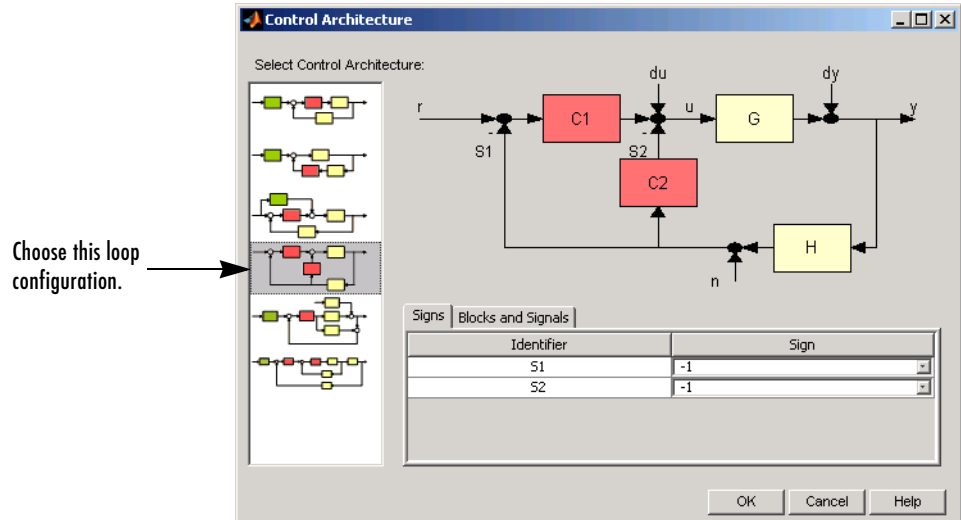
Selecting the Architecture and Importing the Model

Open the SISO Design Tool by typing

```
sisotool
```

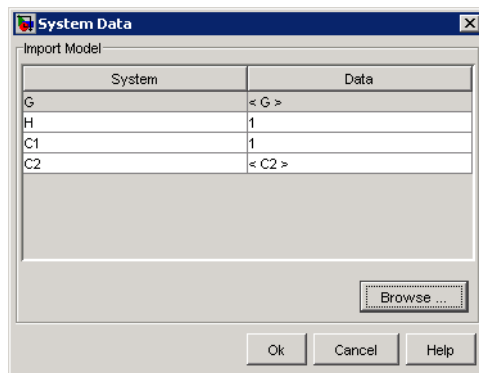
at the MATLAB prompt. Once the Controls & Estimation Tools Manager opens, click **Control Architecture** on the **Architecture** page. Select the

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.

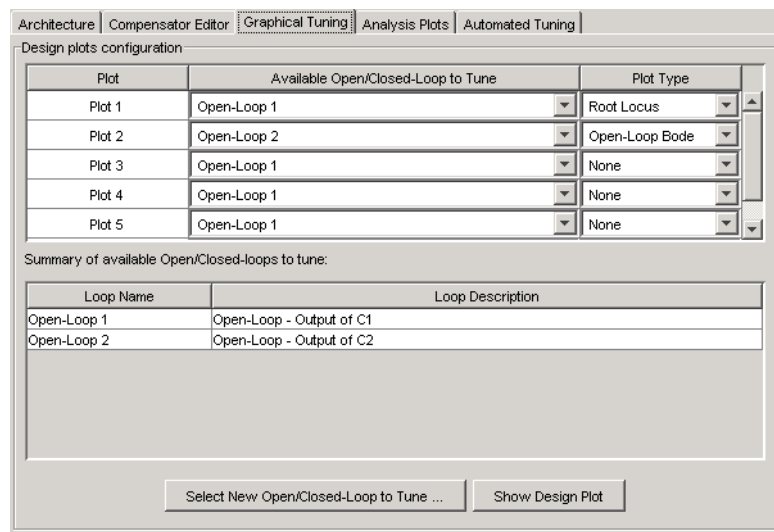


Selecting SISO Design Graphical Editor Views

Once you have selected the multi-loop architecture, click the **Graphical Tuning** tab. Set the plot types as follows:

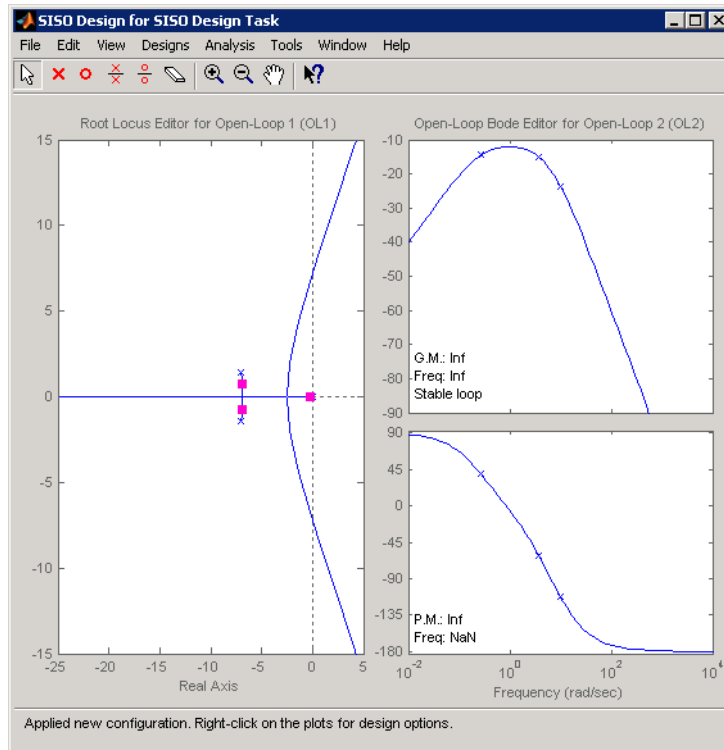
- 1 Open-Loop 1 to “Root-Locus”
- 2 Open-Loop 2 to “Open-Loop Bode”

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.

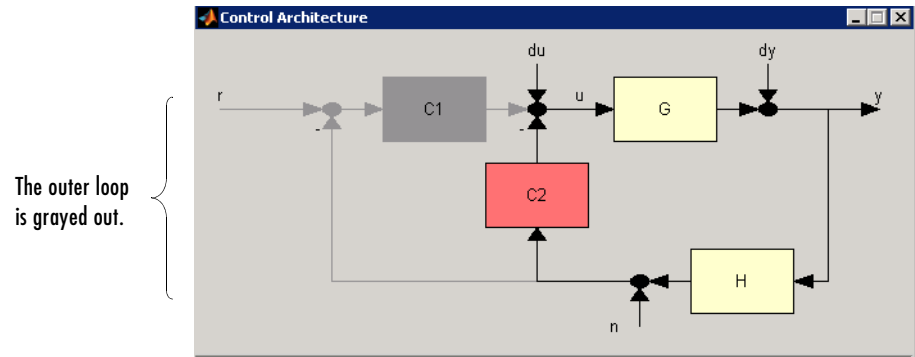


Designing the Inner Loop

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:

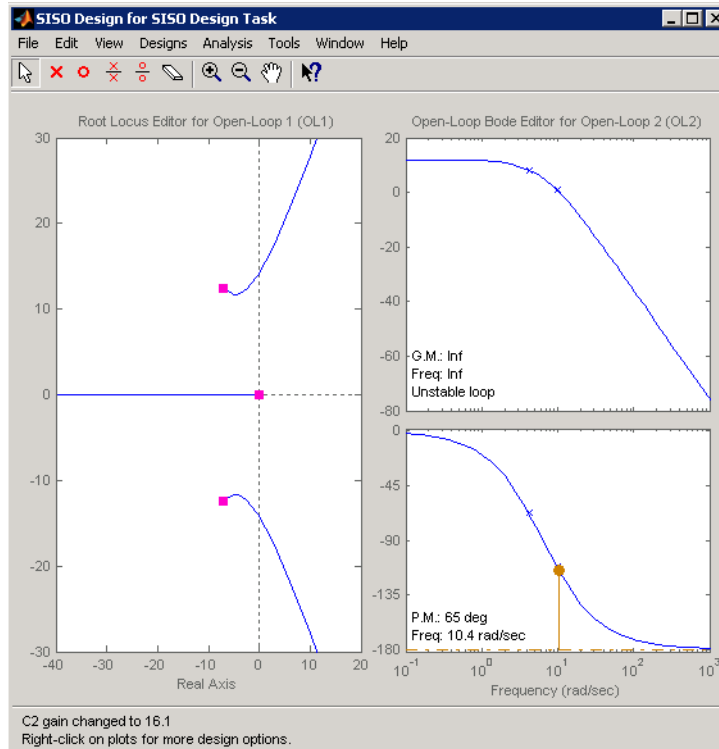
- 1 Click **Loop Configuration**. This opens the Open-Loop Configuration dialog box.
- 2 From the pull-down menu, select **Open-Loop Output of C2**.

- 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

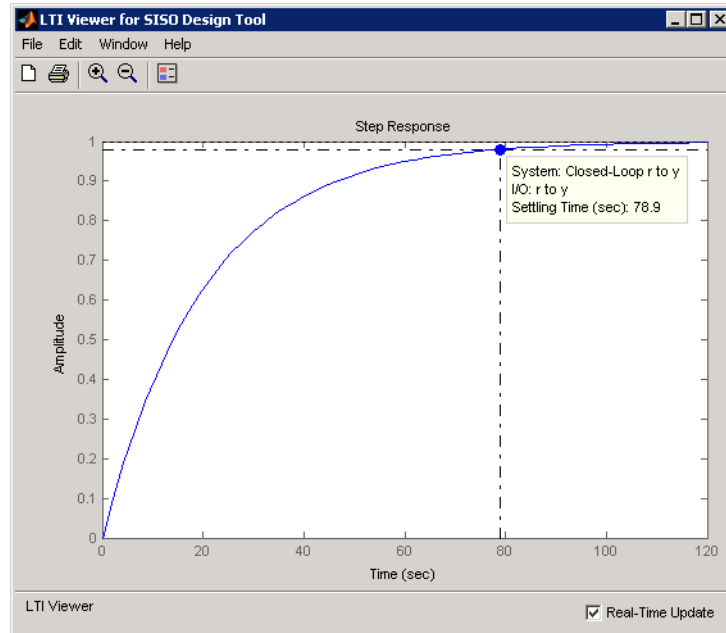
This finishes the design of the inner loop.

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.

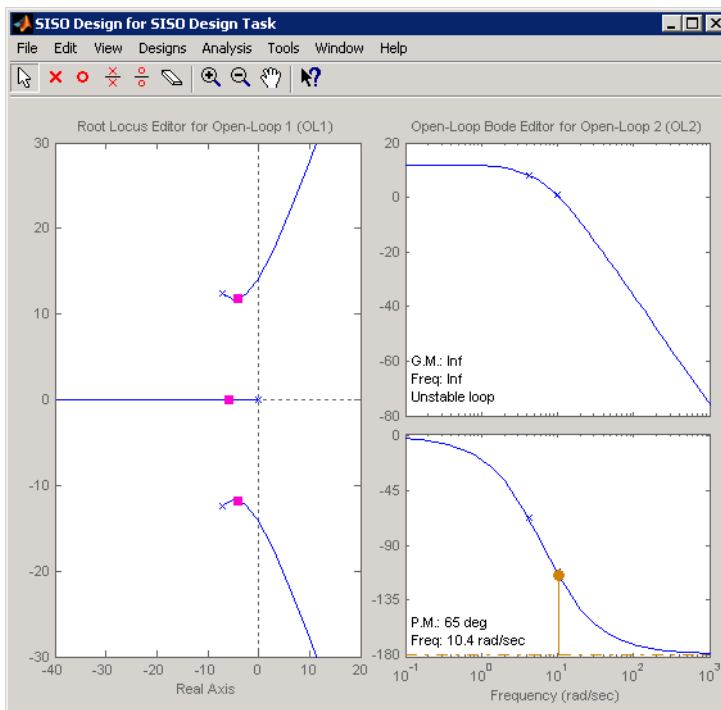
- 2 Select **Step** from the **Plot 1** pull-down menu. This opens the LTI Viewer for SISO Design.
- 3 Right-click in the step response plot and select **Characteristics>Settling Time**. Your LTI Viewer should look like this.



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.

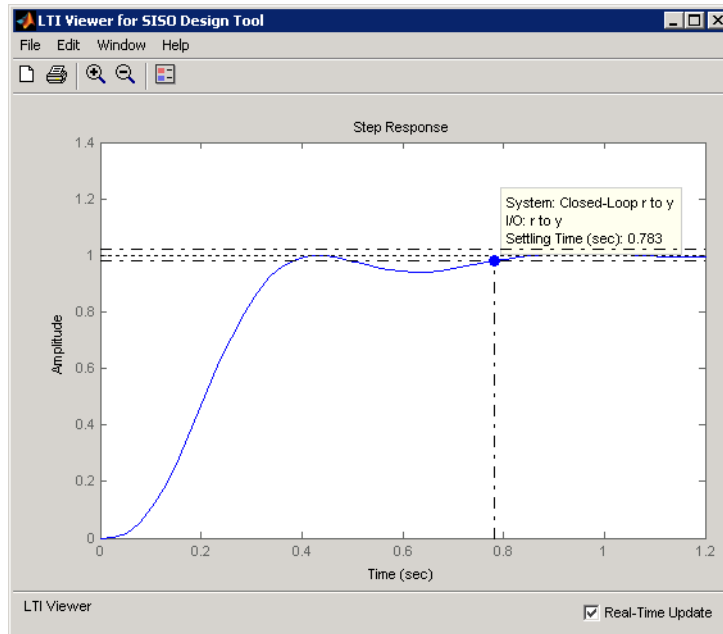
The 90.2 gain seems to yield a good compromise between rise and settling time.



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.

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, the Control System Toolbox provides a set of commands that you can use 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
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,

$$K = \text{place}(A, B, p);$$

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

$$L = \text{place}(A', C', q)$$

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

$$\begin{aligned}\dot{\xi} &= [A - LC - (B - LD)K]\xi + Ly \\ u &= -K\xi\end{aligned}$$

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

$$\text{sys_pp} = \text{ss}(A, B, C, D)$$

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

The Control System Toolbox contains 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 should only be used for systems with a small number of states. The function `place` is a more general and numerically robust alternative to `acker`.

Caution

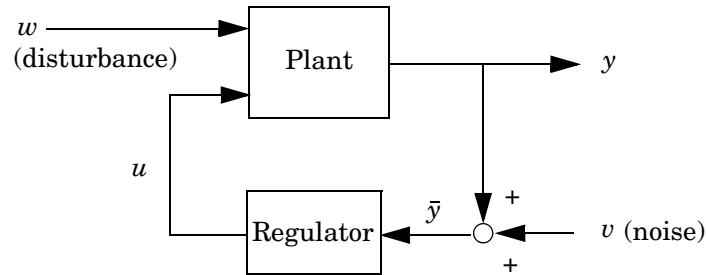
Pole placement can be badly conditioned if you choose unrealistic pole locations. In particular, you should avoid

- 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. It lets you trade off regulation performance and control effort, and to take into account process disturbances and measurement noise. Like pole placement, LQG design requires a state-space model of the plant (use `ss` to convert other model formats to state space). This section focuses on the continuous-time case (see the reference pages for `d1qr` and `kalman` for details on discrete-time LQG design).

LQG design addresses the following regulation problem.



The goal is to regulate the output y around zero. The plant is subject to disturbances w and is driven by controls u . The regulator relies on the noisy measurements $\bar{y} = y + v$ to generate these controls. The plant state and measurement equations are of the form

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

$$\bar{y} = Cx + Du + Hw + v$$

and both w and v are modeled as white noise.

The LQG regulator consists of an optimal state-feedback gain and a Kalman state estimator. You can design these two components independently as shown next.

Optimal State-Feedback Gain

In LQG control, the regulation performance is measured by a quadratic performance criterion of the form

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

The weighting matrices Q , N , and R are user specified and define the tradeoff between regulation performance (how fast $x(t)$ goes to zero) and control effort.

The first design step seeks a state-feedback law $u = -Kx$ that minimizes the cost function $J(u)$. The minimizing gain matrix K is obtained by solving an algebraic Riccati equation. This gain is called the *LQ-optimal* gain.

Syntax. Given the (A,B,C,D) matrices of the system, and the weighting matrices Q, R, and N, use the `lqr` function to construct the LQ-optimal gain, K.

$$K = \text{lqr}(A, B, Q, R, N)$$

If N is omitted, by default its value is 0.

Kalman State Estimator

As in the case of pole placement, the LQ-optimal state feedback $u = -Kx$ is not implementable without full state measurement. It is possible, however, to derive a state estimate \hat{x} such that $u = -K\hat{x}$ remains optimal for the output-feedback problem. This state estimate is generated by the Kalman filter

$$\frac{d}{dt}\hat{x} = A\hat{x} + Bu + L(\bar{y} - C\hat{x} - Du)$$

with inputs u (controls) and \bar{y} (measurements). The noise covariance data

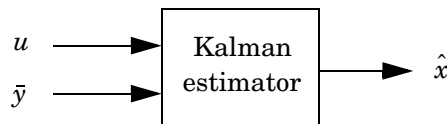
$$E(ww^T) = Q_n, \quad E(vv^T) = R_n, \quad E(uv^T) = N_n$$

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

$$\lim_{t \rightarrow \infty} E((x - \hat{x})(x - \hat{x})^T)$$

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



Syntax. Use the `kalman` function to construct a Kalman filter.

$$[kest, L, P] = \text{kalman}(\text{sys_kf}, Q_n, R_n, N_n);$$

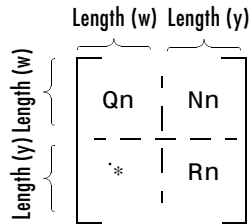
returns a state-space model `kest` of the Kalman estimator given the plant model `sys_kf` and the noise covariance data, Q_n , R_n , and N_n . The plant model equations are the following:

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

$$\bar{y} = Cx + Du + Hw + v$$

where w and v are modeled as white noise. L is the Kalman gain and P the covariance matrix.

The following figure shows the required dimensions for Q_n , R_n , and N_n . If N_n is 0, you can omit it.

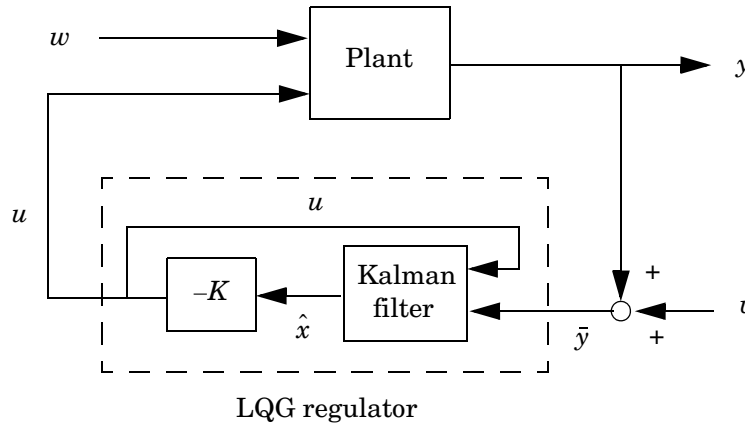


Required Dimensions for Q_n , R_n , and N_n

For a complete example of a Kalman filter implementation, see “Kalman Filtering.”

LQG Regulator

To form the LQG regulator, simply connect the Kalman filter and LQ-optimal gain K as shown in the following figure.



This regulator has state-space equations

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

$$u = -K\hat{x}$$

Syntax. Assuming you have constructed the Kalman filter, `kest`, and the compensator, `K`, use the `lqgreg` function to create the LQG regulator.

```
regulator = lqgreg(kest, K);
```

See “LQG Regulation: Rolling Mill Example” for a more complete discussion of construction LQG regulators.

LQG Design Tools

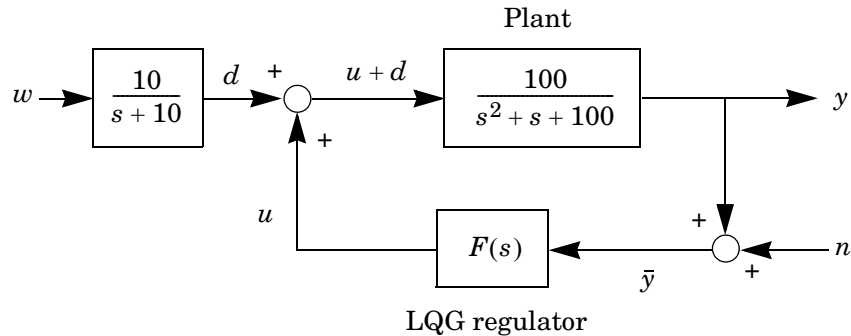
The Control System Toolbox contains functions to perform the LQG design steps outlined in “LQG Regulator”. These functions cover both continuous and

discrete problems as well as the design of discrete LQG regulators for continuous plants. The following table summarizes the LQG design functions.

Function	Description
care	Solve continuous-time algebraic Riccati equations
dare	Solve discrete-time algebraic Riccati equations
dlqr	LQ-optimal gain for discrete systems
kalman	Kalman estimator
kalmd	Discrete Kalman estimator for continuous plant
lqgreg	Form LQG regulator given LQ gain and Kalman filter
lqr	LQ-optimal gain for continuous systems
lqrd	Discrete LQ gain for continuous plant
lqry	LQ-optimal gain with output weighting

Example: LQG Design

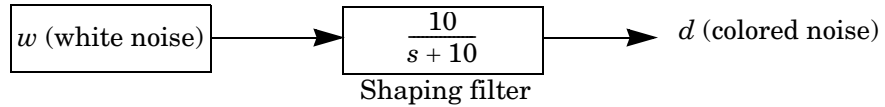
As an example of LQG design, consider the following regulation problem.



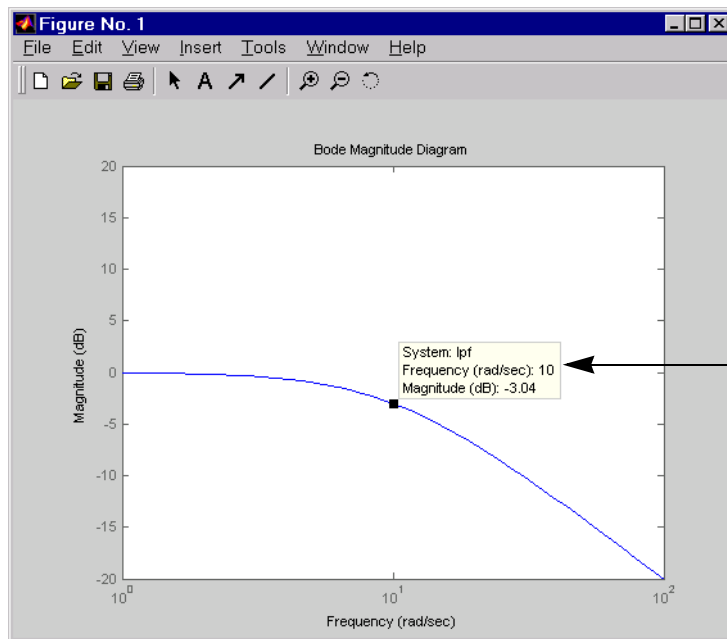
Simple Regulation Loop

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, as this figure shows.



This figure shows the Bode magnitude of the shaping filter.



This data marker verifies that 10 rad/s is the -3 dB point.

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. Note that an open-loop state-space model is

$$\begin{aligned} \dot{x} &= Ax + Bu + Bd && \text{(state equations)} \\ \bar{y} &= Cx + n && \text{(measurements)} \end{aligned}$$

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)

```

The last command returns a state-space model F of the LQG regulator $F(s)$. Note that `lqry`, `kalman`, and `lqgreg` perform discrete-time LQG design when applied 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.

```

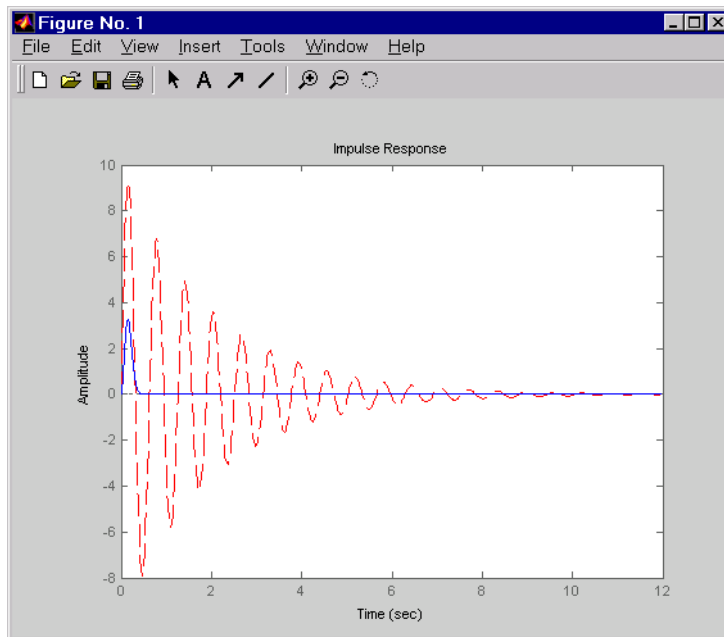
% Close loop
clsys = feedback(sys,F,+1) % Note positive feedback.

% 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-')This figure compares the open-
and closed-loop impulse responses for this example.
```



Comparison of Open- and Closed-Loop Impulse Response for the LQG Example

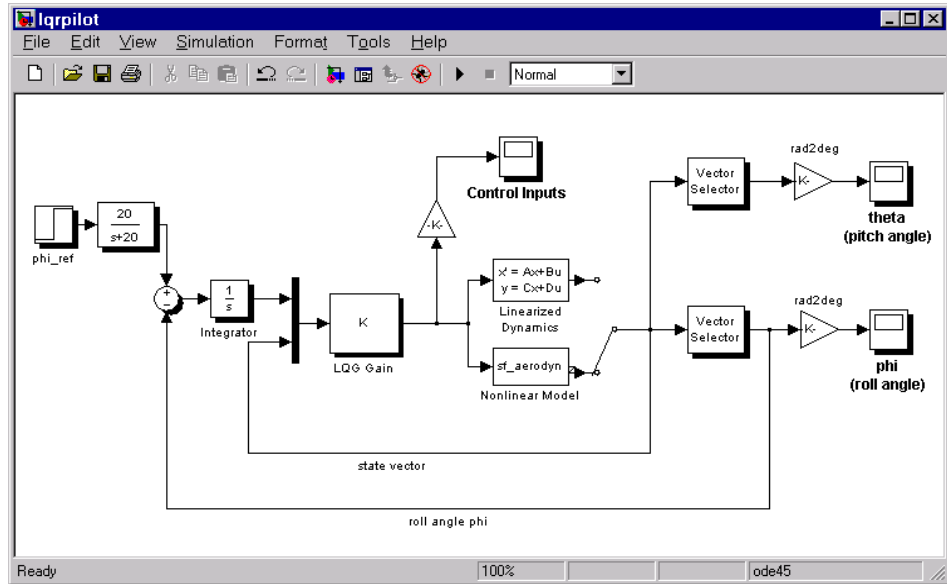
Example: LQG Design for Set Point Tracking

The standard LQG problem is to regulate the plant output around zero. “Example: LQG Design” on page 4-100 describes the classical LQG regulation problem.

You can also apply the LQG design technique to tracking problems, where the goal is to track a reference input (or set point) to the system. To recast the regulator as a tracking problem, you must compare the output y to the

reference signal. The goal is then to drive the error between the output and the reference to zero. A common practice is to add an integrator to the error signal, $e = y - r$, to drive it to zero.

This Simulink block diagram shows a tracking problem in aircraft autopilot design. To open this diagram, type `lqrpilot` at the MATLAB prompt.



Tracking Loop

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

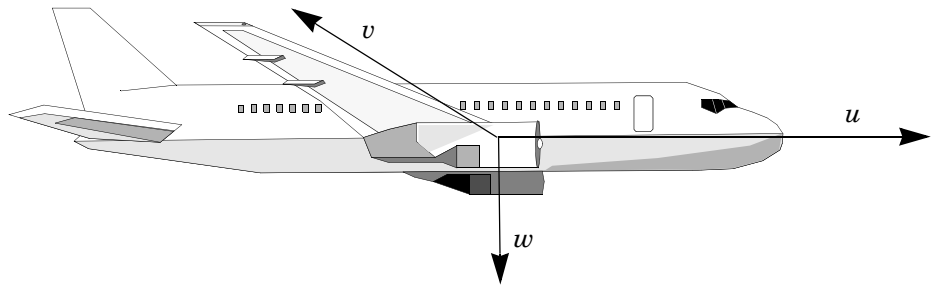
Beginning with the standard state-space equation

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

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 θ and ϕ 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.

$$\dot{x} = Ax + Bu + \begin{bmatrix} -g \sin \theta \\ g \cos \theta \sin \phi \\ g \cos \theta \cos \phi \\ 0 \\ 0 \\ 0 \\ 0 \\ q \cos \phi - r \sin \phi \\ (q \sin \phi + r \cos \phi) \cdot \tan \theta \end{bmatrix}$$

Nonlinear Component of the State-Space Equation

To see the numerical values for A and B , type

```
load lqrpilot
```

A, B

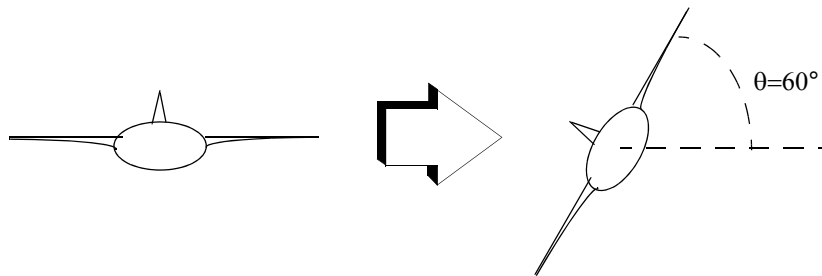
at the MATLAB prompt.

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 $(\phi, \theta) = (0, 15^\circ)$ 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.

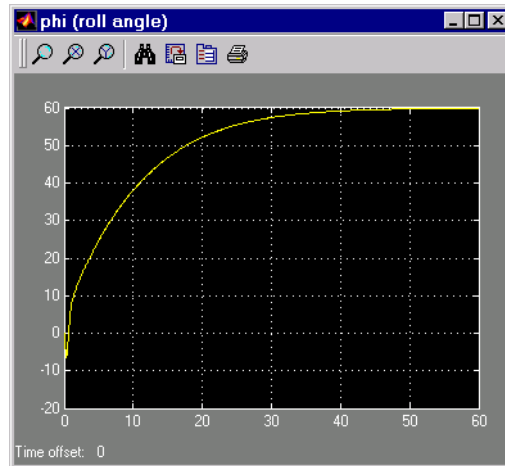
Results

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.

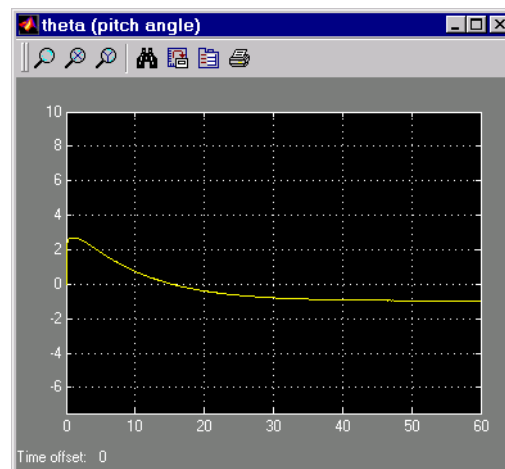
This figure shows the response of ϕ to the 60° step command.



Tracking the Roll Step Command

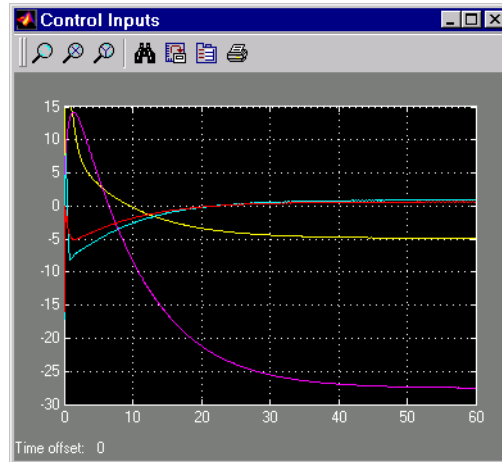
As you can see, the system tracks the commanded 60° roll in about 60 seconds.

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.

Learning More

Locating Demos (p. 5-2)

Where to find Control System Toolbox demos

Online Help (p. 5-2)

How to use the Help Navigator to access online help

Setting Plot Preferences and Properties
(p. 5-2)

How to customize plots generated by the Control System
Toolbox

The MathWorks Online (p. 5-3)

Getting the latest information on the Control System
Toolbox

Locating Demos

To see more Control System Toolbox examples, type

demo

at the MATLAB prompt. This opens the **Demo** pane in the Help browser. Select **Control System Toolbox** under **Toolboxes** to list the available demos.

Alternatively, if you have the Help browser open, you can access the **Demo** pane directly and follow the same procedure.

Online Help

For a more detailed explanation of any of the topics covered in this book, see the documentation listed under Control System Toolbox in the Help Navigator.

The Help Navigator supports string searches. You can specify strings and the online manuals that you want to search. To begin a search, click the **Search** tab. There is also an index available; click the **Index** tab to view it.

Setting Plot Preferences and Properties

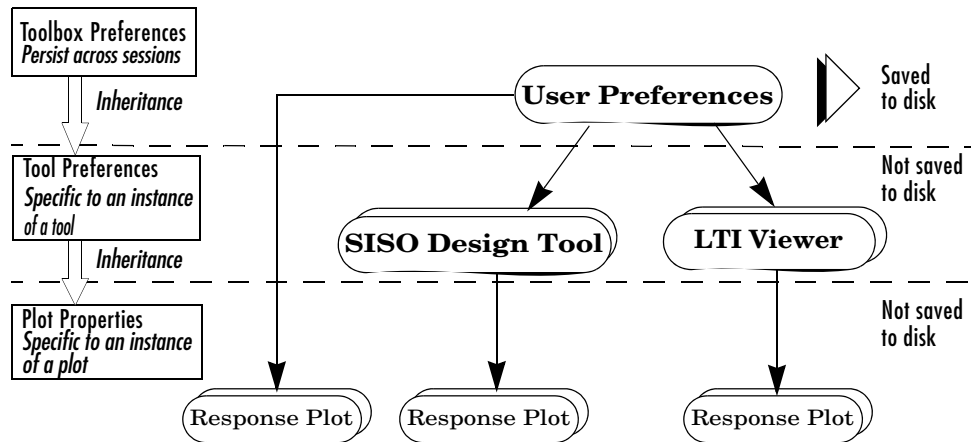
The Control System Toolbox provides three graphical user interfaces (GUIs) that give you control over the visualization of time and frequency plots generated by the toolbox:

- Toolbox Preferences
- Tool Preferences
- Plot Properties

Preferences refer to global options that you can save from session to session or to any LTI Viewer or SISO Design Tool that you open during a single session. *Properties* are options that apply only to the current window. This section gives an overview of the three GUIs; see the online help system for complete descriptions.

Although you can set plot properties in any response plot, you can use the Toolbox Preferences Editor to set properties for any response plot the Control

System Toolbox generates. This figure shows the inheritance hierarchy from toolbox preference to plot properties.



Preference and Property Inheritance Hierarchy

You can activate preference and plot editors by doing the following:

- Toolbox preferences — Select **Toolbox Preferences** under **File** in either the LTI Viewer or the SISO Design Tool.
- Tool preferences — Select **SISO Tool Preferences** under **Edit** for the SISO Design Tool and **Viewer Preferences** under **Edit** in the LTI Viewer.
- Plot properties — Double-click any response plot created by the Control System Toolbox or select **Properties** from the right-click menus.

For a complete discussion of how to use property and preference editors, see “Customization”.

The MathWorks Online

For the very latest information about the Control System Toolbox and other MathWorks products, point your Web browser to

www.mathworks.com

and use your Internet news reader to access the newsgroup

`comp.soft-sys.matlab`

Many books that use MATLAB and the Control System Toolbox to explain control engineering concepts are available from different publishers. A booklet titled MATLAB Based Books is available from The MathWorks and an up-to-date list is available on the Web site.

A

automated tuning design 4-75

B

building models

 MIMO 2-12

 SISO 2-5

C

cell array 2-15

changing

 units on a plot 4-14

comparing

 models in plots 3-40

compensator design

 multi-loop 4-81

compensators

 editing in SISO Design Tool 4-53

 exporting 4-59

 feedback (state estimator design) 4-94

concatenation

 model 2-14

covariance

 noise 4-97

D

data markers 3-38

denominator

 specification 2-15

design

 Kalman estimator 4-97

LQG Tracker, example of 4-103

LQG, description 4-95

LQG, example 4-100

LQG, syntax 4-99

pole placement 4-92

regulators 4-99

root locus 4-92

state estimator 4-93

state estimator, Kalman 4-97

E

estimator

 gain 4-93

 Kalman 4-97

exporting compensators and models 4-59

F

final time. *See also* time response

frequency response 3-36

 customizing plots 3-42

 plotting 3-40

functions

 time and frequency response 3-33

G

gain

 estimator, Kalman 4-97

 selection 4-41

 state-feedback 4-96

Graphical Tuning window 4-2

 changing units on a plot 4-14

display 4-14
right-click menus 4-16
using 4-13

H

Help Navigator 5-2

I

input vector 2-2
installation 1-3

K

Kalman estimator
 continuous 4-97
 gain 4-97
 steady-state 4-97

Kalman filter

See Kalman estimator

L

linear models
 comparing multiple models in plots 3-40
 exporting 4-59
 frequency response. 3-36
 model order reduction 2-25
 time response 3-33

LQG (linear quadratic Gaussian) method

 continuous LQ regulator 4-96
 design

See design, LQG

 gain, optimal state-feedback 4-96

 Kalman state estimator 4-97

 LQ-optimal gain 4-96

 regulator, designing 4-102

 regulator, forming 4-99

 weighting matrices 4-96

LTI Viewer

 adding plot types 3-12

 changing models 3-16

 changing plot types 3-10

 command line initializing 3-13

 comparing multiple models 3-13

 importing models 3-4

 introduction to 3-3

 models, importing 3-4

 opening 3-4

 plot options

 rise time 3-7

 settling time 3-14

 right-click menus 3-5

 SISO Design Task 4-2

M

markers, data 3-38

MathWorks Web site 5-3

MIMO 3-35

model order reduction 2-25

models

 building 2-12

multi-loop design 4-81

N

noise

 covariance 4-97

 white 3-33

notch filter

 parameters 4-34

numerator

specification 2-15

O

online help 5-2
operations on LTI models
 concatenation 2-14
output vector 2-2
overshoot 3-33

P

plotting
 changing units 4-14
 customizing plots 3-42
 multiple systems 3-40
 right-click menus 3-36
 time responses 3-33
pole placement 4-92
 conditioning 4-95

R

reduced-order models 2-25
regulation 4-96, 4-102
 performance 4-96
Riccati equation 4-96
right-click menus
 LTI Viewer 3-5
 plot responses 3-36
rise time 3-33
root locus
 compensator gain 4-47
 designing 4-92
root locus design 4-41
 functions for 4-92

S

settling time 3-33
SISO Design Task in the Control and Estimation
 Tools Manager 4-2
SISO Design Task pane 4-2
 Analysis Plots page 4-10
 Automated Tuning page 4-77
 Design History page 4-62
 using 4-4
SISO Design Tool 4-2
 adding a lead network 4-24
 adding a lead network to a Nichols plot 4-71
 adding a notch filter 4-31
 adding an integrator 4-20
 adding an integrator to the Nichols plot 4-70
 adding poles and zeros 4-49
 adjusting compensator gain in a Nichols plot
 4-66
 adjusting the compensator gain in Bode design
 4-17
 applying automated PID tuning 4-77
 automated tuning design 4-75
 Bode Diagram Design 4-17
 changing the compensator gain, Bode
 magnitude 4-17
 changing the compensator gain, root locus
 4-47
 compensators, editing 4-53
 editing preferences 4-14
 exporting compensator and models 4-59
 feedback structures 4-8
 importing models into 4-5
 loop responses 4-10
 Nichols plot design 4-65
 opening 4-2
 opening a Nichols plot 4-66
 root locus design 4-41

storing and retrieving designs 4-61
Using the Graphical Tuning window 4-13
SISO Tool Preferences Editor 4-14
state
 estimator 4-93
 vector 2-2
state estimator design
 regulators/compensators 4-94
state-space models 2-2
steady state error 3-33

T

time response
 customizing plots 3-42
 final time 3-35
 functions 3-33
 MIMO 3-35
 plotting 3-40
 time range 3-35
 to white noise 3-33
 vector of time samples 3-36
transfer functions
 MIMO 2-14
transfer functions (TF) 2-2

U

units
 changing on a plot 4-14

W

white noise 3-33
 See also noise

Z

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

