

# Control System Toolbox

For Use with MATLAB®

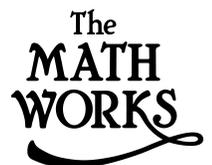
Computation

Visualization

Programming

Getting Started

*Version 5*



## How to Contact The MathWorks:



www.mathworks.com  
comp.soft-sys.matlab

Web  
Newsgroup



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

Technical support  
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

Mail

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

### *Getting Started with the Control System Toolbox*

© COPYRIGHT 2000 - 2001 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 or for the federal government of the United States. By accepting delivery of the Program, the government hereby agrees that this software qualifies as "commercial" computer software within the meaning of FAR Part 12.212, DFARS Part 227.7202-1, DFARS Part 227.7202-3, DFARS Part 252.227-7013, and DFARS Part 252.227-7014. The terms and conditions of The MathWorks, Inc. Software License Agreement shall pertain to the government's use and disclosure of the Program and Documentation, and shall supersede any conflicting contractual terms or conditions. If this license fails to meet the government's minimum needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to MathWorks.

MATLAB, Simulink, Stateflow, Handle Graphics, and Real-Time Workshop are registered trademarks, and Target Language Compiler is a trademark of The MathWorks, Inc.

Other product or brand names are trademarks or registered trademarks of their respective holders.

Printing History: November 2000 First printing New for Version 5 (Release 12)  
June 2001 Revised for Version 5.1 (Online only)

## Introduction

1

<b>What Is the Control System Toolbox?</b> .....	<b>1-2</b>
<b>Installation</b> .....	<b>1-3</b>
<b>Using the Documentation</b> .....	<b>1-4</b>
<b>Related Products</b> .....	<b>1-6</b>
<b>Typographical Conventions</b> .....	<b>1-8</b>

## Building Models

2

<b>Linear Models</b> .....	<b>2-3</b>
Linear Model Representations .....	<b>2-3</b>
SISO Example: the DC Motor .....	<b>2-4</b>
Constructing SISO Models .....	<b>2-6</b>
Discrete Time Systems .....	<b>2-10</b>
Adding Delays to Linear Models .....	<b>2-11</b>
LTI Objects .....	<b>2-11</b>
<b>MIMO Models</b> .....	<b>2-13</b>
MIMO Example: Jet Transport Aircraft .....	<b>2-13</b>
Constructing MIMO Transfer Functions .....	<b>2-15</b>
Accessing I/O Pairs in MIMO Systems .....	<b>2-17</b>
<b>Arrays of Linear Models</b> .....	<b>2-18</b>
<b>Model Characteristics</b> .....	<b>2-21</b>

<b>Interconnecting Linear Models</b> .....	<b>2-22</b>
Feedback Interconnection .....	<b>2-23</b>
<b>Continuous/Discrete Conversions</b> .....	<b>2-24</b>
Discrete DC Motor Model .....	<b>2-24</b>
<b>Model Order Reduction</b> .....	<b>2-26</b>
Example: Gasifier Model .....	<b>2-26</b>

## Analyzing Models

### 3

<b>LTI Viewer</b> .....	<b>3-3</b>
Example: Time and Frequency Responses of the DC Motor .....	<b>3-3</b>
Right-Click Menus .....	<b>3-5</b>
Displaying Response Characteristics on a Plot .....	<b>3-7</b>
Changing Plot Type .....	<b>3-10</b>
Showing Multiple Response Types .....	<b>3-12</b>
Comparing Multiple Models .....	<b>3-13</b>
<b>Functions for Time and Frequency Response</b> .....	<b>3-17</b>
Time and Frequency Responses .....	<b>3-17</b>
Plotting MIMO Model Responses .....	<b>3-19</b>
Data Markers .....	<b>3-22</b>
Plotting and Comparing Multiple Systems .....	<b>3-23</b>
Creating Custom Plots .....	<b>3-26</b>
<b>Simulink LTI Viewer</b> .....	<b>3-29</b>
Using the Simulink LTI Viewer .....	<b>3-29</b>
Comparing Linear and Nonlinear Models .....	<b>3-30</b>
Bode Plots of Linearized Models .....	<b>3-37</b>
Specifying Operating Conditions .....	<b>3-42</b>

<b>The SISO Design Tool</b> .....	<b>4-3</b>
Opening the SISO Design Tool .....	4-3
Importing Models into the SISO Design Tool .....	4-5
Feedback Structure .....	4-6
Loop Responses .....	4-7
<b>Bode Diagram Design</b> .....	<b>4-8</b>
Example: DC Motor .....	4-8
Adjusting the Compensator Gain .....	4-8
Right-Click Menus .....	4-9
Adjusting the Bandwidth .....	4-9
Adding an Integrator .....	4-12
Adding a Lead Network .....	4-14
Moving Compensator Poles and Zeros .....	4-17
Changing Units on a Plot .....	4-19
Adding a Notch Filter .....	4-20
Adding a Prefilter .....	4-24
<b>Root Locus Design</b> .....	<b>4-27</b>
Example: Electrohydraulic Servomechanism .....	4-28
Changing the Compensator Gain .....	4-35
Adding Poles and Zeros to the Compensator .....	4-36
Editing Compensator Pole and Zero Locations .....	4-40
Viewing Damping Ratios .....	4-42
Exporting the Compensator and Models .....	4-45
Storing and Retrieving Intermediate Designs .....	4-46
<b>Nichols Plot Design</b> .....	<b>4-48</b>
DC Motor Example .....	4-48
Opening a Nichols Plot .....	4-49
Adjusting the Compensator Gain .....	4-50
Adding an Integrator .....	4-52
Adding a Lead Network .....	4-53
<b>Functions For Compensator Design</b> .....	<b>4-57</b>
Root Locus Design .....	4-57
Pole Placement .....	4-57

Linear-Quadratic-Gaussian (LQG) Design . . . . .	<b>4-61</b>
Example: LQG Design . . . . .	<b>4-66</b>
Example: LQG Design for Set Point Tracking . . . . .	<b>4-70</b>

## **Learning More**

# **5**

Demos . . . . .	<b>5-2</b>
Online Help . . . . .	<b>5-2</b>
Setting Plot Preferences and Properties . . . . .	<b>5-3</b>
The MathWorks Online . . . . .	<b>5-4</b>

# Introduction

---

<b>What Is the Control System Toolbox? . . . . .</b>	<b>1-2</b>
<b>Installation . . . . .</b>	<b>1-3</b>
<b>Using the Documentation . . . . .</b>	<b>1-4</b>
<b>Related Products . . . . .</b>	<b>1-6</b>
<b>Typographical Conventions . . . . .</b>	<b>1-8</b>

## What Is the Control System Toolbox?

MATLAB® has a rich collection of functions immediately useful to the control engineer or system theorist. Complex arithmetic, eigenvalues, root-finding, matrix inversion, and FFTs are just a few examples of MATLAB's important numerical tools. More generally, MATLAB's 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 (GUI's) 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 Guide* for your platform. We recommend that you store the files from this toolbox in a directory named `control` off 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.

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

```
demo
```

at the MATLAB prompt and select **Control System** under the **Toolboxes** heading. In addition, Design Case Studies in the online documentation contains detailed examples of various design problems.

## Using the Documentation

**If you are a new user**, this manual, *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 **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 online and to PDF versions of the same documentation.

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

- “Release Notes” — For 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” — 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 (GUI’s) that you can use to analyze systems and design SISO compensators

**All toolbox users** should use the online Control System Toolbox Function Reference 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.

## Related Products

The MathWorks provides several associated products that are especially relevant to the kinds of tasks you can perform with the Control System Toolbox. For more information about any of these products, see either:

- The online documentation for that product, if it is installed or if you are reading the documentation from the CD
- The MathWorks Web site, at <http://www.mathworks.com>; see the “products” section

The table below lists MathWorks products that complement the functionality of the Control System Toolbox.

Product	Description
Fuzzy Logic Toolbox	Tool to help master fuzzy logic techniques and their application to practical control problems
LMI Control Toolbox	Convex optimization algorithms for solving linear matrix inequalities (LMI), with application to robust control, multi-objective control, and gain scheduling
Model Predictive Control Toolbox	MATLAB functions for the analysis and design of model predictive control (MPC) systems, including multivariable applications with time delays and constrained variables
$\mu$ -Analysis and Synthesis Toolbox	Computational algorithms for the structured singular value, $\mu$ , applicable to robustness and performance analysis for systems with modeling and parameter uncertainties
Nonlinear Control Design Blockset	Simulink block libraries that provide a time-domain-based optimization approach to system design; automatically tunes parameters based on user-defined time-domain performance constraints

<b>Product</b>	<b>Description</b>
Robust Control Toolbox	Tools for modeling, analysis, and design of “robust” multivariable feedback control systems using $H_\infty$ techniques
Simulink	Interactive, graphical environment for modeling, simulating, and prototyping dynamic systems
System Identification Toolbox	Tool for building accurate, simplified models of complex systems from noisy time-series data

## Typographical Conventions

This manual uses some or all of these conventions.

Item	Convention Used	Example
Example code	Monospace font	To assign the value 5 to A, enter <code>A = 5</code>
Function names/syntax	Monospace font	The <code>cos</code> function finds the cosine of each array element. Syntax line example is <code>MLGetVar ML_var_name</code>
Keys	<b>Boldface</b> with an initial capital letter	Press the <b>Return</b> key.
Literal strings (in syntax descriptions in reference chapters)	<b>Monospace bold</b> for literals	<code>f = freqspace(n, 'whole')</code>
Mathematical expressions	<i>Italics</i> for variables Standard text font for functions, operators, and constants	This vector represents the polynomial $p = x^2 + 2x + 3$
MATLAB output	Monospace font	MATLAB responds with <code>A =</code> <code>5</code>
Menu titles, menu items, dialog boxes, and controls	<b>Boldface</b> with an initial capital letter	Choose the <b>File</b> menu.
New terms	<i>Italics</i>	An <i>array</i> is an ordered collection of information.
Omitted input arguments	(...) ellipsis denotes all of the input/output arguments from preceding syntaxes.	<code>[c,ia,ib] = union(...)</code>
String variables (from a finite list)	<i>Monospace italics</i>	<code>sysc = d2c(sysd,'method')</code>

# Building Models

---

<b>Linear Models</b> . . . . .	2-3
<b>MIMO Models</b> . . . . .	2-13
<b>Arrays of Linear Models</b> . . . . .	2-18
<b>Model Characteristics</b> . . . . .	2-21
<b>Interconnecting Linear Models</b> . . . . .	2-22
<b>Continuous/Discrete Conversions</b> . . . . .	2-24
<b>Model Order Reduction</b> . . . . .	2-26

This chapter discusses how to build models of linear, time invariant (LTI) dynamical systems using functions from the Control System Toolbox. It begins by developing a simple single-input, single-output (SISO) model of a DC motor and describes the various model representations possible, including:

- Transfer functions
- State-space
- Zero/pole/gain
- Frequency response data

This chapter then describes how to build multiple-input multiple-output (MIMO) models by presenting a jet transport model. It also discusses topics related to creating general LTI models, including:

- How to access model characteristics
- Conversions between model representations
- Building larger models from smaller ones
- Accessing and manipulating I/O pairs
- LTI objects, which are MATLAB objects that store multiple linear models in a single variable

This chapter also discusses discrete-time systems, including analog to discrete-time conversion, sample time specification, and how to introduce time delays in your linear system. The last section describes functions that perform model order reduction and presents an example of how to perform a model order reduction.

## Linear Models

Typically, control engineers begin by developing a mathematical description of the dynamical system that they want to control. This to-be-controlled system 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 under Creating and Manipulating Models in the online documentation 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.

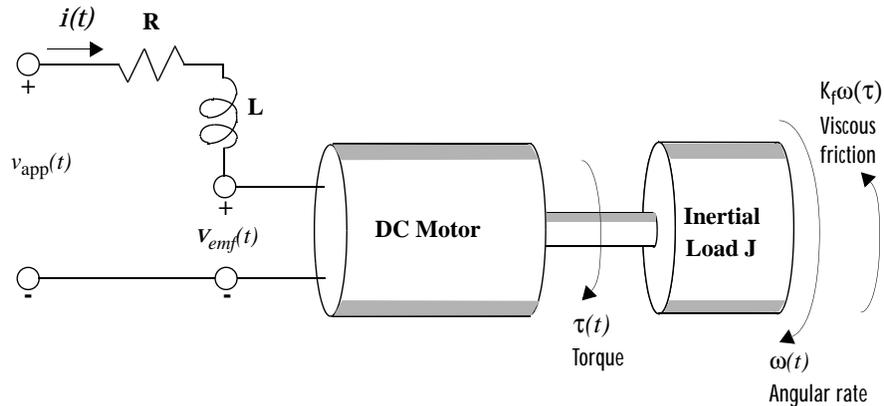


Figure 2-1: A Simple Model of a DC Motor Driving an Inertial Load

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  $\tau$  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  $\omega$  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 dynamical system. The current  $i$  and the angular rate  $\omega$  are the two states of the system. The applied voltage,  $v_{app}$ , is the input to the system, and the angular velocity  $\omega$  is the output.

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

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

Figure 2-2: 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 directly

### Constructing a State-Space Model of the DC Motor

Listed below are nominal values for the various parameters of a DC motor.

$$R = 2.0 \text{ \% Ohms}$$

```

L= 0.5 % Henrys
Km = .015 % Torque constant
Kb = .015 % emf constant
Kf = 0.2 % Nms
J= 0.02 % kg.m^2/s^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* online for more information on numerical issues.

---

### Constructing Transfer Function and Zero/Pole/Gain Models

In the DC motor example, the state-space approach produced 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 Systems 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:

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

For a complete description of adding time delays to models, see Time Delays online under Creating and Manipulating Models.

## 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}
    Ts: 0
    ioDelay: 0
    InputDelay: 0
    OutputDelay: 0
```

```
    InputName: {''}  
    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](#) online under the Control System Toolbox.

## 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” in “Creating and Manipulating Models” online 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 the “Structures and Cell Arrays” in MATLAB’s online 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
```

$$\#2: \frac{s + 2}{s^2 + 4s + 5}$$

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.

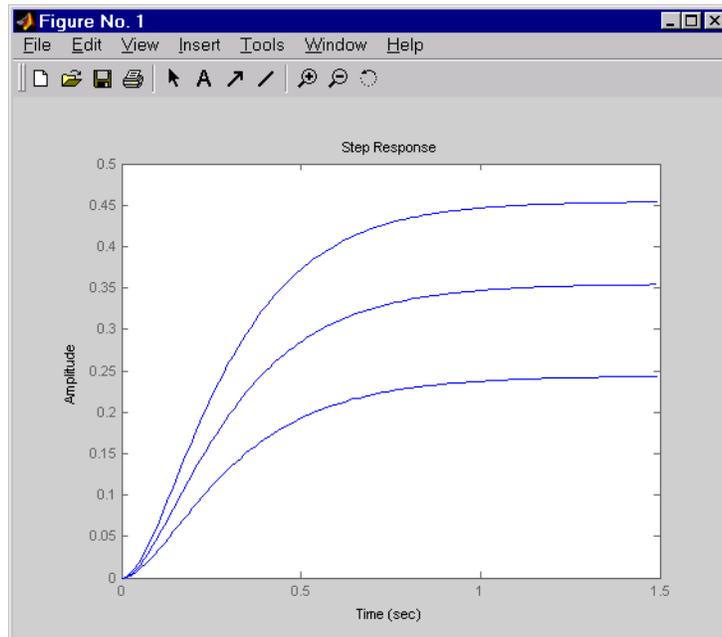


Figure 2-3: Step Responses for an LTI Array Containing Three Models

**LTI arrays are useful for performing batch analysis on an entire set of models. For more information, see [Arrays of LTI Models](#) online under [Creating and Manipulating 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 model. Their LTI-based syntax is summarized in the table below.

**Table 2-1: 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 $L_\infty$ )
<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](#) under [Creating and Manipulating Models](#) online for more information. State-space models have highest precedence while transfer functions have 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](#) online under [Creating and Manipulating Models](#). The Control System Toolbox also supports matrix-like indexing for extracting subsystems; see [Extracting and Modifying Subsystems](#) online in [Operations on LTI Models](#) (also under [Creating and Manipulating Models](#)) for more information.

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

**Table 2-2: 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,

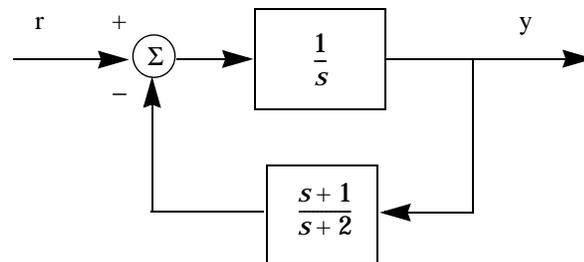
```
sys_f = feedback(tf(1,[1 0]), tf([1 1],[1 2]))
```

computes the closed-loop transfer function from  $r$  to  $y$  for the feedback loop shown in Figure 2-4. The result is

Transfer function:

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

This figure shows the interconnected system in block diagram format.



**Figure 2-4: 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 online 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 2-4 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 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 ltiexamples
```

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

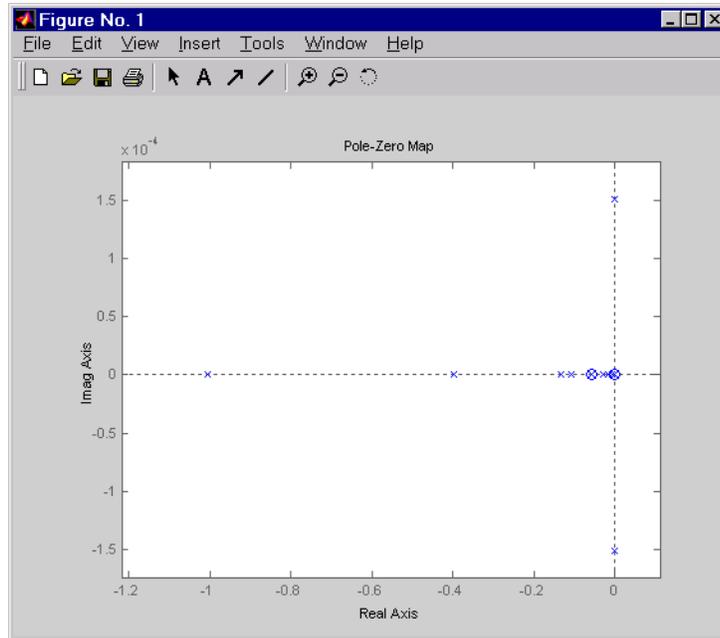
```
size(gasf)
```

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 4-33 for an example.

This figure shows the results.



**Figure 2-5: Pole-Zero Map of the Gasifier Model (Zoomed In)**

Since the model displays near pole-zero cancellations, it's 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)
```

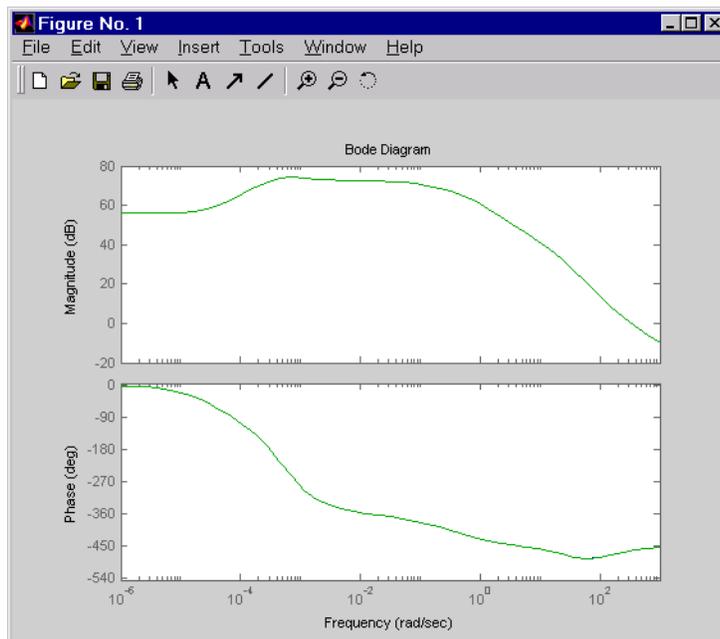
MATLAB responds with

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.



**Figure 2-6: 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

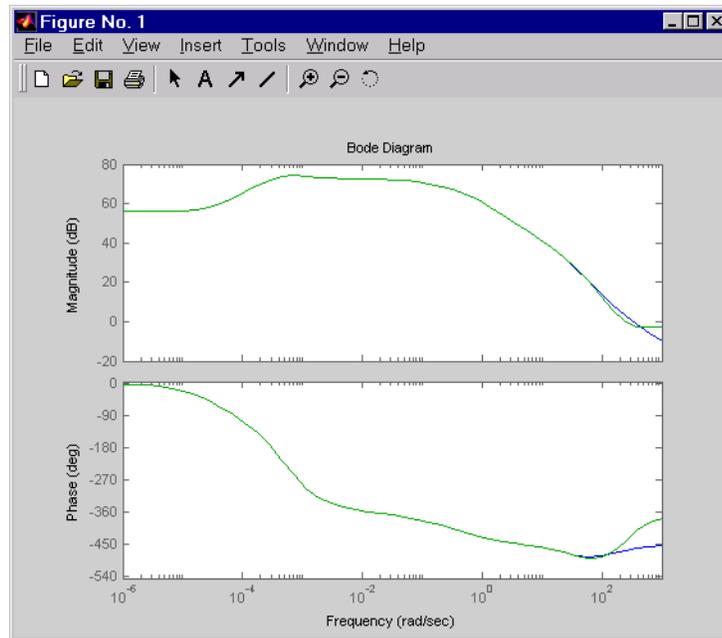
```
State-space model with 1 output, 1 input, and 10 states.
```

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.



**Figure 2-7: Comparison of 25 and 10 State Models**

The figure shows good agreement until the frequency reaches 5 rad/sec. 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.

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. 93-96.



# Analyzing Models

---

<b>LTI Viewer</b> . . . . .	3-3
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
<b>Functions for Time and Frequency Response</b> . . . . .	3-17
Time and Frequency Responses . . . . .	3-17
Plotting MIMO Model Responses . . . . .	3-19
Data Markers . . . . .	3-22
Plotting and Comparing Multiple Systems . . . . .	3-23
Creating Custom Plots . . . . .	3-26
<b>Simulink LTI Viewer</b> . . . . .	3-29
Using the Simulink LTI Viewer . . . . .	3-29
Comparing Linear and Nonlinear Models . . . . .	3-30
Bode Plots of Linearized Models . . . . .	3-37
Specifying Operating Conditions . . . . .	3-42

This chapter shows how you can analyze models using tools provided by the Control System Toolbox. It begins by introducing the LTI Viewer, a graphical user interface (GUI) that simplifies the analysis of linear, time-invariant systems.

The chapter continues with a discussion of commands provided by the Control System Toolbox for plotting the time and frequency responses of linear models. This set of commands provides an open and extensible environment for the analysis of control systems. You can use these commands when you need more flexibility than the LTI Viewer provides.

Finally, this chapter discusses the Simulink LTI Viewer, for use with Simulink. You can use this tool to linearize models or parts of models, analyze linearized models, and compare linear and nonlinear models.

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.

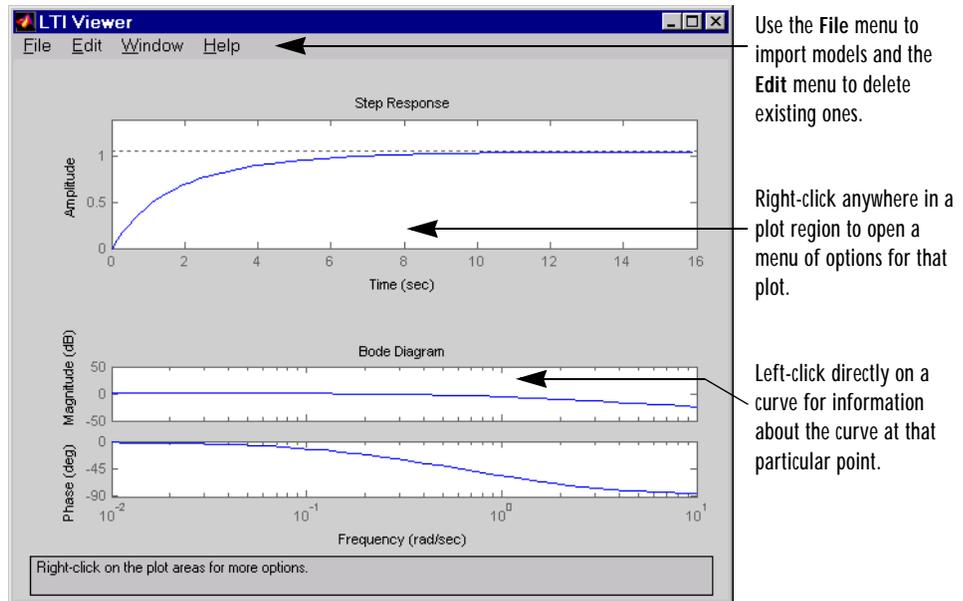
---

**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. However, all models must have the same input and output sizes. If your models have unequal input or output sizes, you must display them in different LTI Viewers.

---

## LTI Viewer

This picture shows an LTI Viewer with two response plots.



**Figure 3-1: The LTI Viewer with Step Response and Bode 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

The section entitled “SISO Example: the DC Motor” on page 2-4 presented 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

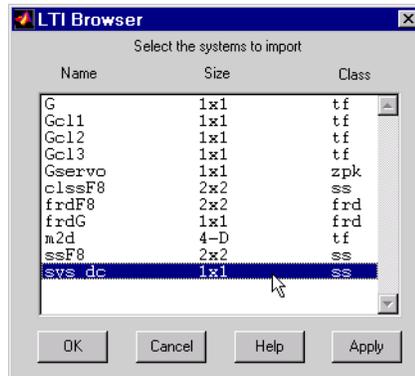
To open the LTI Viewer, type

```
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 **LTI Browser** dialog box, which lists all the models available in your MATLAB workspace.



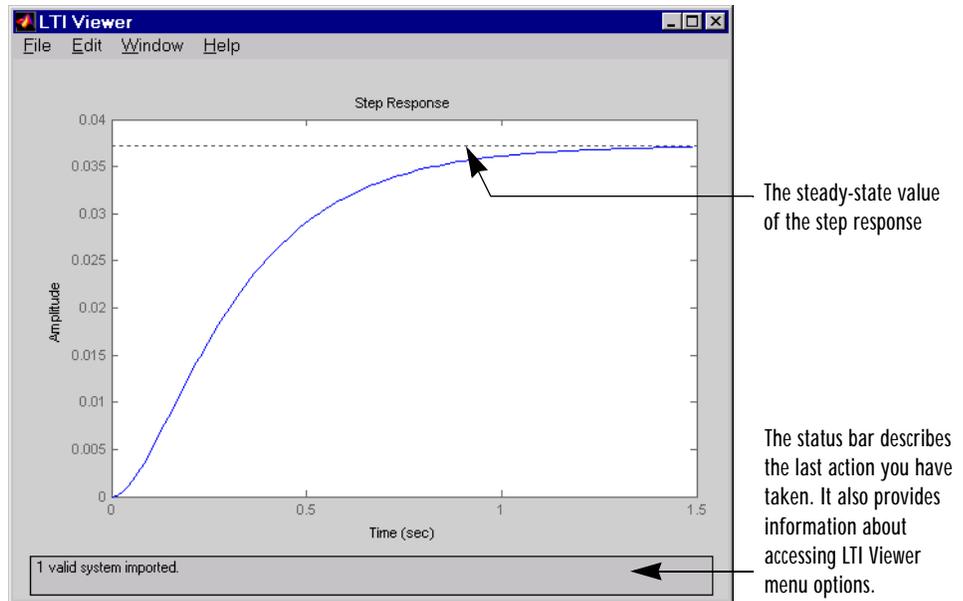
**Figure 3-2: The LTI Browser with the DC Motor Model Selected**

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 deselect any models, hold down the **Ctrl** key while you click on 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 picture 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)
```

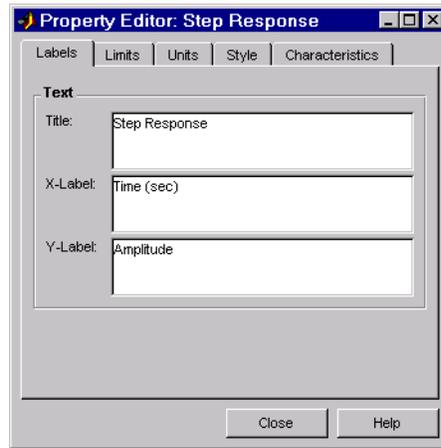
See the `ltiview` reference page for a complete list of options.

## Right-Click Menu

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 sigma plots.
- **Systems** — Select or deselect 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.
- **Zoom** — Zoom in and out of plot regions.
- **Grid** — Add grids to your plots.
- **Properties** — Open the Property Editor.

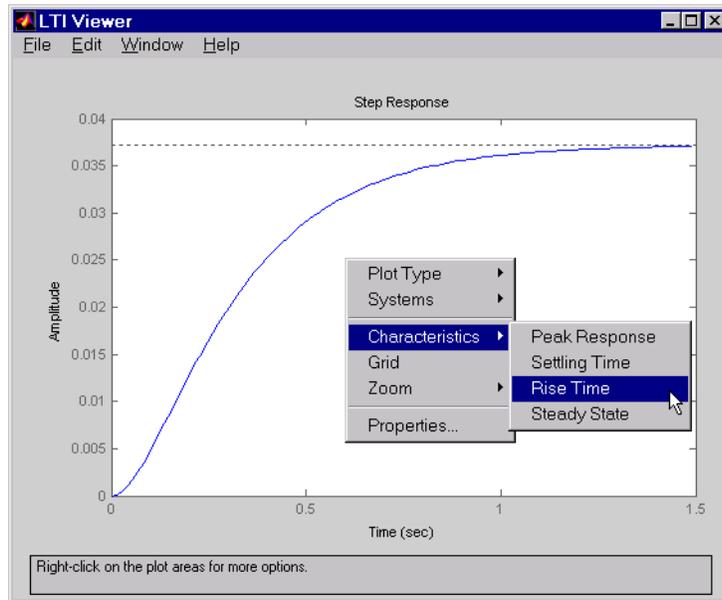


You can use this editor to customize various attributes of your plot. See [Customizing Plot Properties and Preferences](#) online under [Customization](#) for a full description of the Property Editor.

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

## Displaying Response Characteristics on a Plot

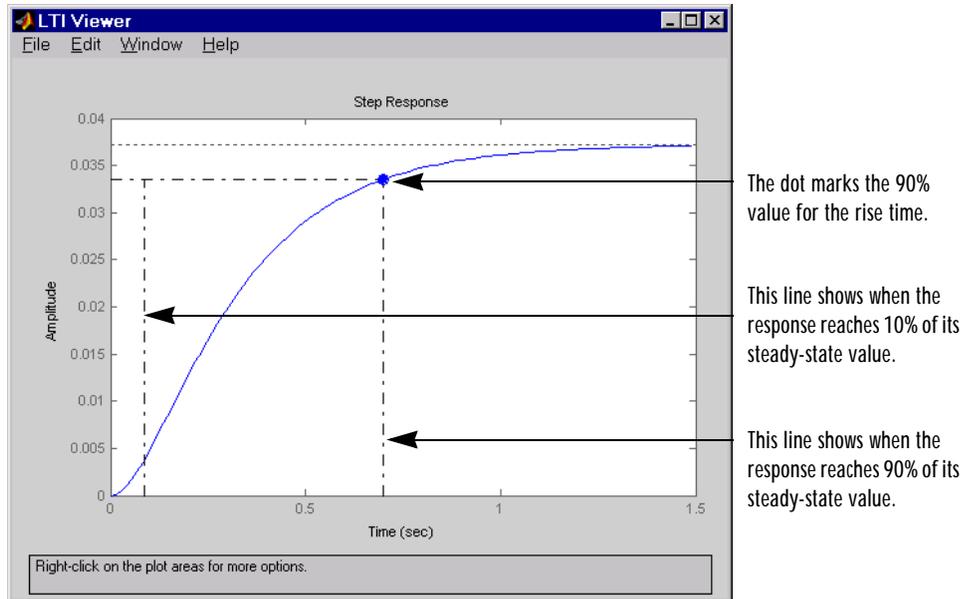
For example, to see the rise time for the DC motor step response, right-click your mouse and select **Rise Time** under **Characteristics**, as this picture illustrates.



**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.

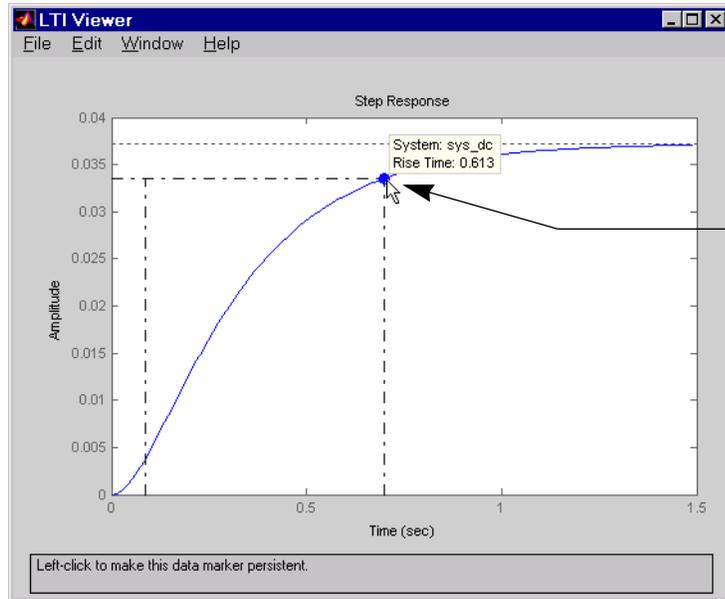
The LTI Viewer calculates and displays the rise time for the step response.



**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 on the blue dot.

For example, this picture shows the rise time value for the DC motor step response.



Place your cursor over the blue dot to display a data marker with the system name and the 90% rise time value.

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

**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 3-22.

## 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.

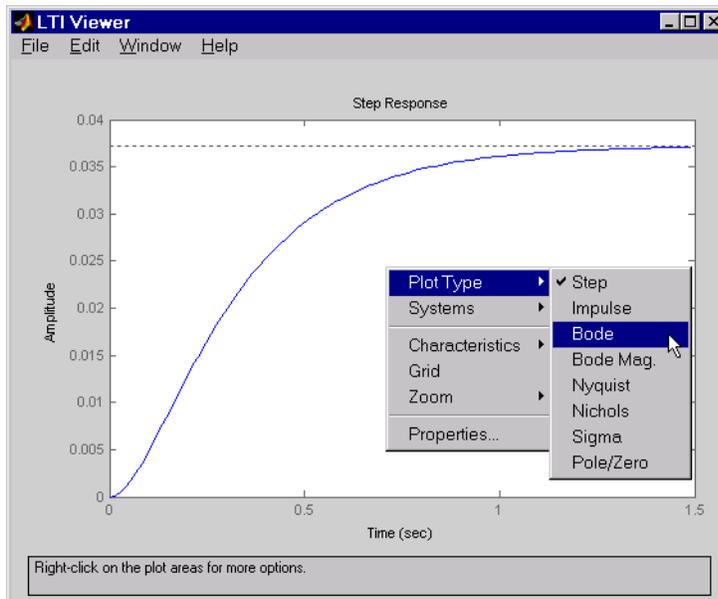


Figure 3-7: Changing the Step Response to a Bode Plot

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

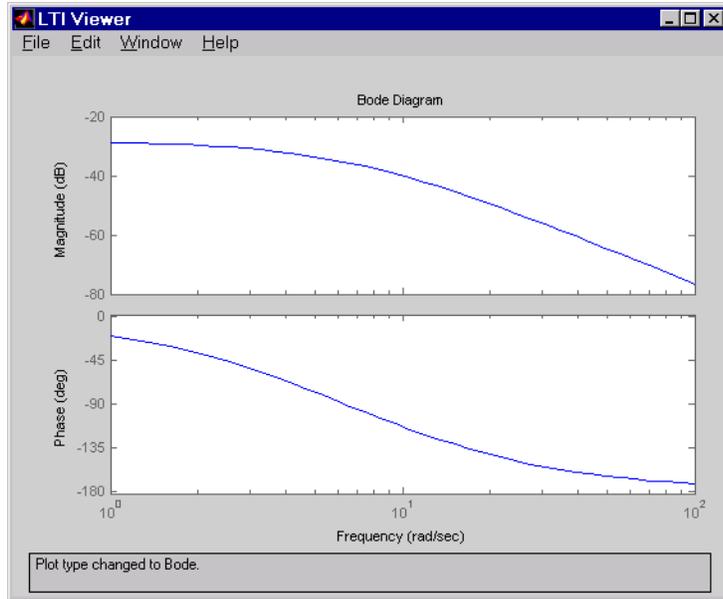
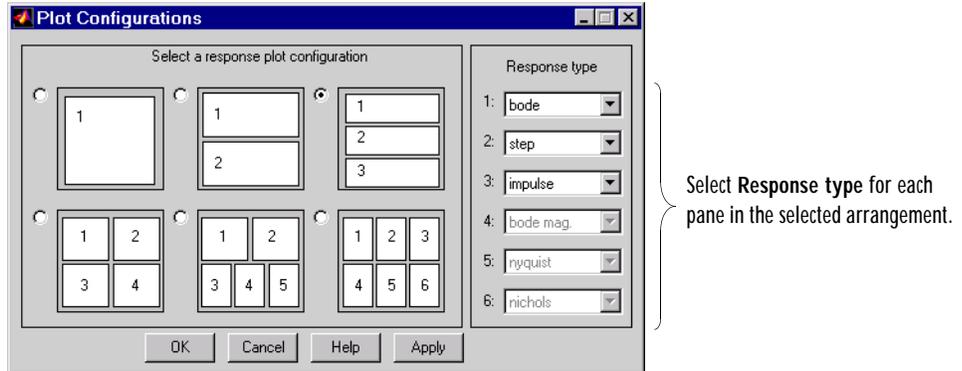


Figure 3-8: Bode Plot for the DC Motor Model

## 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-hand side menus from eight available plot types:

- Step
- Impulse
- Bode (magnitude and phase)
- Bode Magnitude (only)
- Nyquist
- Nichols
- Sigma
- 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

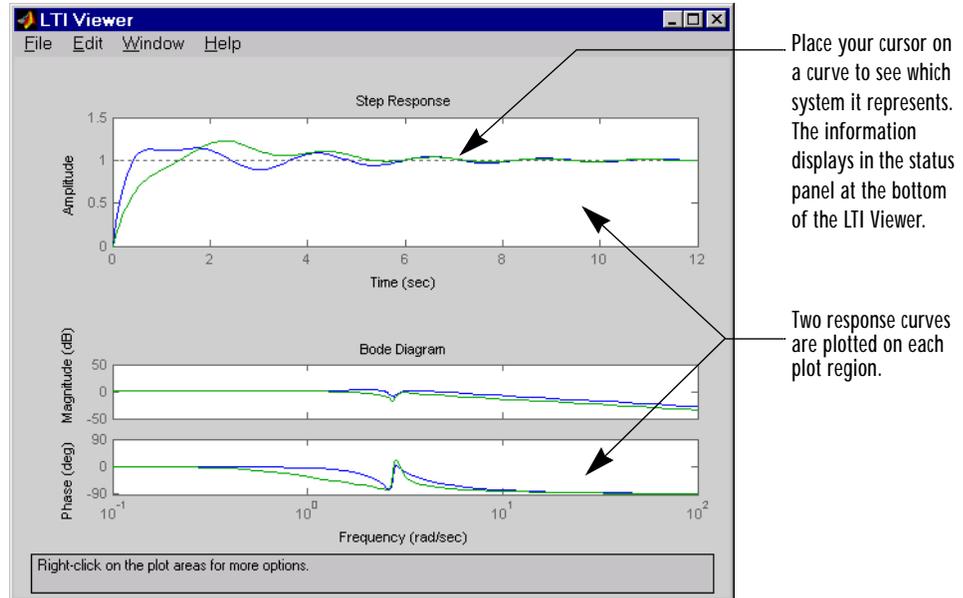
To load the two models `Gc11`, and `Gc12` into the LTI Viewer, select **Import** under the **File** menu and select the desired models in the LTI Browser. See “Importing Models into the LTI Viewer” on page 3-4 for a description of how to select groups of models. If necessary, you can reconfigure the viewer to display both the step responses and 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. Type

```
ltiview({'step';'bode'},Gc11,Gc12)
```

to do this.

Either approach opens the following LTI Viewer.



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

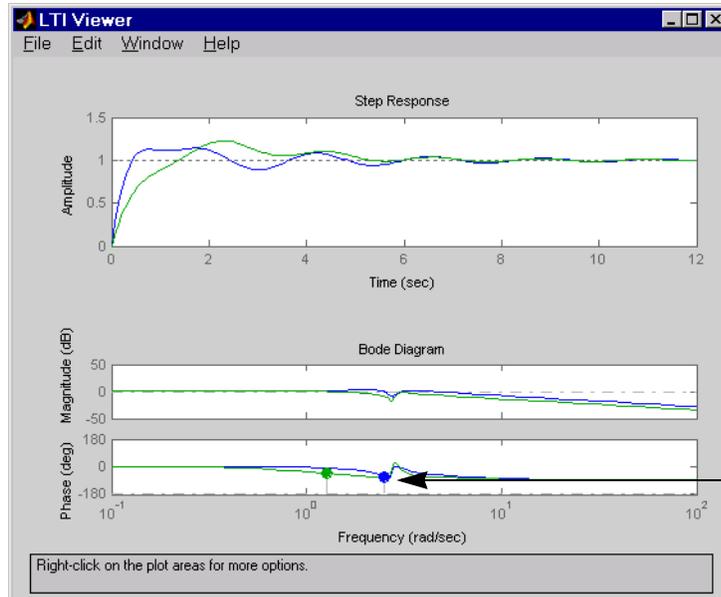
#### Inspecting Response Characteristics

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 (min)** under the **Characteristics** menu item.

Your LTI Viewer should now look like this.



Place your cursor on the blue or green dots to see phase margin data.

**Figure 3-11: Multiple Plots with Response Characteristics Added**

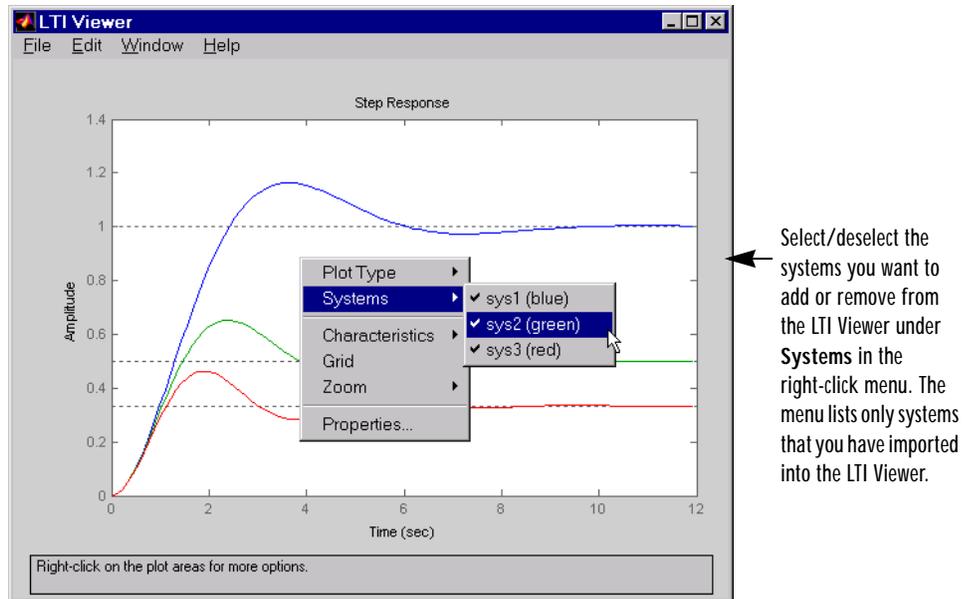
The minimum stability margins, meaning the smallest magnitude phase and gain margins, display as green and blue markers on the Bode phase diagram. If you want to see all the gain and phase margins of a system, select **Stability Margins (all)** under the **Characteristics** menu item.

#### Toggle Model Visibility

If you have imported more than one model, you can select and deselect 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 picture shows how to deselect the second of the three models using right-click menu options.



**Figure 3-12: Using Right-Click Menus to Select/Deselect 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.

## 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 plots
- Nichols plots
- Nyquist plots
- Singular value plots

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
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.

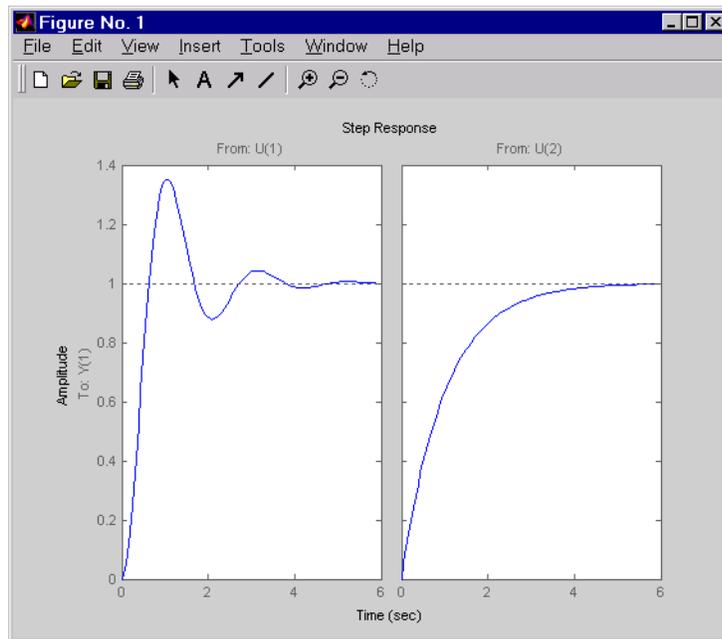
Frequency-domain plots automatically generate an appropriate frequency range as well.

## 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,

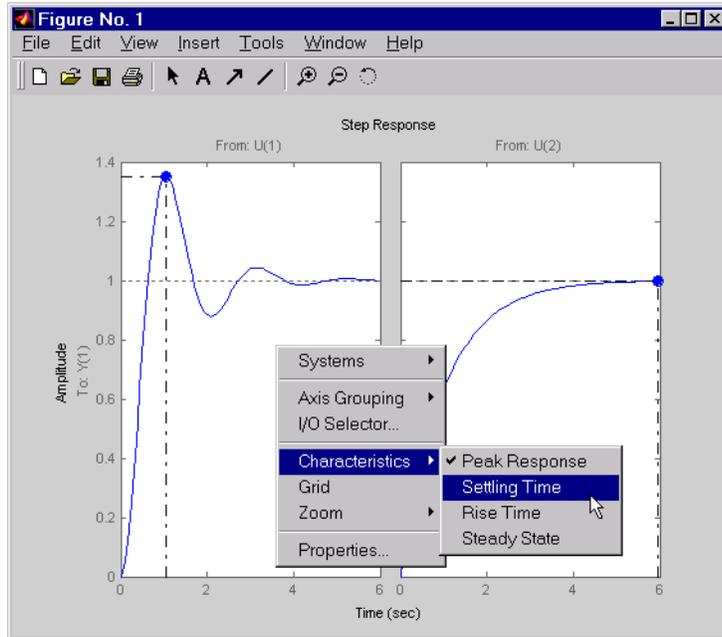
```
step(h,10) % Simulates from 0 to 10 seconds
```

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 Menu

All the time and frequency response functions provide right-click menus that allow you to customize your plots. This picture shows the plots from Figure 3-13, Step Responses for a MIMO Model, on page 3-19, with the right-click menu open.

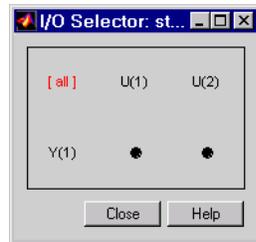


**Figure 3-14: Using the Right-Click Menu in a Step Response Plot**

The options you can select include:

- **Systems** — Select or deselect 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 Window**.



Use this window to select/deselect which inputs and outputs to plot.

- **Zoom** — Zoom in and out of plot regions.
- **Grid** — Add grids to your plots.
- **Properties** — Open the Property Editor, which you can use to customize various attributes of your plot. See Customization in the Control System Toolbox online documentation 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.

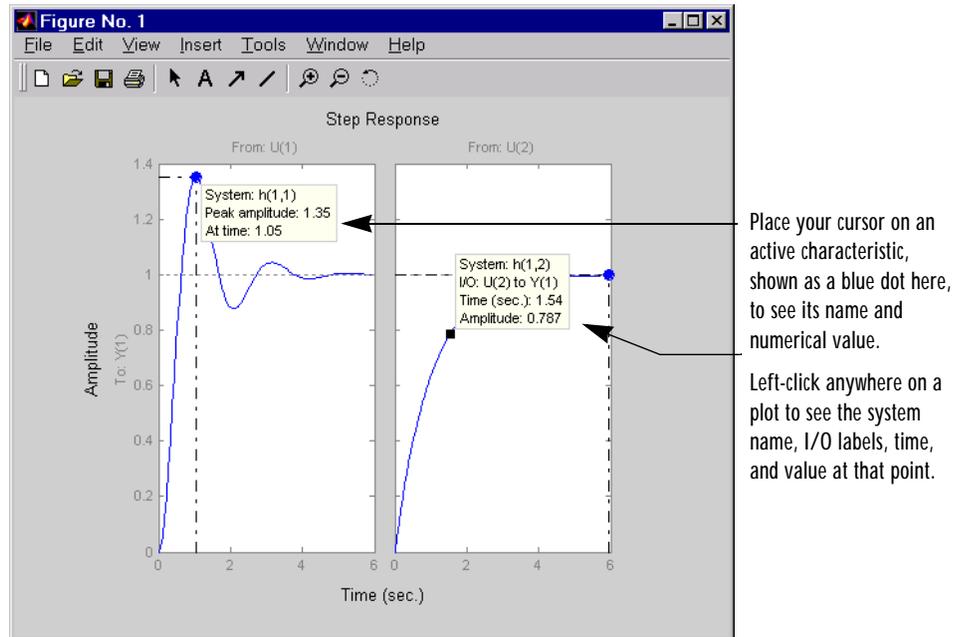


Figure 3-15: Using Plot Markers to Identify Data Points

You can move a data marker by:

- Grabbing the black 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 on the marker.

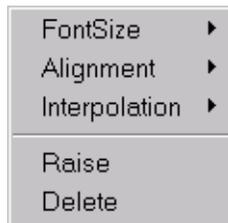
---

**Note** Data markers do not apply to the SISO Design Tool, which displays data about plot characteristics in the status panel at the bottom of the SISO Design Tool window.

---

### Right-Click Menu

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



Property options for the marker include:

- **Font Size** — Change the font size.
- **Alignment** — Change the position of the marker. Available options are top-right, top-left, bottom-right, and bottom-left.
- **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.
- **Raise** — If two markers overlap, raise brings the selected marker to the front.
- **Delete** — Delete the selected marker. Alternatively, left-click anywhere in the empty plot region to delete all markers in the plot.

### 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.

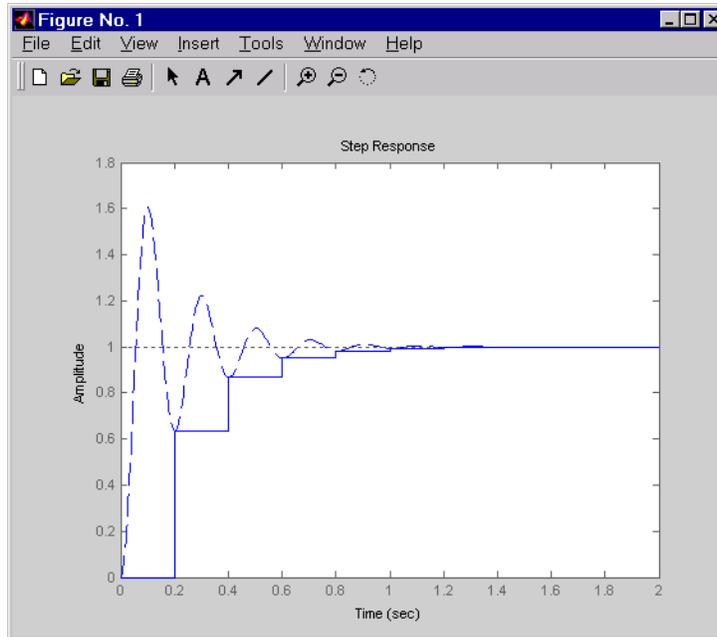
You can plot responses of multiple models on the same plot. These models do not need to be all continuous-time or all discrete-time.

### Example: Comparing Continuous and Discretized Systems

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

```
syc = tf(1000,[1 10 1000])  
sysd = c2d(syc,0.2)          % ZOH sampled at 0.2 second  
  
step(syc,'--',sysd,'-')    % Compare step responses
```

These commands produce the plot shown below.

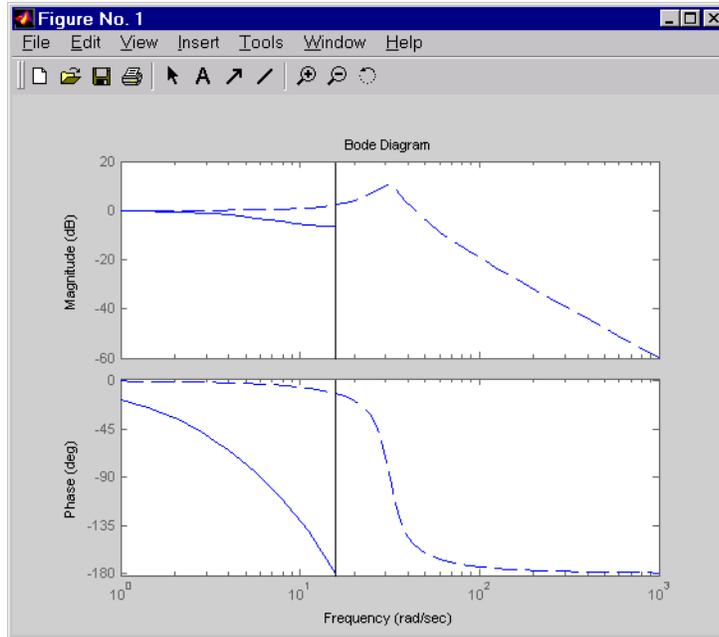


**Figure 3-16: Comparison of a Continuous Model to Its Discretized Version**

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

```
bode(syssc, '- - ',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

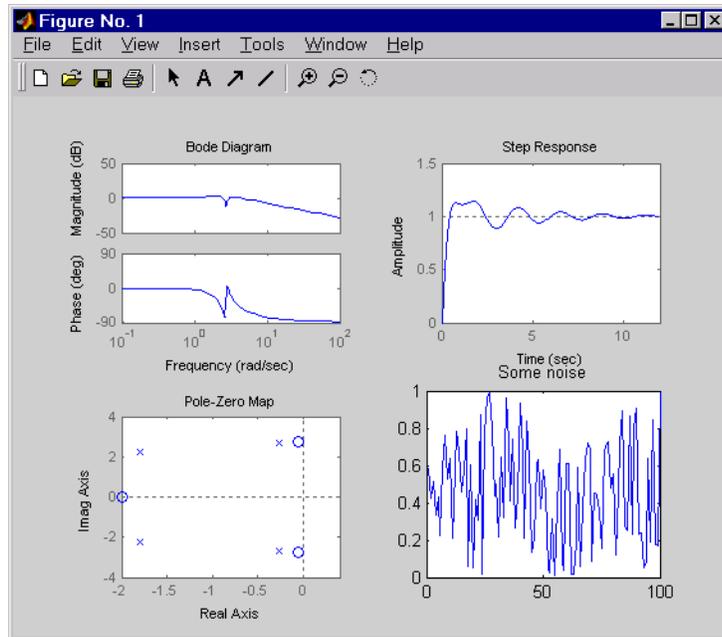
For example, the following sequence of commands displays the Bode plot, step response, pole/zero map, and some additional data in a single figure window.

```

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)
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 MATLAB's 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.

---

## Simulink LTI Viewer

If you have Simulink, you can use the Simulink LTI Viewer, a version of the LTI Viewer that performs linear analysis on any portion of a Simulink model.

The Simulink LTI Viewer features:

- Drag-and-drop blocks that identify the location for the inputs and outputs of the portion of a Simulink model you want to analyze.
- The ability to specify the operating conditions about which the Simulink model is linearized for analysis in the LTI Viewer.
- Access to all time and frequency response analysis tools featured in the LTI Viewer.
- The ability to compare a set of (linearized) models obtained by varying either the operating conditions or some model parameter values.

### Using the Simulink LTI Viewer

To learn about the Simulink LTI Viewer, this section presents an example that shows you how to linearize a portion of the Simulink model for an F14 jet aircraft. To open the F14 jet aircraft model, type

```
f14
```

at the MATLAB prompt.

This opens the following diagram.

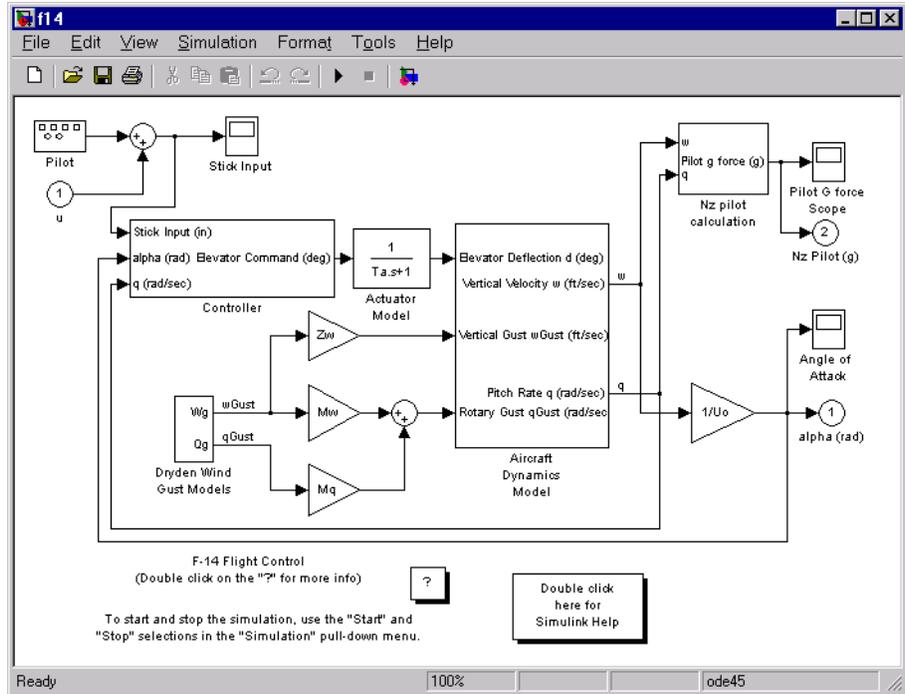


Figure 3-19: Nonlinear Model of F14 Aircraft Pitch Axis

This Simulink model, f14, contains static nonlinearities. You can use the Simulink LTI Viewer to analyze linearized versions of this model.

### Comparing Linear and Nonlinear Models

Suppose you want to compare the linearized step response between the Stick Input to the Pitch Rate,  $q$ , to the step response of the original nonlinear model.

The basic procedure for carrying out this type of analysis is outlined in the remaining sections, which present the F14 example in detail.

## Opening the Simulink LTI Viewer

To open a Simulink LTI Viewer linked to the f14 Simulink model:

- 1 Go to the **Tools** menu on the Simulink model.
- 2 Select **Linear Analysis**.

When you select **Linear Analysis**, two new windows open, an empty Simulink LTI Viewer window and a Simulink diagram called **Model\_Inputs\_and\_Outputs**, which contains Input Point and Output Point blocks.

This figure shows how to open the **Model\_Inputs\_and\_Outputs** diagram.

Select **Linear Analysis** from the Simulink model **Tools** menu to open the Simulink LTI Viewer and the **Model\_Inputs\_and\_Outputs** window.

This set of Simulink blocks opens when you select **Linear Analysis**. Use these blocks to specify the inputs and outputs of the portion of the Simulink model you want to analyze.

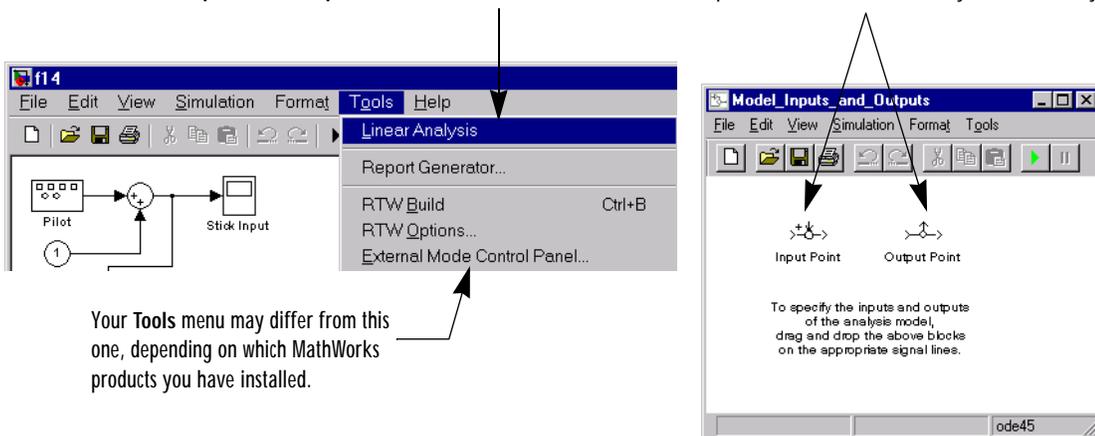


Figure 3-20: Opening the Model\_Inputs\_and\_Outputs Dialog Box from Simulink

The next section shows how you can select inputs and outputs for the linearized model.

### Specifying the Simulink Model Portion for Analysis

To specify the portion of the Simulink model you want to analyze, mark its input and output signals on the Simulink model using the Input Point and Output Point blocks in the **Model\_Inputs\_and\_Outputs** window. This defines an input/output relationship that is linearized and analyzed by the LTI Viewer. This to-be-linearized portion of your model is called the *analysis model*.

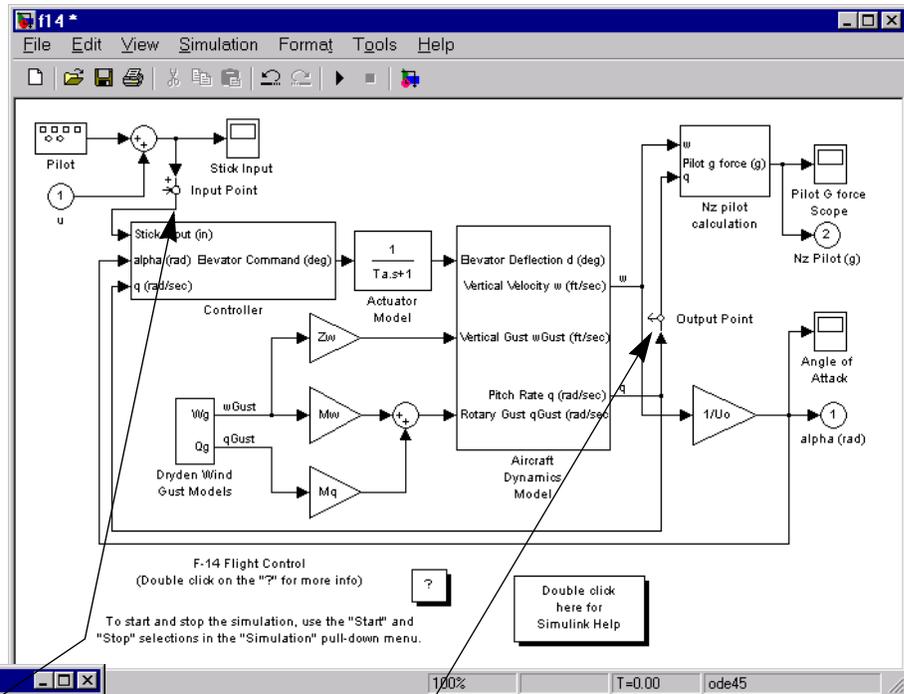
To designate the input and output signals of your analysis model, insert Input Point and Output Point blocks on the corresponding signal lines in your Simulink diagram.

To set up the analysis model for the f14 Simulink model:

- 1 Insert an Input Point block on the line going to the Stick Input.
- 2 Insert an Output Point block on the line labeled q (the pitch rate).

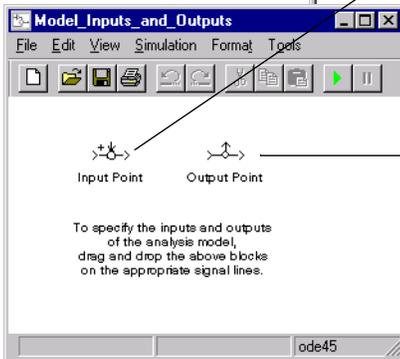
This picture illustrates the procedure for adding linearization input and output points to the f14 model.

Select an Input Point block, drag it, and release it on the Simulink model. Place it on the Stick Input line. This point will be the input to the linearized model.



To start and stop the simulation, use the "Start" and "Stop" selections in the "Simulation" pull-down menu.

Double click here for Simulink Help



Select an Output Point block and place it in the F14 model on the q (pitch rate) line.

Figure 3-21: Selecting Input and Output Points for the Linearized Portion of the F14 Model

Keep the following in mind when using the Input Point and Output Point blocks to specify analysis models:

- You can place the Input Point and Output Point blocks on any scalar or vector signal line in the Simulink model, with the exception of signal lines connected to any block in the Power System Blockset.
- You can insert Input Point and Output Point blocks at different levels of a Simulink model hierarchy.
- There is no limit on the number of these blocks you can use. If you have multiple input and/or output points in your model, you can find each I/O system listed under the **Systems** menu available by right-clicking in the plot region.

### Default Operating Conditions

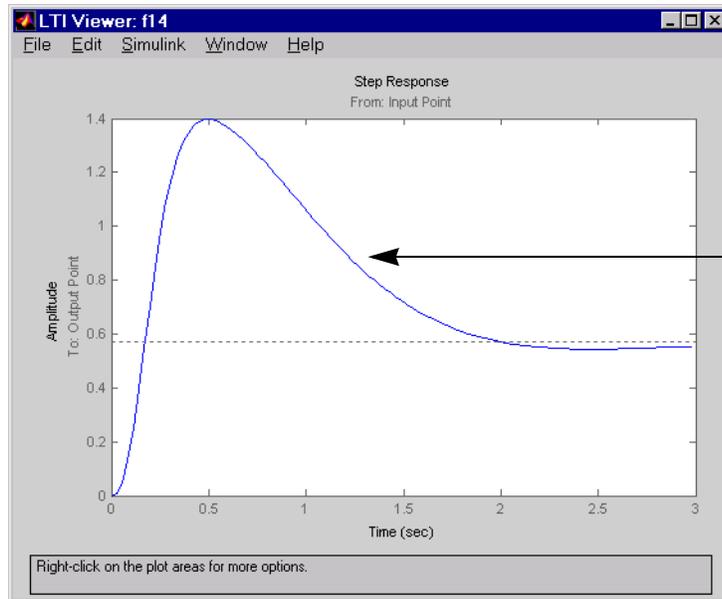
The *operating conditions* are the values of the state and input vectors about which the linearization of the analysis model occurs. *Initial conditions* are the values of the state and input vectors at time  $t=0$ . By default, Simulink sets the initial conditions to zero; for more information on setting initial conditions in Simulink, see the *Using Simulink* manual.

The Simulink LTI Viewer by default assumes that the operating conditions are the initial conditions set in Simulink for the model. This is the case in the F14 example, where the initial and operating conditions are set to zero. You can, however, select any operating conditions you require; see “Specifying Operating Conditions” on page 3-42 for more information.

### Performing Linear Analysis

Once you have specified your analysis model, and (optionally) set the operating conditions, you are ready to analyze it with the Simulink LTI Viewer.

To perform the linearization and import your linearized f14 model into the LTI Viewer, select **Get Linearized Model** on the **Simulink** menu on the LTI Viewer. This produces the following response plot.



The Simulink LTI Viewer displays the selected response plot for the linearized model.

The default plot type is a step response.

**Figure 3-22: Step Response from Stick Input to Pitch Rate for the Linearized F14 Model**

Each time you select **Get Linearized Model** in the LTI Viewer's **Simulink** menu, the LTI Viewer relinearizes the model and adds the new analysis model to the list of models available to the viewer. The Simulink LTI Viewer maintains a history of the linearized models, which is useful for comparing models when varying parameters.

You can view the linearized models in the LTI Viewer or change the plot type using the right-click menus.

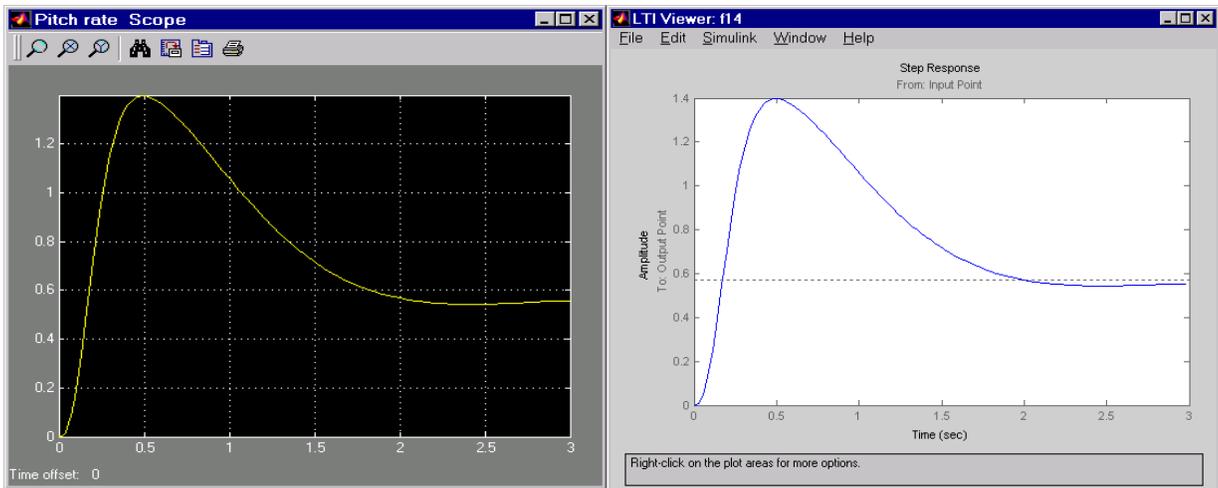
### Comparing the Step Response of the Linear and Nonlinear F14 Models

To compare the linear and nonlinear step responses, you must generate a step response for the original (nonlinear) f14 model.

To do this, follow these steps:

- 1 Replace the Signal Generator block with a Step block. Set the **Step time** to 0 in this block.
- 2 Add a Scope block to the line labeled q (pitch rate) in the diagram.
- 3 Set the **Stop time** in the **Parameters** dialog box to 3 seconds.
- 4 Run the model.

The output to the Pitch Rate Scope block is on the left side of this picture.



**Figure 3-23: Comparison of Step Responses from Stick Input to Pitch Rate for the Nonlinear and Linear F14 Models, Respectively**

As these two graphs demonstrate, the agreement between the linear and nonlinear models is quite good.

## Bode Plots of Linearized Models

You can use the Simulink LTI Viewer to compare Bode plots for various controllers. For example, you can change the gain  $K_i$  in the F14 controller shown below and compare the Bode magnitudes.

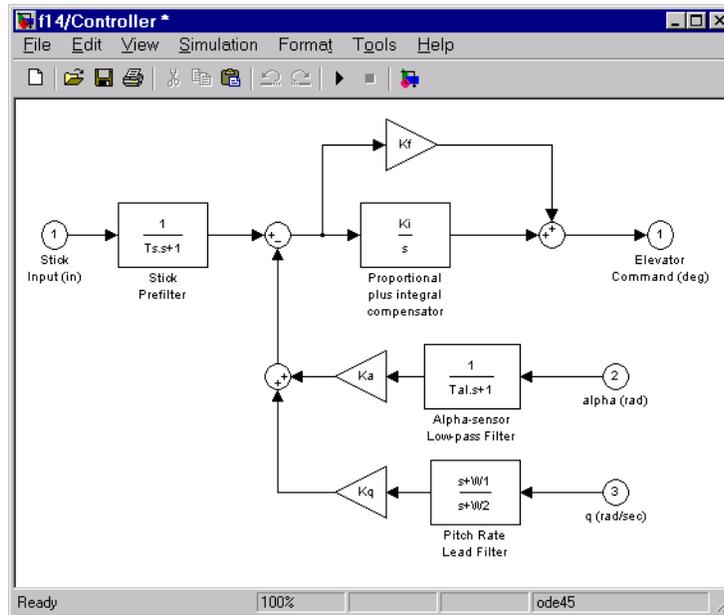
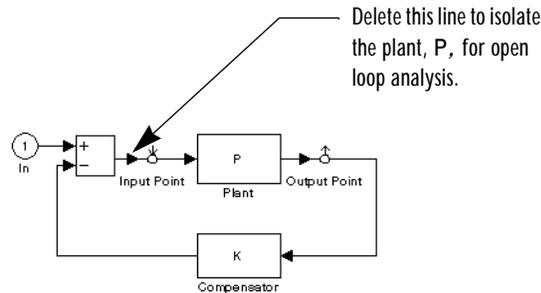


Figure 3-24: F14 Controller

#### Open-Loop Bode Response for the F14 Model

To specify open-loop analysis models, you must understand how Simulink performs linearization on the model. Placing the Input Point and Output Point blocks on your Simulink model does not break any connection or isolate any component.

For example, consider the following simple diagram.



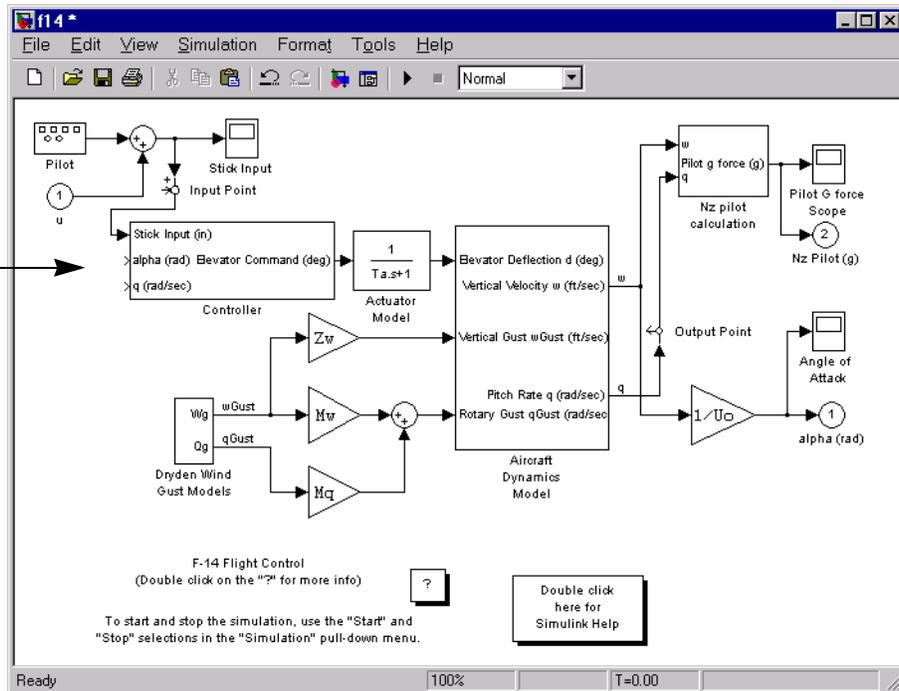
Based on the location of the Input Point and Output Point blocks, you might think that the analysis model specified by these blocks is simply the plant model,  $P$ . However, due to the feedback loop, this analysis model is actually the closed-loop transfer function  $P/(1 + PK)$ .

If you want to analyze the (open-loop) plant  $P$  instead, you need to open the loop, for example, by deleting the line between the Sum and Input Point blocks.

By default, the Simulink LTI Viewer performs closed-loop analysis whenever your diagram contains feedback loops between your Input and Output Point blocks. If you want an open loop Bode response plot, you must open the feedback path somewhere in the model by removing feedback lines.

In the F14 example, one way to create an open-loop model is to delete the feedback lines from pitch rate,  $q$ , and angle of attack,  $\alpha$ , back to the F14 controller. This picture shows the f14 model with the feedback paths deleted.

This picture shows the deletion of the  $\alpha$  and  $q$  feedback loops to create an open-loop analysis model. Compare this to Figure 3-19, Nonlinear Model of F14 Aircraft Pitch Axis, on page 3-30.



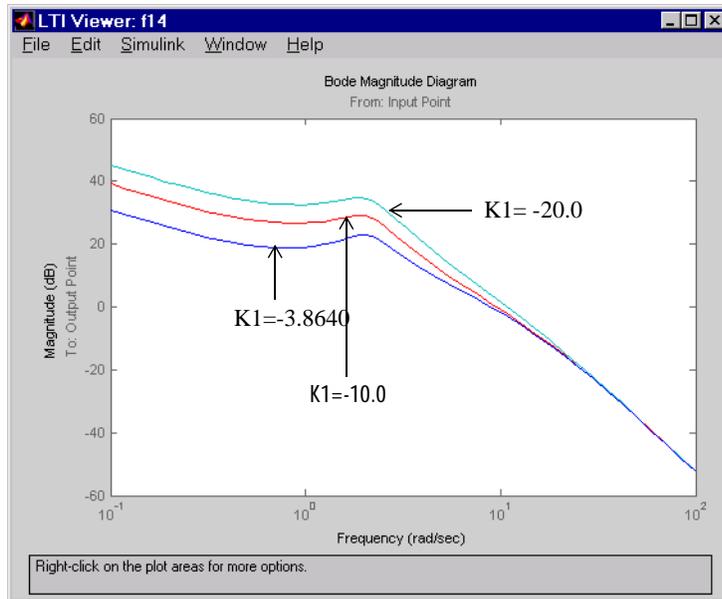
**Figure 3-25: F14 Model with the Pitch Rate,  $q$ , and Angle of Attack,  $\alpha$ , Feedback Paths Removed**

Once you have removed the feedback paths, you can proceed with open-loop analysis of your linearized model.

### Comparing Bode Magnitudes in the Simulink LTI Viewer

The Simulink LTI Viewer maintains a history of plots, so you can vary  $K_i$  in the controller and compare the changes in the Bode response. Each time you change  $K_i$ , select **Get Linearized Model** under the **Simulink** menu and the

viewer will plot the new results. This picture shows the open-loop Bode magnitude for three values of  $K_i$ .



**Figure 3-26: Open-Loop Bode Magnitudes for Three Values of  $K_1$**

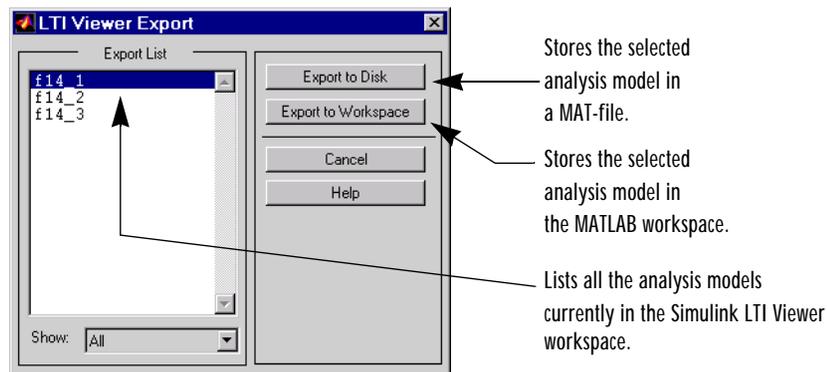
This example compares Bode magnitudes for several values of a parameter. You can, however, alter other elements of your analysis model for comparison. These elements include:

- Any of the analysis model parameters
- The operating conditions
- The location of any of the Input Point or Output Point blocks

## Saving Analysis Models

The analysis models obtained each time you select **Get Linearized Model** are stored only in the Simulink LTI Viewer workspace. You can save these models into the main MATLAB workspace by selecting **Export** from the Simulink LTI Viewer **File** menu.

Selecting **Export** opens the window shown below.



**Figure 3-27: Exporting Analysis Models From the LTI Viewer Workspace**

If you save models to a MAT-file, MATLAB prompts you to name the file. The variable names contained in that file are the same as those you selected from the **Export List**. The variable names of each model you save to the MATLAB workspace are also the same as those listed in the **Export List**. It's up to you to modify the names of these variables after you've saved them.

## Removing Input Points and Output Points

There are two ways you can remove Input Point or Output Point blocks from the Simulink model:

- 1 One by one: Select the Input Point or Output Point block you want to remove and delete it as you would any other Simulink block.
- 2 All at once: To remove all Input Point and Output Point blocks, select **Remove Input/Output Points** from the **Simulink** menu in the LTI Viewer.

When you delete an Input Point or an Output Point block, the signal lines coming into and out of this block are automatically reconnected.

## Specifying Operating Conditions

If you have nonlinear components in your Simulink model, the Simulink LTI Viewer automatically linearizes them when you select **Get Linearized Model**. The Simulink LTI Viewer uses the initial state values you set in the Simulink diagram as default settings for linearization points for the states in the diagram. You also have the option to linearize about the operating conditions of your choice.

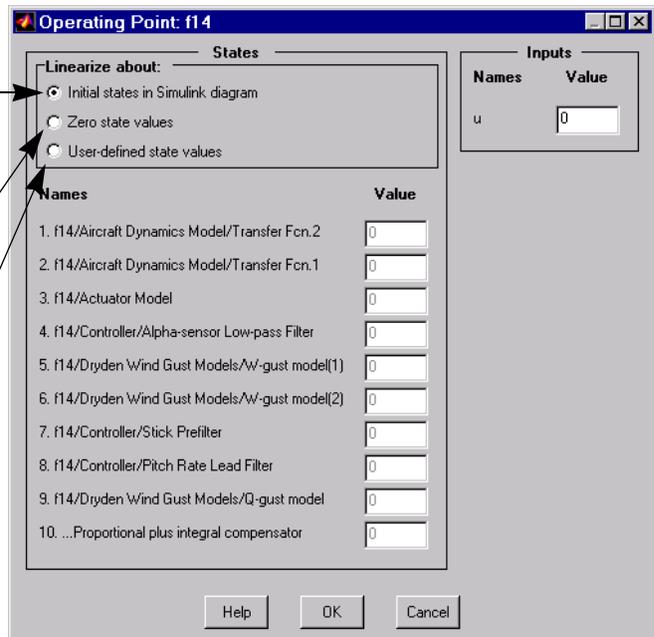
If you want your analysis model to be linearized about a zero state, or other state and input operating conditions, you must specify the desired operating points in the **Operating Point** dialog box *before* selecting **Get Linearized Model**.

To open the **Operating Point** dialog box, select **Set Operating Point** in the **Simulink** menu. This picture shows the open **Operating Point** dialog box.

Uses initial conditions for the states from the Simulink diagram. This is the default setting.

Sets the state values to zero for each linearization.

Sets the states to values specified in the Value fields. If you select this option, specify the state values in the Value field. Each state has its own field.

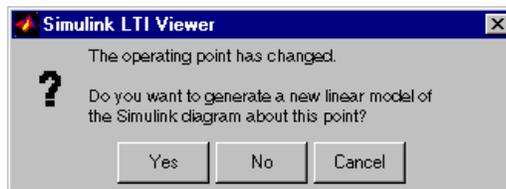


**Figure 3-28: The Operating Point Dialog Box for Changing Linearization Points**

To change the default operating point behavior, follow these steps:

- 1 Change the radio button selection (under **Linearize about:**) from the default to either:
  - Set all state values for the linearization to zero
  - Define your own state values for the linearization
- 2 Use the fields under **Value** to specify the operating conditions for each input (and state) listed in the **Operating Point** window. You do not have to specify the states if you choose the **Zero state values** radio button.

Once you have selected new operating conditions, press **OK**. This action opens a new dialog box, which is pictured below.



Select **Yes** to finish the process.

Note the following:

- The inputs listed on the **Operating Point** window correspond to the Inport blocks on the top level of your Simulink model.
- All states and inputs in the Simulink diagram are listed in the **Operating Point** window, not just those associated with your analysis model.
- If you want to change the operating conditions, you need only change those values associated with your analysis model.
- While the **Operating Point** window is in the **User-defined initial state values** mode, the values listed in the window remain in effect throughout your Simulink LTI Viewer session unless you change these.
- While the **Operating Point** window is in the **Initial state in Simulink diagram** mode, the linearization values used by the Simulink LTI Viewer are updated as you change any state values in your Simulink diagram.

- To change the initial state values of your Simulink diagram:
  - a Select **Parameters** under the **Simulation** menu on your Simulink diagram.
  - b Choose the **Workspace I/O** tab in the **Simulation Parameters** window.
  - c Load initial states from the MATLAB workspace using the appropriate field.

# Designing Compensators

---

<b>The SISO Design Tool . . . . .</b>	<b>4-3</b>
<b>Bode Diagram Design . . . . .</b>	<b>4-8</b>
<b>Root Locus Design . . . . .</b>	<b>4-27</b>
<b>Nichols Plot Design . . . . .</b>	<b>4-48</b>
<b>Functions For Compensator Design . . . . .</b>	<b>4-57</b>

This chapter discusses how to build compensators using functions from the Control System Toolbox. It begins with a description of the SISO Design Tool, a graphical user interface (GUI) that simplifies the task of designing controllers. Through two design examples, a DC motor and an electrohydraulic servomechanism, this chapter shows you how to use the SISO Design Tool to design compensators by root locus, Bode diagram, and Nichols plot design techniques, and how to analyze the resulting designs.

If you need to develop custom applications, or must perform MIMO (multiple input multiple output) design, the Control System Toolbox provides a set of commands that implement a variety of design algorithms, including root locus design, pole placement, and linear quadratic Gaussian (LQG) design.

## The SISO Design Tool

The SISO Design Tool is a graphical user interface (GUI) that facilitates the design of compensators for single-input, single-output feedback loops. The SISO Design Tool allows you to 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.

### 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,” imported.

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 the SISO Design Tool with the root locus and open loop Bode diagrams for the DC motor plotted by default.

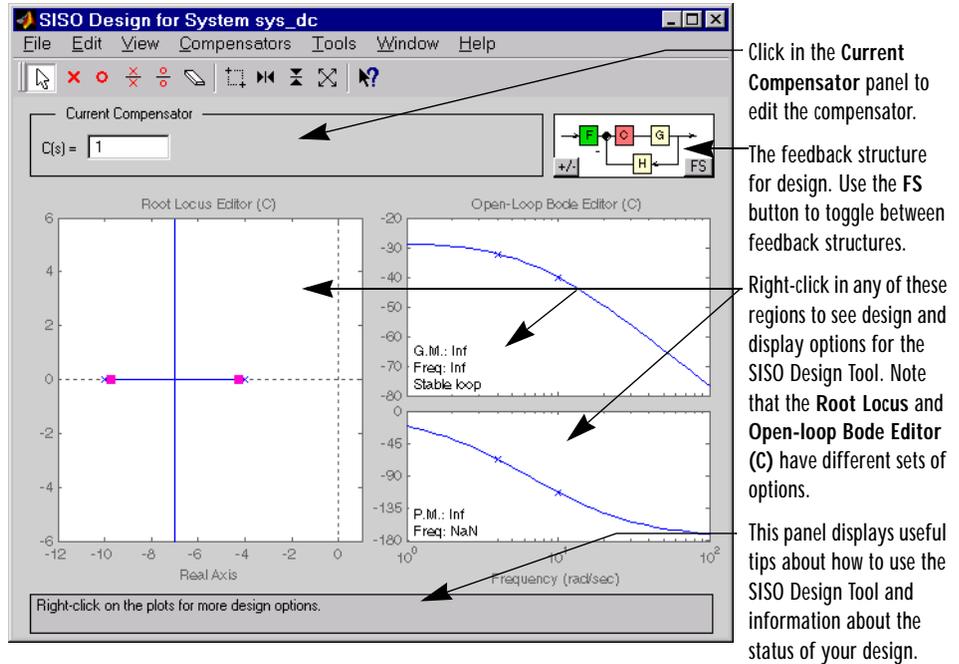


Figure 4-1: SISO Design Tool with the DC Motor Example

The SISO Design Tool displays:

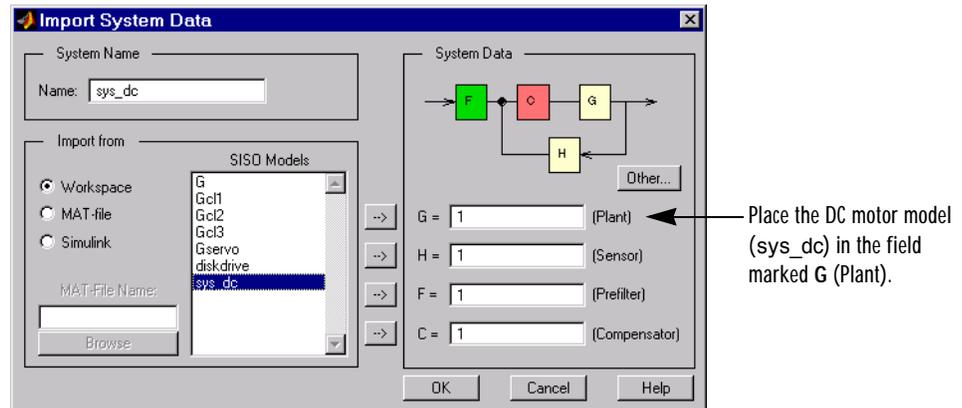
- Poles as x's
- Zeros as o's
- Gain and phase margins (by default) in the lower left-hand corners of the Bode magnitude and phase plots

## Importing Models into the SISO Design Tool

If you type

```
sisotool
```

at the MATLAB prompt, an empty SISO Design Tool opens. You can import the DC motor model by selecting **Import Model** under the **File** menu. This opens the **Import System Data** dialog box, which is shown below.



**Figure 4-2: Importing the DC Motor Model into the SISO Design Tool**

Follow these steps to import the DC motor model:

- 1 Select `sys_dc` under **SISO Models**.
- 2 Place it into the **G** field under **Design Model** by pressing the right arrow button to the left of **G**.
- 3 Press **OK**.

## 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 picture.

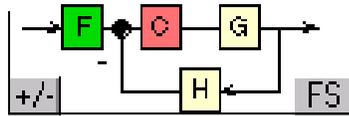


Figure 4-3: The Default Feedback Structure — Compensator in the Forward Path

In this picture, 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. Note that this means that by default, the compensator has unity gain. **G** contains the DC motor model, `sys_dc`.

### Alternative Feedback Structure

Clicking the **FS** button toggles between the default feedback structure and a feedback structure that places the compensator in the feedback path. This picture shows the alternate feedback structure.

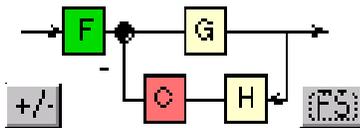
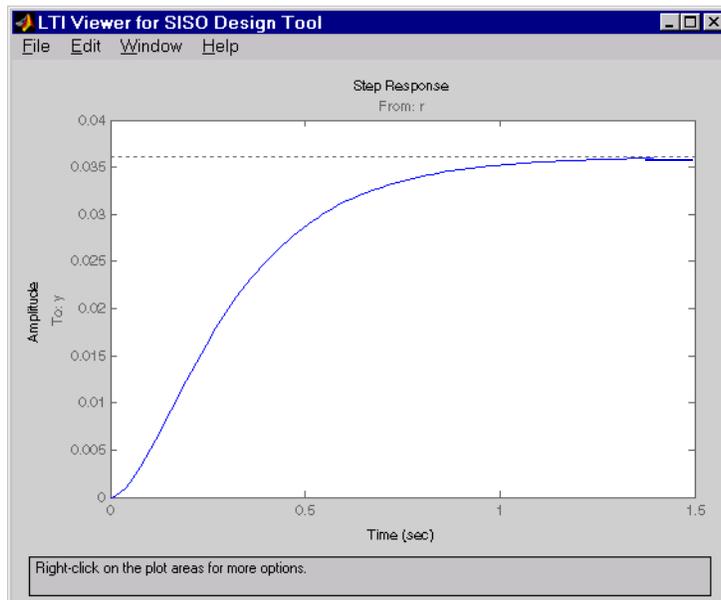


Figure 4-4: Alternate Feedback Structure with the Compensator in the Feedback Loop

## Loop Responses

As you iterate on a compensator design, you may find it convenient to be able to examine the various loop responses (for example, step or impulse responses). To view, for example, the closed-loop step response, select **Closed-Loop Step** under **Loop Responses** from the **Tools** menu. This opens an LTI Viewer with the closed-loop step response of the DC motor. For instructions on how to operate the LTI Viewer, see “LTI Viewer” on page 3-3.

This figure shows the resulting plot.



**Figure 4-5: 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 iterate on a design, the LTI Viewer associated with your SISO Design Tool will automatically update the response plots you have chosen.

## 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 2-4, the transfer function of the DC motor is

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

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

Figure 4-5 on page 4-7 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 Move the mouse pointer over the Bode magnitude line. Notice how the pointer becomes a hand.
- 2 Grab the Bode magnitude line by holding down the left mouse button when the hand appears.

- 3 Drag the Bode plot line upward.
- 4 Release the mouse button. The gain and poles changes as the closed-loop set point is recomputed.

The SISO Design Tool calculates the compensator gain, and the value appears in the **C(s)** text box on the GUI.

Alternatively, you can set the gain by entering the desired value in the **C(s)** field in the **Current Compensator** panel.

## Right-Click Menus

The SISO Design Tool has right-click menus available in any of the plot regions. The menus are customized for each plot type; open the Bode magnitude menu by right-clicking your mouse in the white space of the Bode magnitude plot. This menu appears.



**Figure 4-6: Right-Click Menu for the Bode Magnitude Plot**

The right-click menus contain numerous features. The DC motor example makes use of many of the available features; for a complete discussion of the right-click menus, see the online help for the SISO Design Tool in GUI Reference.

## Adjusting the Bandwidth

Since the design requirements include a 0.5 second rise time, try setting the gain so that the DC crossover frequency is about 3 rad/sec. The rationale for setting the bandwidth to 3 rad/sec 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 on 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/sec. This changes both the SISO Design Tool display and the LTI Viewer step response.

This figure shows the SISO Design Tool.

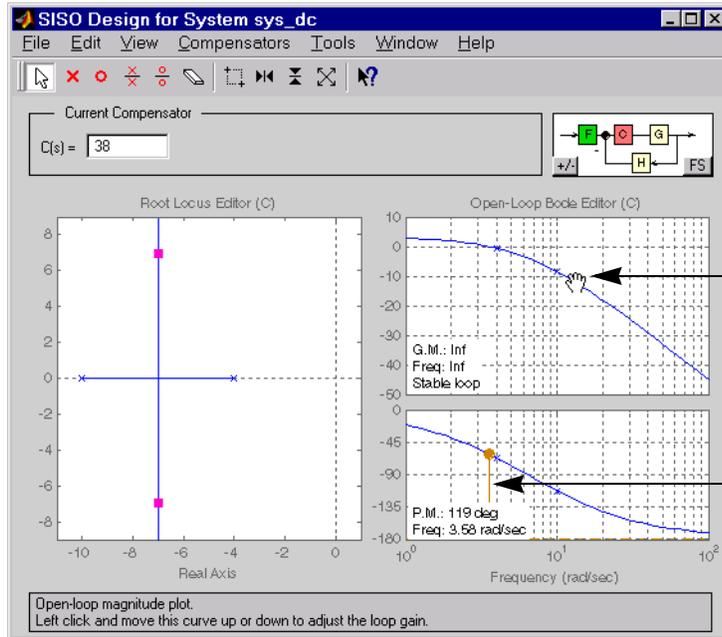
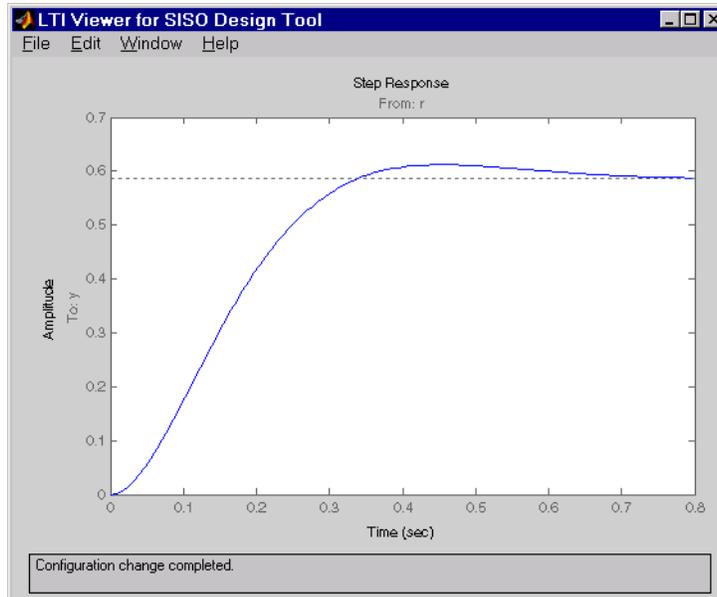


Figure 4-7: Root Locus and Bode Plots for the DC Motor

For a crossover at 3 dB, the compensator gain should be about 38. By default, the SISO Design Tool displays gain and phase margin information in the lower left-hand corners of the Bode diagrams. In the Bode magnitude plot, it also tells you if your closed-loop system is stable or unstable.

This plot shows the associated closed-loop step response in the LTI Viewer.



**Figure 4-8: 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 do this, select **Add Pole/Zero** and then **Integrator** from the right-click menu. This figure shows the process.

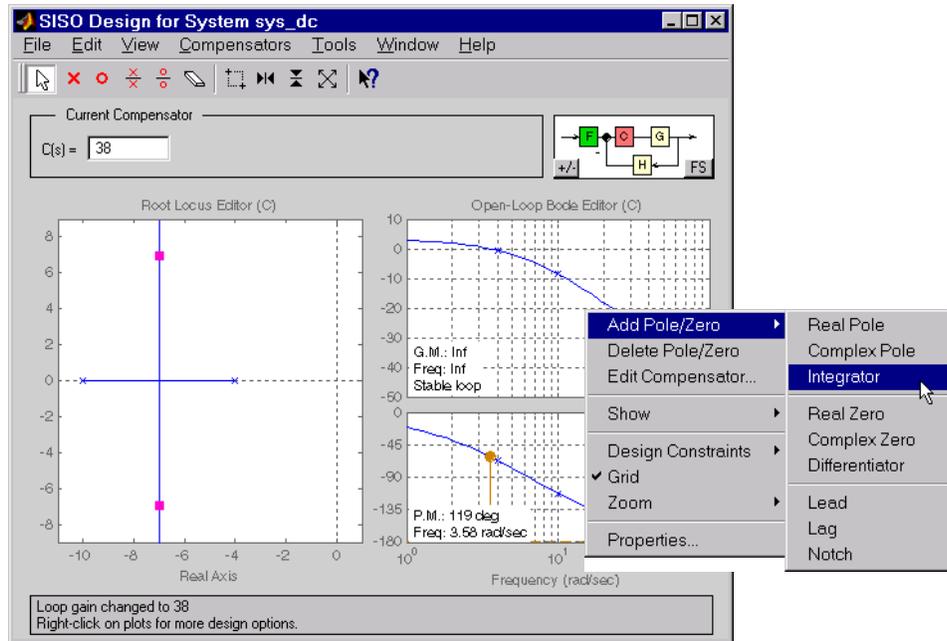
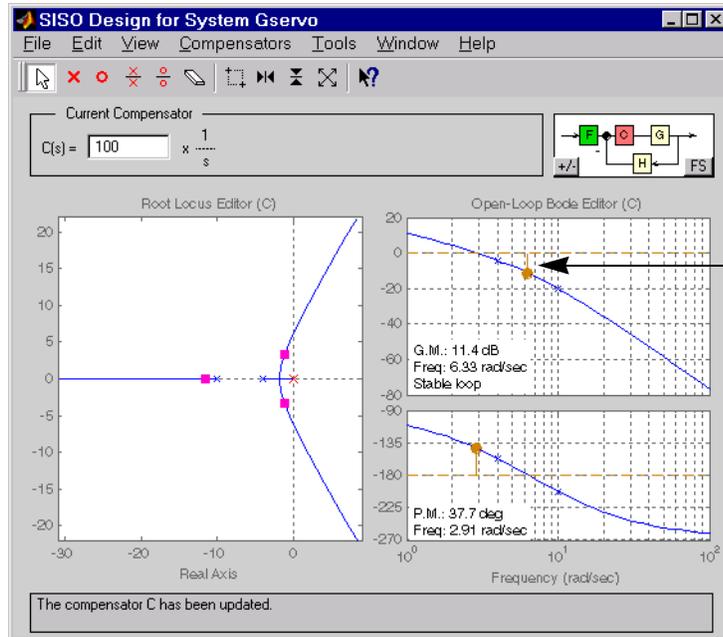


Figure 4-9: Using the Right-Click Menu to Add an Integrator

Notice adding the integrator changed the crossover frequency of the system. Readjust the compensator gain to bring the crossover back to 3 dB; the gain should be about 100.

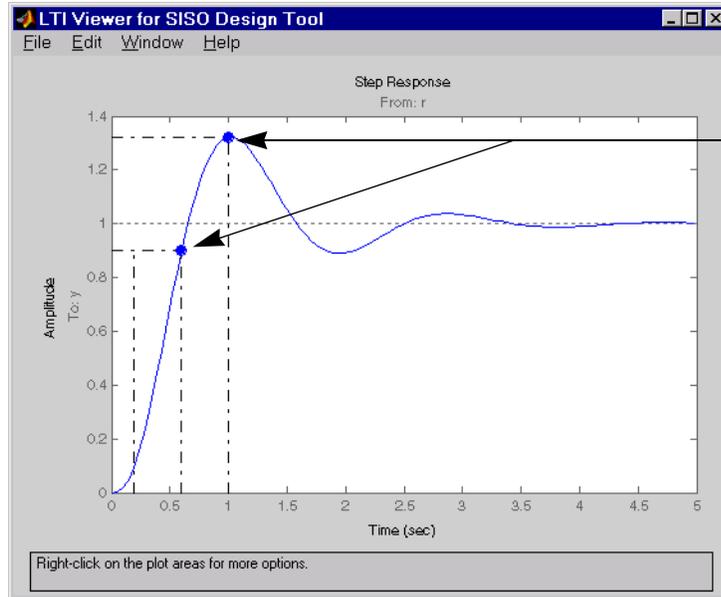
Once you have added the integrator and readjusted the compensator gain, the SISO Design Tool shows a red 'x' at the origin of the root locus plot.



Adding an integrator changed the gain margin from infinity to 11.5 dB. The SISO Design Tool displays the gain margin as a brown stem.

**Figure 4-10: The SISO Design Tool Displays the Integrator on the Root Locus Plot**

This figure shows the closed-loop step response.



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

**Figure 4-11: The 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  $40^\circ$  or more. In the current compensator design, the gain margin is 11.5 dB and the phase margin is  $38.1^\circ$ , 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 make this easier to do on the diagram, zoom in on the  $x$ -axis. First, select **Zoom In-X** from the right-click menu; then select a region of the Bode magnitude plot by left-clicking and dragging your mouse. The range from 1 to about 50 rad/sec is good. This picture shows the process.

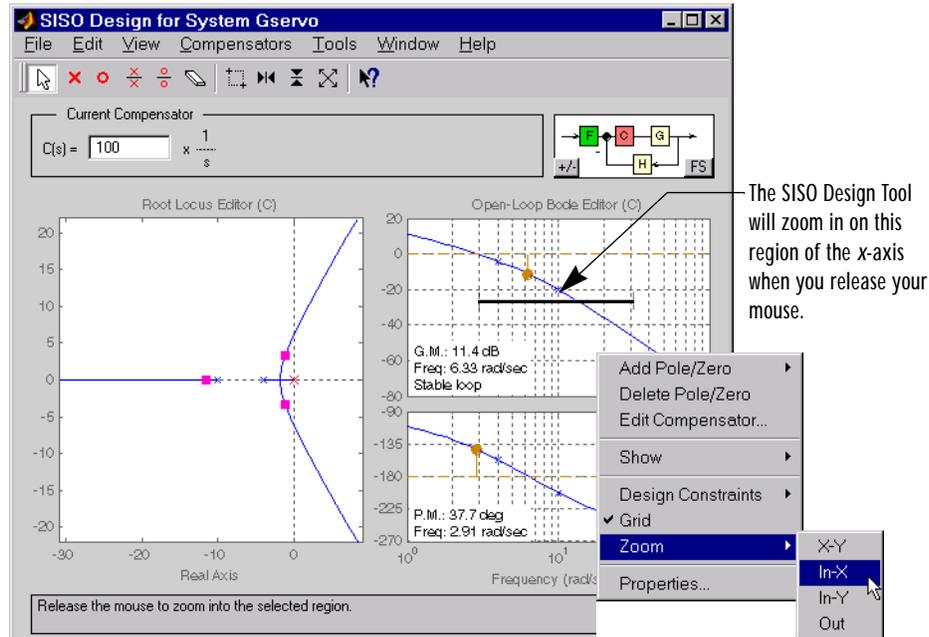
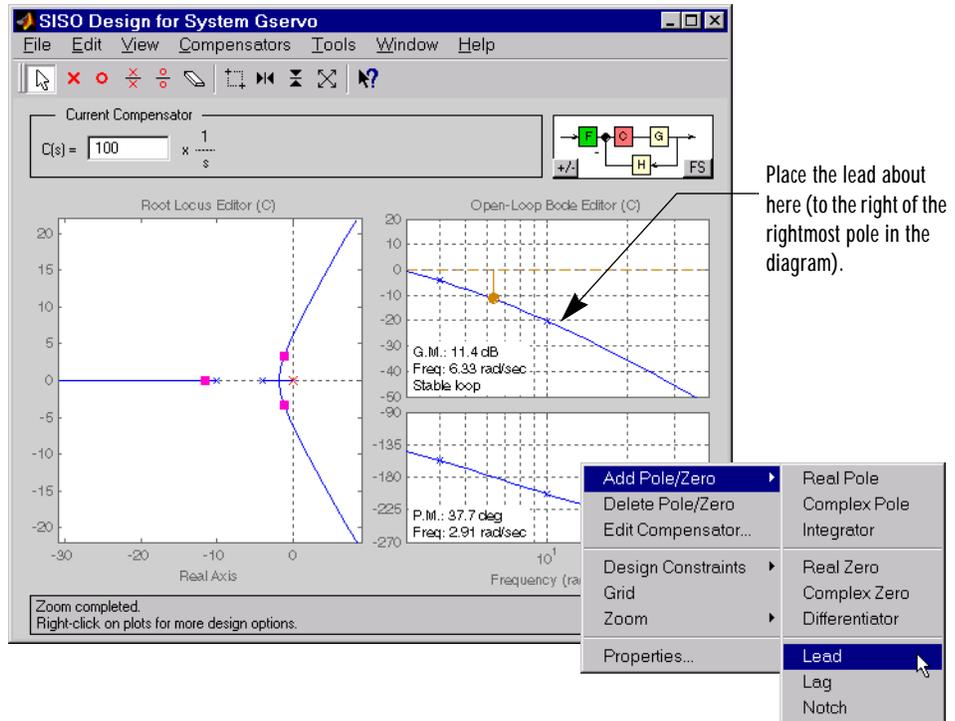


Figure 4-12: Zooming in on the X-Axis of the Bode Plots

To add the lead network, choose **Add Pole/Zero** and then **Lead** in the right-click menu for the Open-Loop Bode diagram. This figure shows the process of adding a lead network to your controller.



**Figure 4-13: Adding a Lead Network to the DC Motor Compensator Using Right-Click Menus**

Selecting a lead network causes your cursor to change to an 'x.' Position this 'x' on the Bode magnitude curve slightly to the right of the rightmost pole and

click. Your SIS0 Design Tool and LTI Viewer plots should now look similar to these.

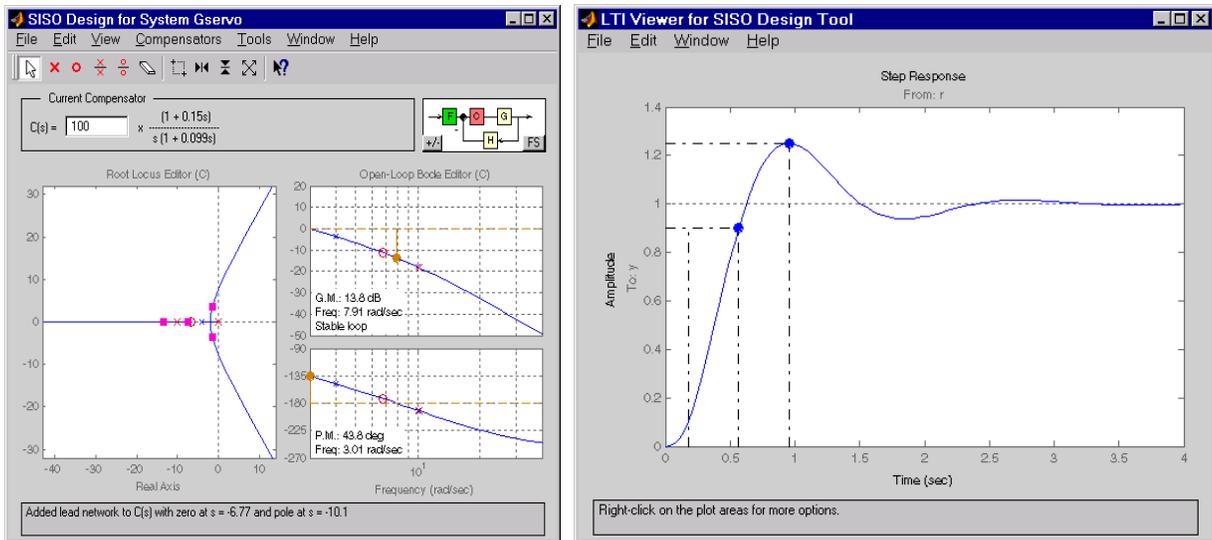


Figure 4-14: Root Locus, Bode, and Step Response Plots for the DC Motor with a Lead Network

The Step Response plot shows that the rise time is now about 0.4 second and peak response is 1.25 rad/sec (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, move the lead network zero closer to the leftmost (slowest) pole of the DC motor plant (denoted by a blue 'x'). To do this, just grab the zero and drag it with your mouse. Try positioning the zero near the slowest plant pole.

Now try moving the lead network pole to the right. Notice how the gain margin increases as you do this. You can also use the gain to increase the gain margin; grab the Bode magnitude curve and drag it upward with your mouse to see the gain and gain margin increase.

As you tune these parameters, take a look at the LTI Viewer. You will see the closed-loop step response alter with each parameter change you make. The figure below shows the final values for a design that meets the specifications.

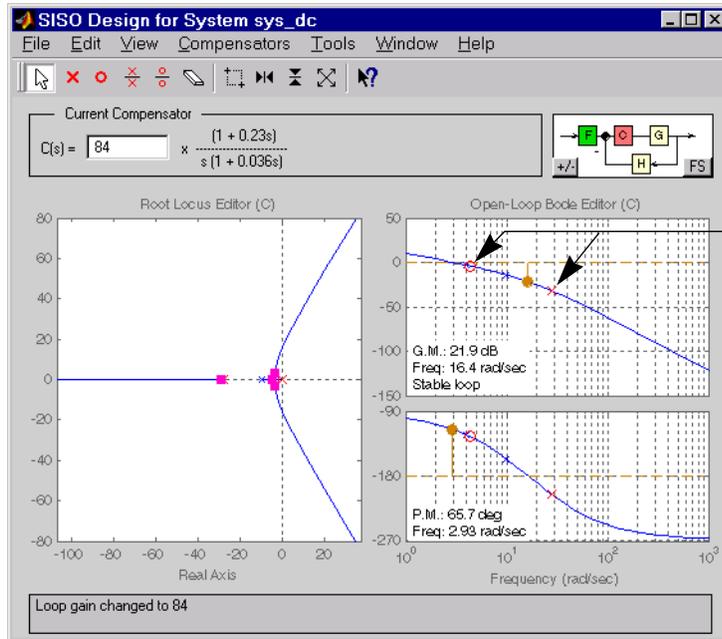


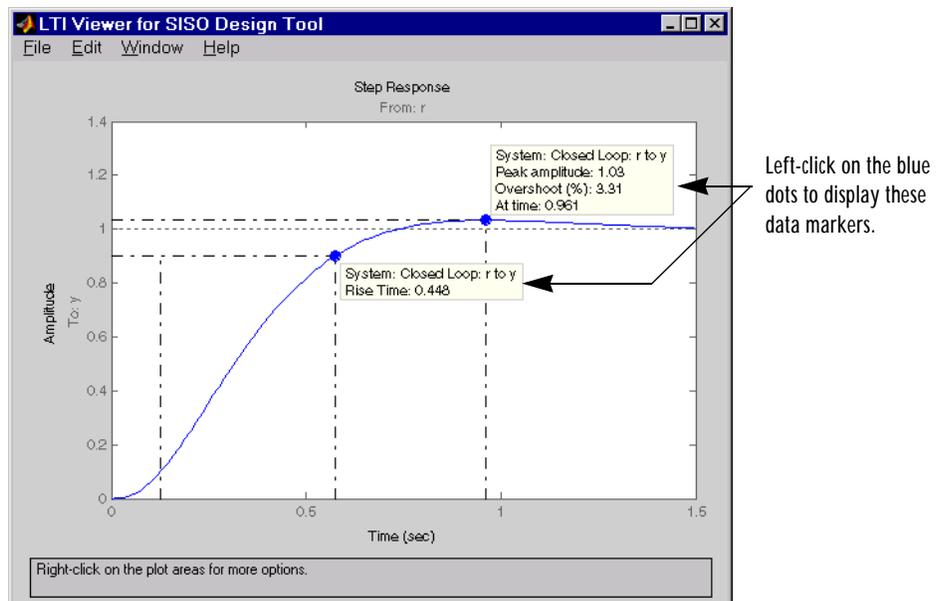
Figure 4-15: Final Design Parameters for the DC Motor Compensator

The values for this final design are as follows:

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

You can use the **Edit Compensator** dialog box to specify the exact values. Double-click in the Current Compensator panel to open the window. This figure shows that the gain margin is 22 dB, and the phase margin is 66°. To see if the design meets the rise time and overshoot requirements, go to the closed-loop step response, right-click in an empty region of the plot, and select

**Characteristics** and then **Rise Time** and **Peak Overshoot**. This figure shows the plot with the rise time and overshoot denoted by large dots on the curve.



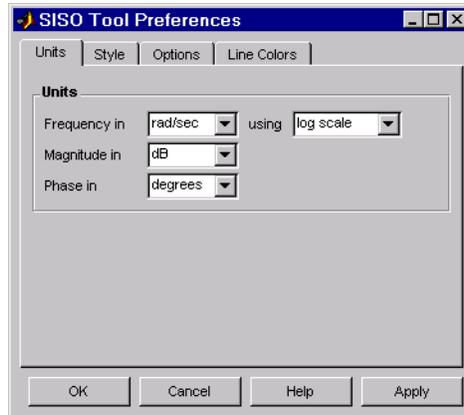
**Figure 4-16: Step Response for the Final Compensator Design**

The step response shows that the rise time is 0.45 second, and the peak amplitude is 1.03 rad/sec, or an overshoot of 3%. These results meet the design specifications.

## Changing Units on a Plot

The Control System Toolbox provides editors for setting plot options. If you want, for example, to change the frequency units on all the Bode plots created

in the SISO Design Tool from rad/sec to Hertz, select **SISO Tool Preferences** under **Edit** in the menu bar. This opens the **SISO Tool Preferences** editor.



**Figure 4-17: The SISO Tool Preferences Editor**

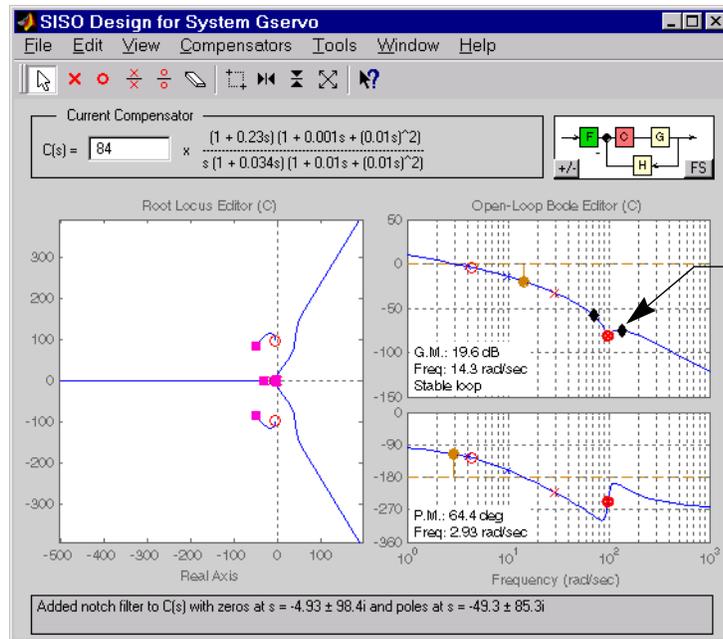
Use the menu options on the **Units** page to make the change. This unit change persists for the entire session.

For more information about property and preference setting, see “Customization” online under the Control System Toolbox.

### 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, select **Add Notch** from the right-click menu and place the filter at the frequency you want to attenuate. A black ‘x’ will appear next to the mouse arrow; place it at the frequency you want to attenuate.

This figure shows the result.



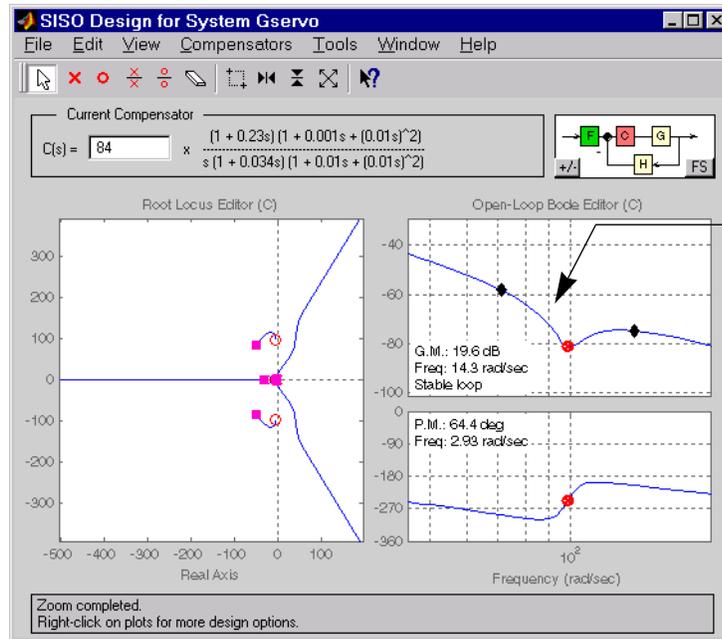
See Figure 4-15 for a close look at the notch filter parameters.

**Figure 4-18: The SISO Tool with a Notch Filter Added to the DC Motor Compensator**

Note that to add the notch filter it was necessary to zoom out, since the notch frequency is at a higher frequency than Figure 4-15 on page 4-18 displayed.

To see the notch filter parameters in more detail, select **Zoom X-Y** in the right-click menu for the Bode magnitude plot. Left-click and drag your mouse to draw a box around the notch filter. When you release the mouse, the SISO Design Tool will zoom in on the selected region.

This figure zooms in on the notch filter to show the adjustable parameters.



**Figure 4-19: Manipulating Notch Filter Parameters**

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  $\xi_1$ ,  $\xi_2$ , and  $\omega_n$ . The ratio of  $\xi_2/\xi_1$  sets the depth of the notch, and  $\omega_n$  is the natural frequency of the notch. This diagram

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

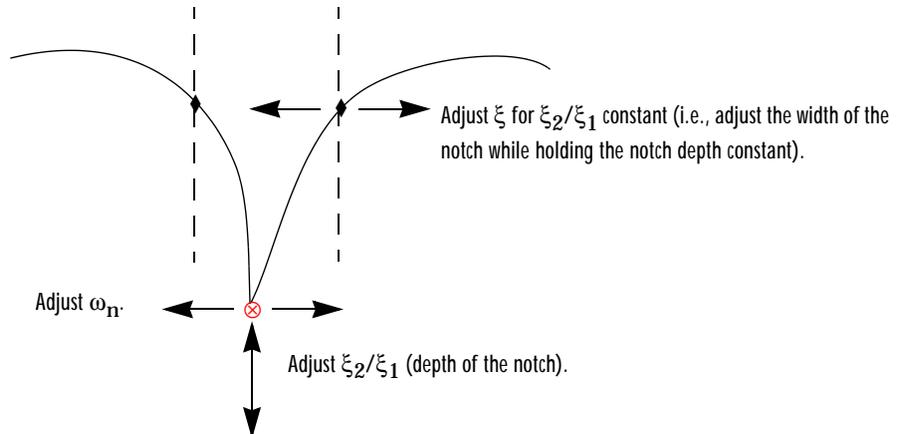


Figure 4-20: A Close Look at Notch Filter Parameters

## Adding a Prefilter

You can use the SISO Design Tool to add and modify a prefilter to 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. To do this, first open a Bode diagram for the prefilter by selecting **Prefilter Bode** from the **View** menu.

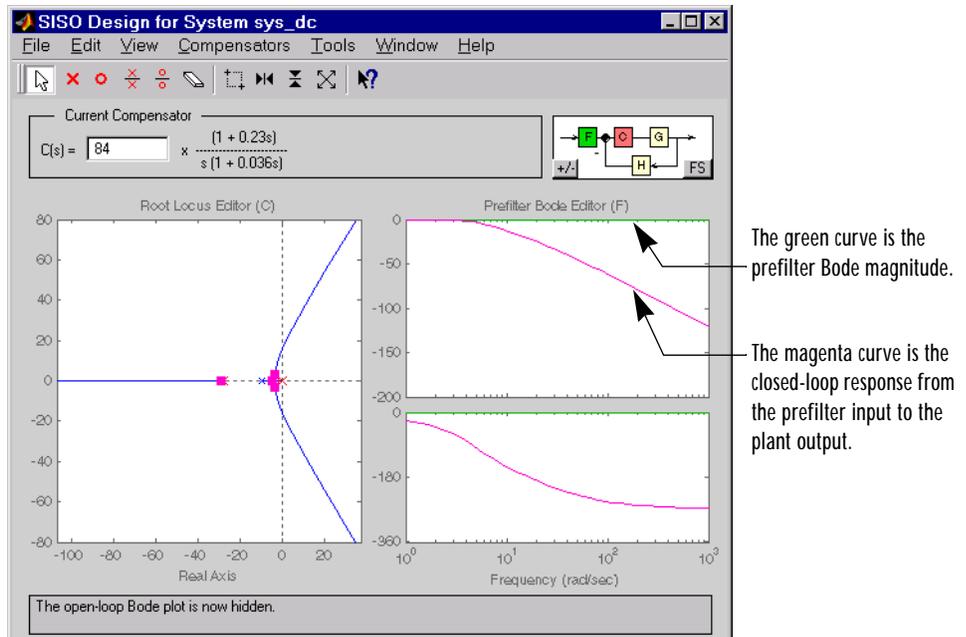


Figure 4-21: Opening the Prefilter Bode Diagram

For clarity, the above figure does not show the open-loop Bode diagram for the compensator (C). To remove the Bode diagram from the display, uncheck **Open-loop Bode** under the **View** menu.

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), including:

- The right-click menu
- The **Edit Compensator F** window
- The **Prefilter Bode** menu from **Edit**
- The toolbar

A quick way to create a lowpass roll-off filter is to add a pair of complex poles. To do this, first activate the grid for the prefilter Bode, then select **Add Pole/Zero** and **Complex Pole** from the prefilter Bode right-click menu. For this example, try to place the poles at about 50 rad/sec. This figure shows the poles added to the prefilter Bode diagram.

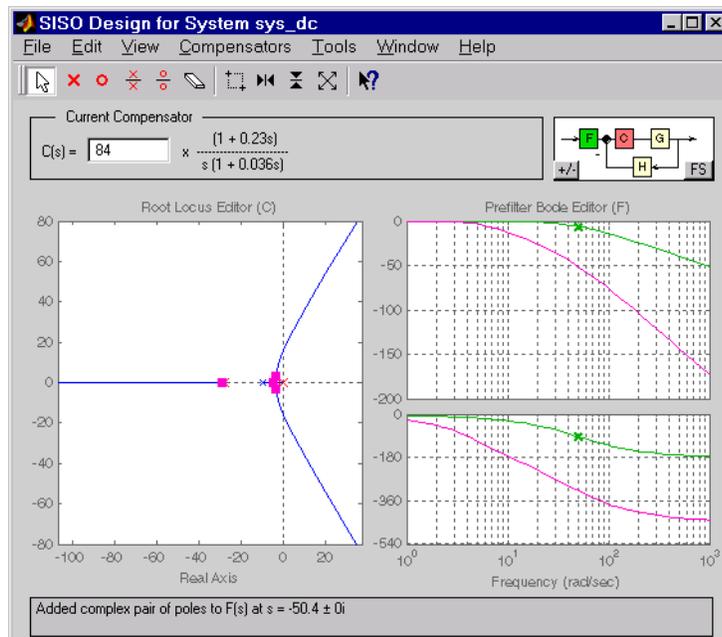


Figure 4-22: Adding a Complex Pair of Poles to the Prefilter Bode Diagram

By default, the damping ratio of the complex pair is 1.0, which means that there are two real-valued poles at -50.4 rad/sec. The green curve, which

represents the prefilter Bode response, shows the -3 dB point for the roll-off is at 50.4 rad/sec. 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 selecting **Import** from the **Edit** menu. This opens the **Import System Data** window.

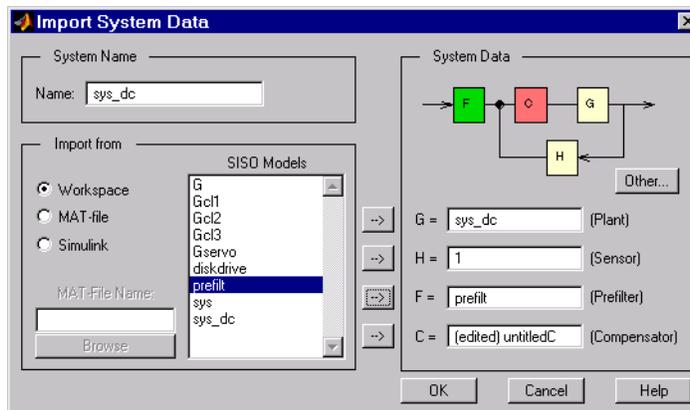


Figure 4-23: Importing a Prefilter

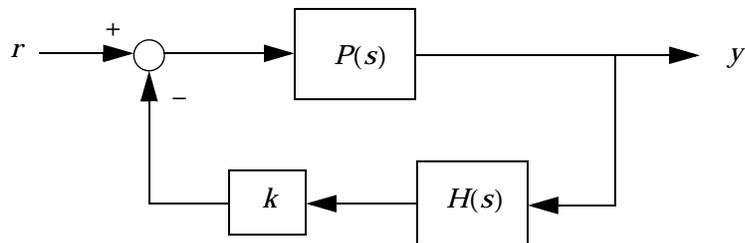
Select `prefilt` from the **SISO Models** list and press the arrow to the left of the **Prefilter** field to import the prefilter model. Once you've imported the prefilter model, you can modify it using the methods described above.

## 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.

The root locus diagram shows the trajectories of the closed-loop poles of a feedback system as a single system parameter varies over a continuous range of values. Typically, the root locus method is used to tune the loop gain of a SISO control system by specifying a feedback gain the 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. Since the SISO Design Tool displays 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.

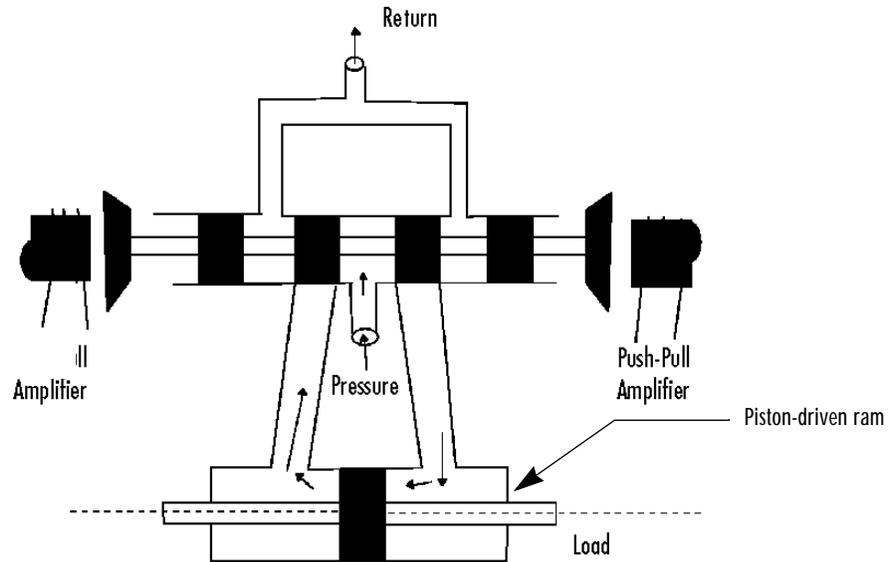


Figure 4-24: An Electrohydraulic Servomechanism

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 figure below.

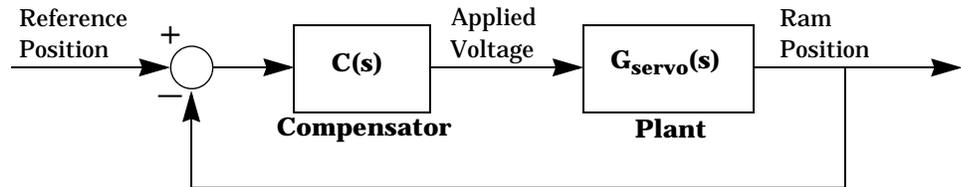


Figure 4-25: Feedback Control Structure for an Electrohydraulic Servomechanism

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 displays 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%.

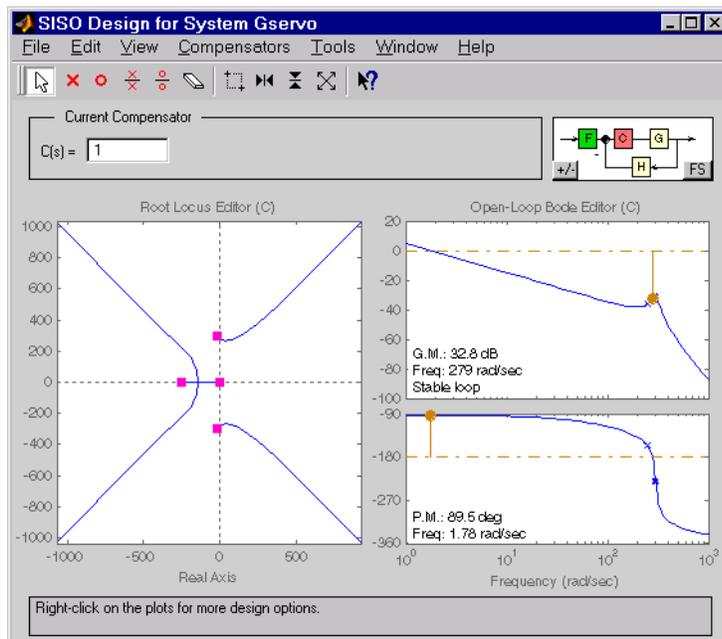
The remainder of this section discusses how to use the SISO Design Tool to design a controller to meet these specifications.

### Opening the SISO Tool

Open the SISO Design Tool and import the model by typing

```
sisotool(Gservo)
```

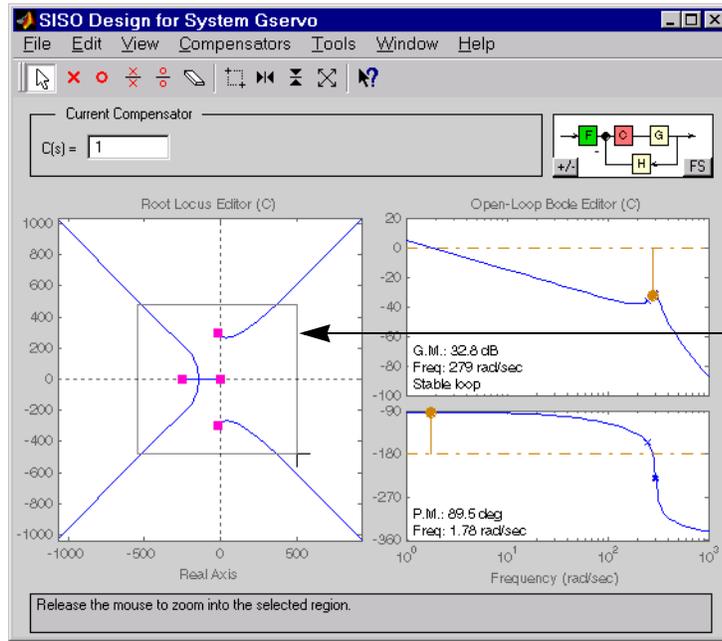
at the MATLAB prompt. This opens the SISO Design Tool with the servomechanism plant imported.



**Figure 4-26: SISO Design Tool Showing the Root Locus and Bode Plots for the Electrohydraulic Servomechanism Plant**

## Zooming

Using the right-click menu in the root locus, select **X-Y** under **Zoom**. Hold down 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 down your mouse's left button to select a rectangular region for zooming in. When you let go of the button, the SISO Design Tool replots the root locus with the new axis boundaries.

To undo the zoom, select **Zoom Out** in the right-click menu.

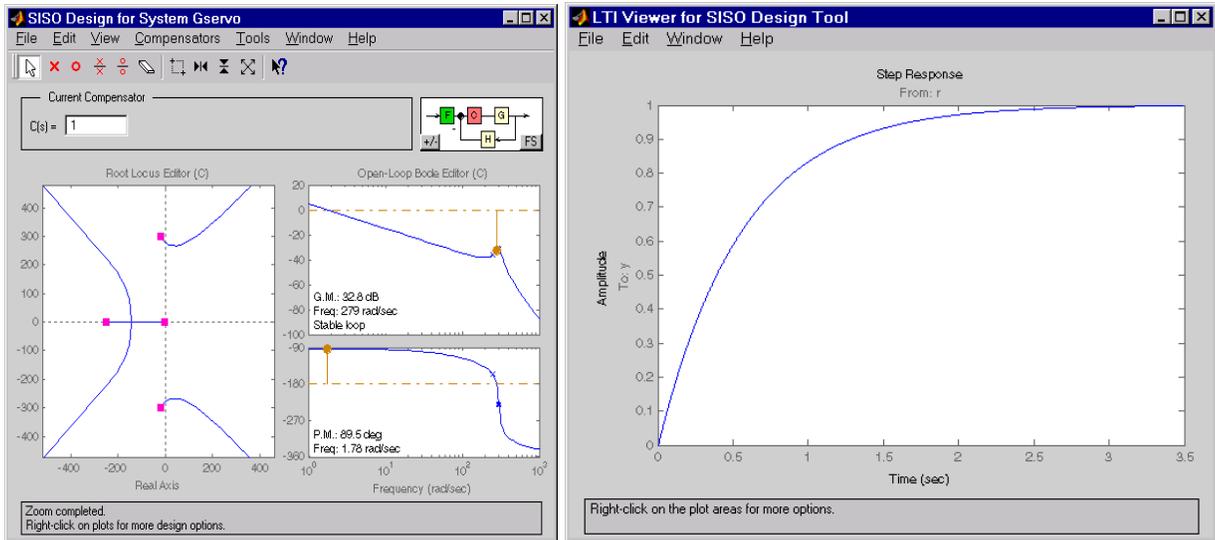
Figure 4-27: Zooming in on a Region in the Root Locus Plot

Alternatively, you use the zoom icons on the toolbar:

-  — Zoom in X-Y
-  — Zoom in X
-  — Zoom in Y
-  — Zoom out

As in the DC motor example, open an LTI Viewer by selecting **Closed-Loop Step** in the **Loop Responses** menu under **Tools** in the menu bar. You now

should have two windows, the SISO Design Tool and the associated LTI Viewer side by side.

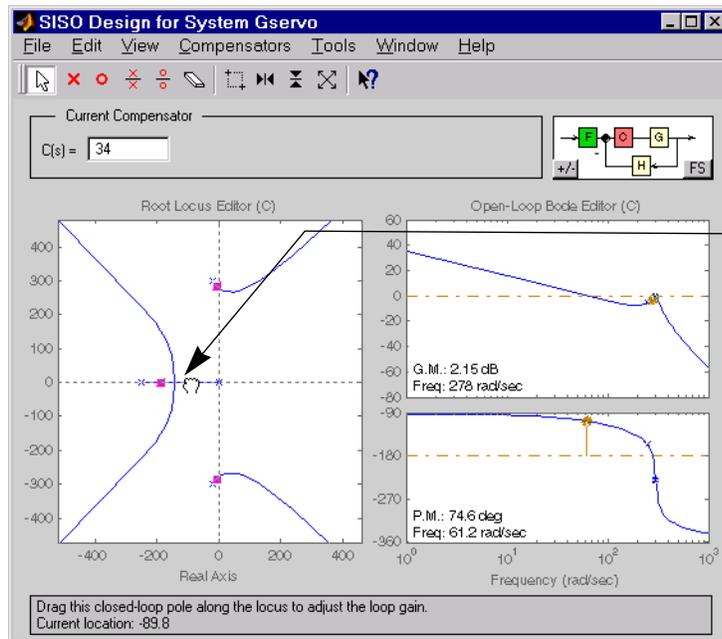


**Figure 4-28: SISO Design Tool and Associated LTI Viewer for the Electrohydraulic Servomechanism**

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 4-31.

## 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 grabbing the red squares on the root locus plot and moving them along the curve. This figure shows the procedure.



Move the red squares to change the compensator gain. The SISO Design Tool calculates the compensator gain  $C(s)$  and displays it in the **Current Compensator** panel.

**Figure 4-29: Changing the Compensator Gain in the Root Locus Plot**

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

Alternatively, you can change the compensator gain by entering values into the **C(s)** field in the **Current Compensator** panel.

### Closed-Loop Response

Change the gain to 20 by editing the text box next to **Gain**, and pressing the **Enter** key. Notice that the locations of the closed-loop poles on the root locus are recalculated for the new gain set point.

This figure shows the associated closed-loop step response for the gain of 20.

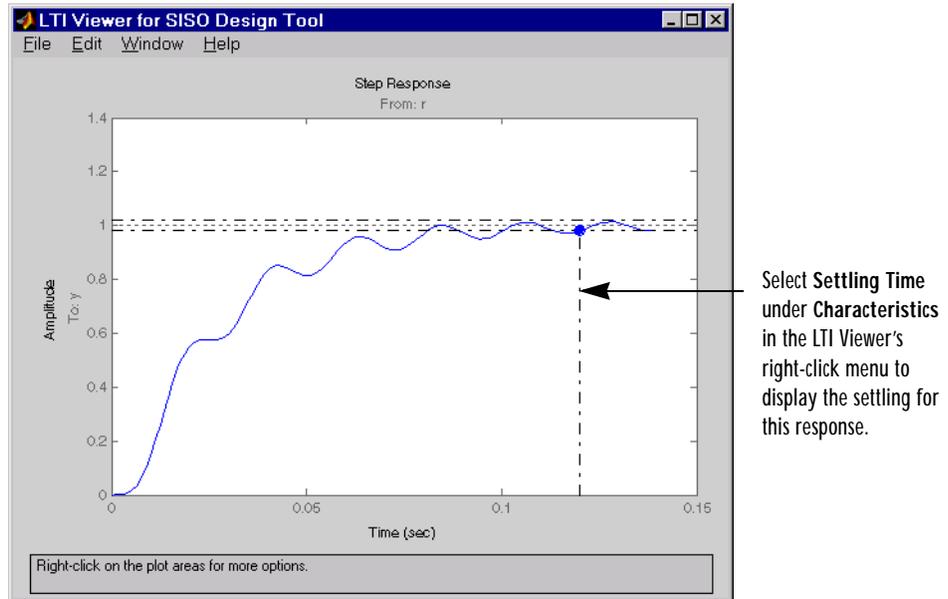


Figure 4-30: Step Response with the Settling Time for  $C(s) = 20$

This closed-loop response does not meet the desired settling time requirement (0.05 seconds or less) and exhibits unwanted ringing. The next section 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 underdamped. Further increases force the system into instability, so meeting the design requirements with only a gain in the compensator is not possible.

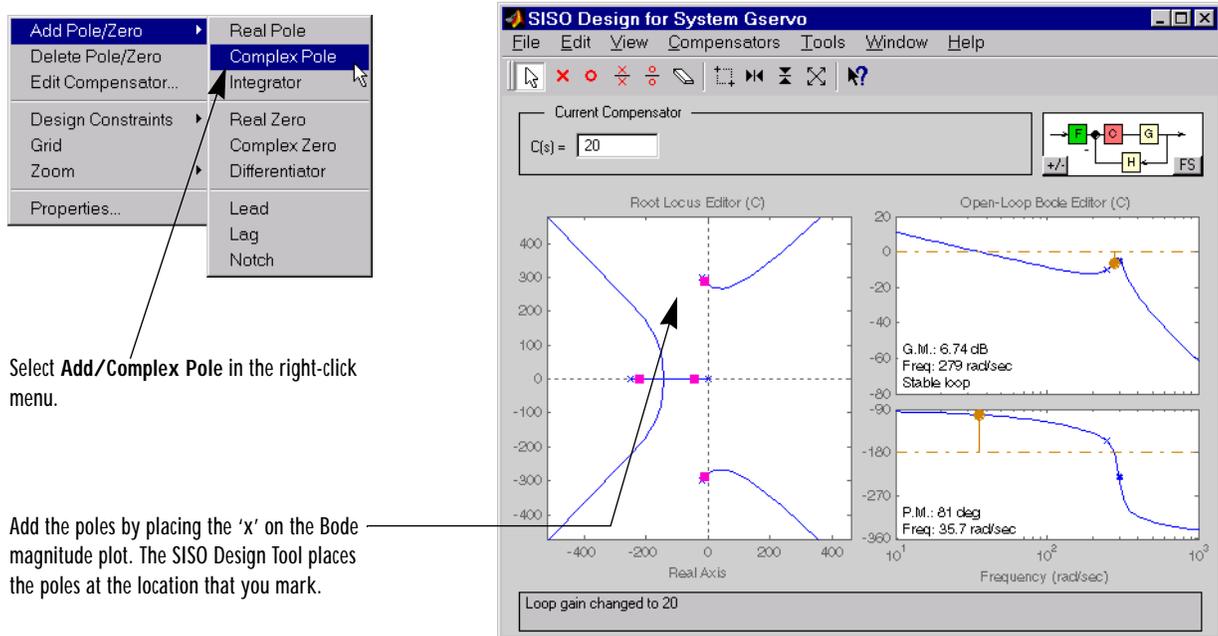
There are three sets of parameters that specify the compensator: poles, zeros, and gain. Once you have selected the gain, you can add poles or zeros to the compensator.

## Adding Poles to the Compensator

Try adding a complex conjugate compensator pole pair on the root locus plot:

- 1 Open the right-click menu and select **Add Pole/Zero** and then **Complex Pole**.
- 2 Click on the root locus plot region where you would like to add one of the complex poles.

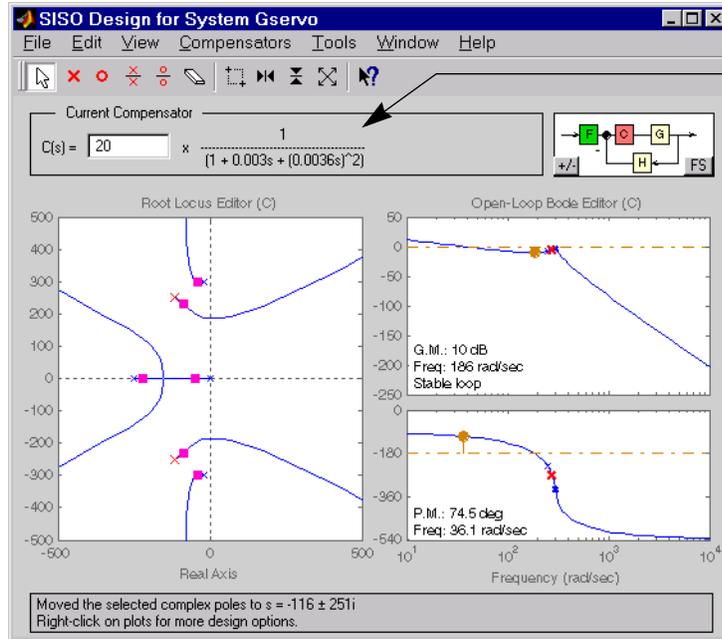
This figure shows these two steps.



**Figure 4-31: Adding a Complex Pair of Poles to the Compensator**

Try placing the 'x' somewhere to the left of the complex pole pair near the imaginary axis (the figure above shows a good spot). Once you have added the complex pair of poles, the LTI Viewer response plots change and both the root locus and Bode plots display the new poles. This figure shows the SISO Design

Tool with the new poles added. For clarity, you may want to zoom out further, as was done here.



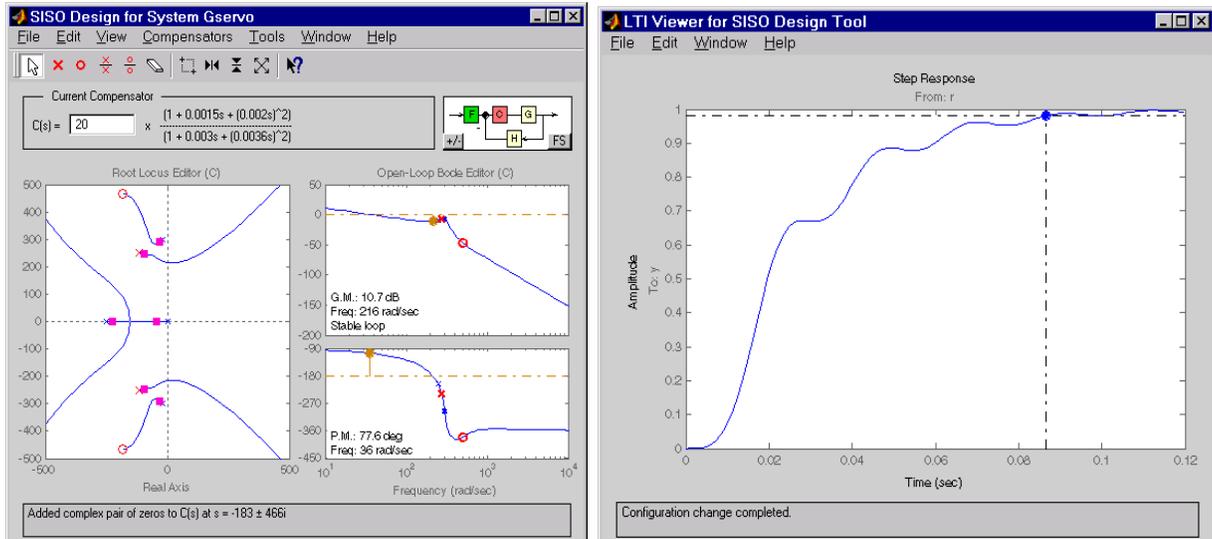
The Current Compensator displays the new complex pair of poles. If you see NumC or DenC, widen the GUI to see the full transfer function.

The SISO Design Tool displays the new poles as a pair of red x's on the root locus plot.

Figure 4-32: The 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. Try adding a pair of complex zeros just to the left of complex closed-loop poles you just added to the compensator. This figure shows the results.



**Figure 4-33: Electrohydraulic Servomechanism Example with Complex Zeros Added**

If your step response is unstable, lower the gain by grabbing a red box in the right-hand plane and moving it into the left-half 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

The next section shows you how to place poles and zeros by specifying their numerical values. It also presents a solution that meets the design specifications for the servomechanism example.

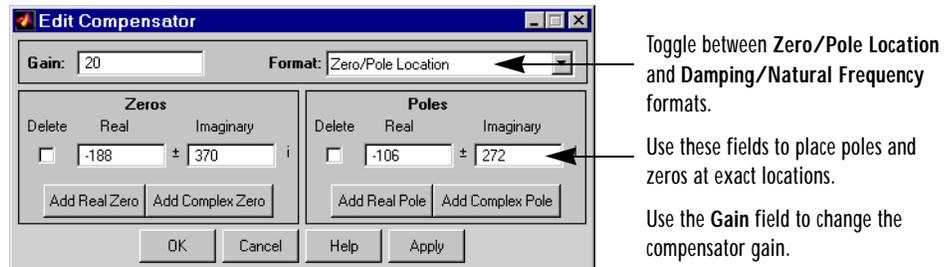
## Editing Compensator Pole and Zero Locations

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 **Edit Compensator** window to change the gain value and the pole and zero locations of your compensator.

There are three ways to open the **Edit Compensator** window from the SISO Design Tool:

- Select **Edit Compensator** under the **Edit** menu on the menu bar.
- Select **Edit Compensator** in the right-click menu. This option is available in both the root locus and Bode plot right-click menus.
- Double-click your mouse in the **Current Compensator** panel.

Whichever method you choose, the following window appears.



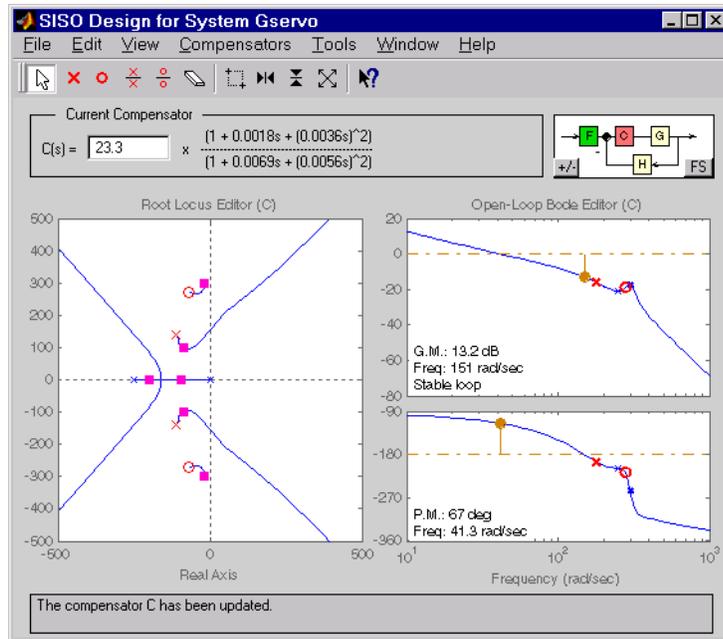
**Figure 4-34:** Use the Edit Compensator Window to Add, Delete and Move Compensator Poles and Zeros

You can use the **Edit Compensator** window to:

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

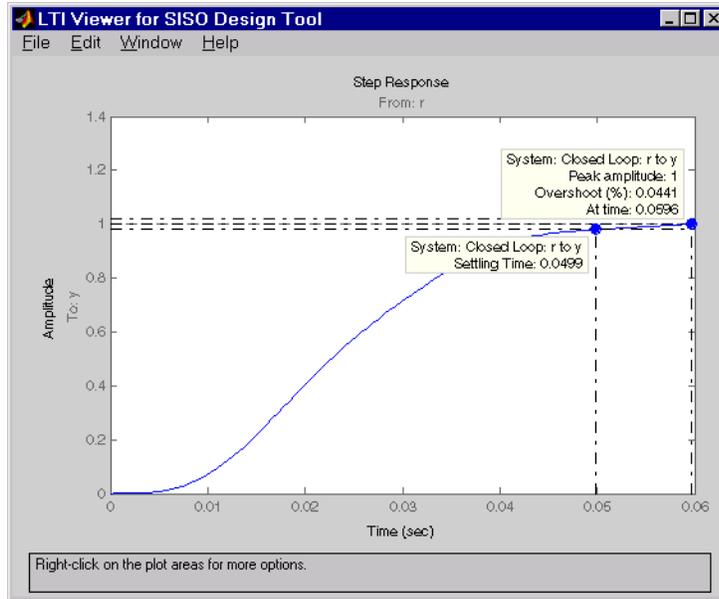
For this example, edit the poles to be at  $-110 \pm 140i$  and the zeros at  $-70 \pm 270i$ . Set the compensator gain to 23.3.

Your SISO Design Tool now looks like this.



**Figure 4-35: SISO Design Tool with the Final Values for the Electrohydraulic Servomechanism Design Example**

To see that this design meets the design requirements, take a look at the step response of the closed-loop system.



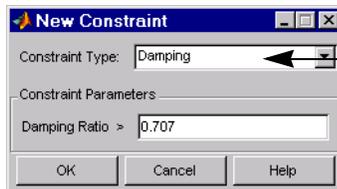
**Figure 4-36: Closed-Loop Step Response for the Final Design of the Electrohydraulic Servomechanism Example**

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 SISO Design Tool provides design constraints 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**

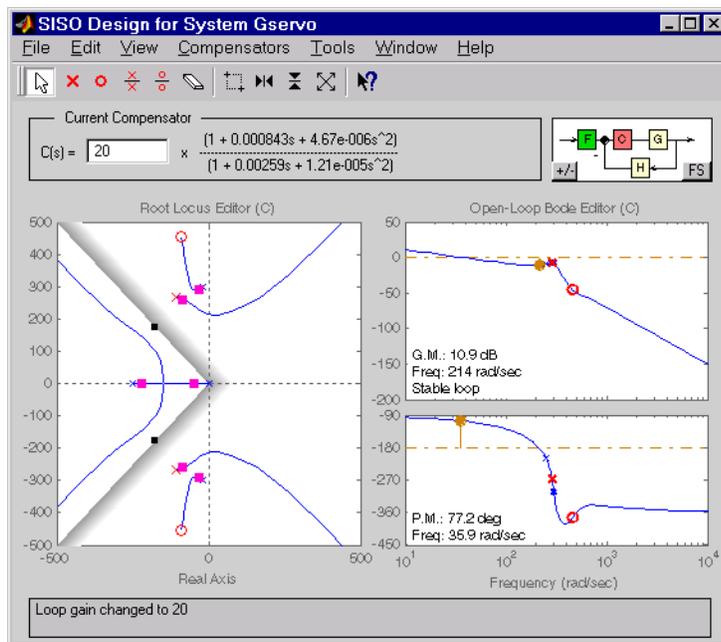
**Constraints** and then **New** from the right-click menu in the root locus. This opens the **Design Constraints** editor.



Select Damping from this pull-down menu. The default value 0.707 sets lines on the root locus corresponding to a 0.707 damping ratio.

**Figure 4-37: The New Constraint Editor**

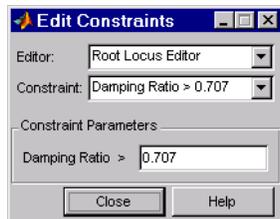
Applying damping ratios to the root locus plot results in a pair of shaded rays at the desired slope, as this figure shows.



**Figure 4-38: Root Locus Displaying 0.707 Damping Ratio Lines**

Try moving the complex pair of poles you've 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 Constraint** and then **Edit** from the right-click menu. This opens the **Edit Constraints** window.



**Figure 4-39: The Edit Constraints Window**

Specify the new damping ratio in this window.

An alternate way to adjust a constraint is to left-click on the constraint itself to select it. Two black squares appear on the constraint 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 constraints, for example, a settling time constraint, reopen the **New Constraint** window and choose settling time from the pull-down menu. You can have multiple types of design constraints in one plot, or more than one instance of any type.

The types of constraints available depend on which view you use for your design. See *Design Constraints in the Control System Toolbox* online documentation for a description of all the constraint types 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 Tool. The window shown below opens.

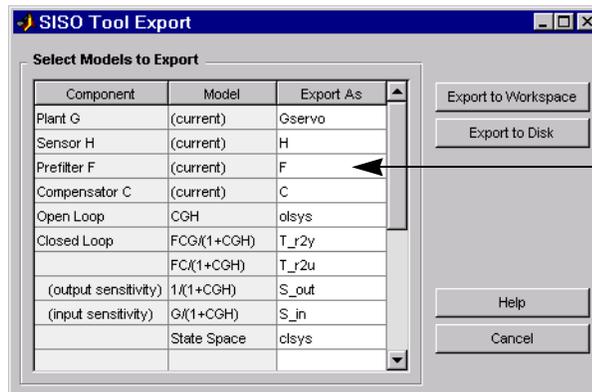


Figure 4-40: SISO Tool Export Window

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

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

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.

Selecting **Export to Disk** opens the window shown below.



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.

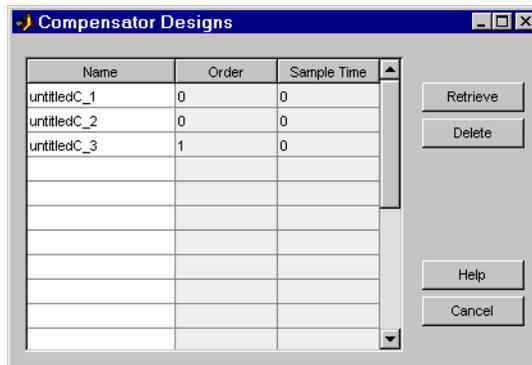
### Storing and Retrieving Intermediate Designs

You can store and retrieve intermediate compensators while you iterate on your compensator design. To store intermediate designs, select **Store** under **Compensator** in the menu bar of the SISO Design Tool. This window opens.



The default name is UntitledC\_1; the suffix increments when you store additional compensators. You can rename the designs by editing the **Store as** field.

To retrieve intermediate designs, select **Retrieve** under the **Compensator** menu. This opens the **Compensator Designs** window.



**Figure 4-41: The Compensator Designs Window**

To retrieve a design, select it from the list of names and press **Retrieve**. The SISO Design Tool automatically reverts to the selected compensator design. Note that you can rename the stored compensator by editing the name in the selected **Name** cell.

The **Compensator Designs** window also lists the order of each compensator design, and, if the compensator is digital, the sample time. The default sample time value for continuous-time compensators is 0.

You can delete an intermediate design by selecting it and pressing the **Delete** button.

## 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 4-8, 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 2-4, 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 4-31:

- 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 ltexamples
sisotool({'bode','nichols'},sys_dc)
```

This figure opens.

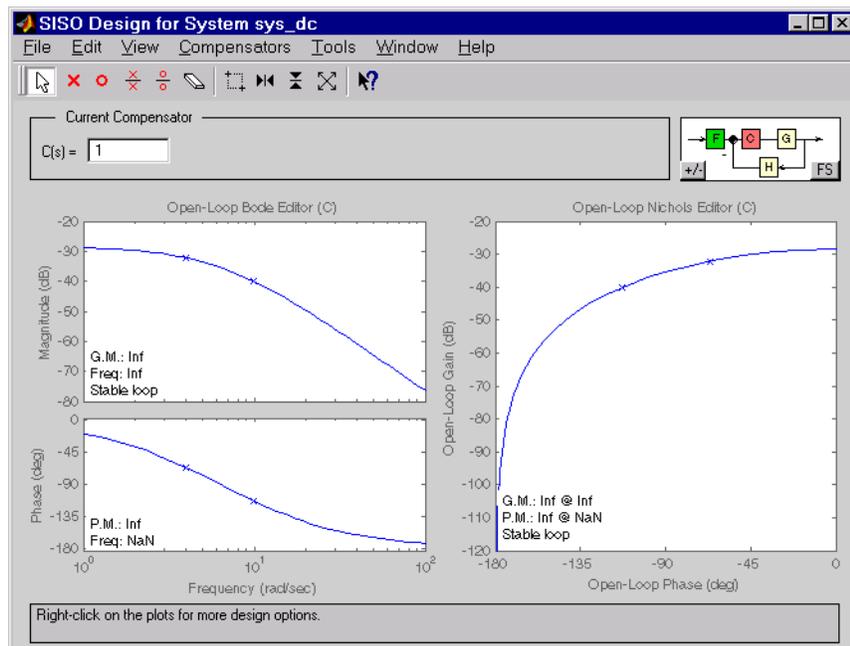


Figure 4-42: SISO Design Tool with a Nichols Plot

## Adjusting the Compensator Gain

You can adjust the compensator gain 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.

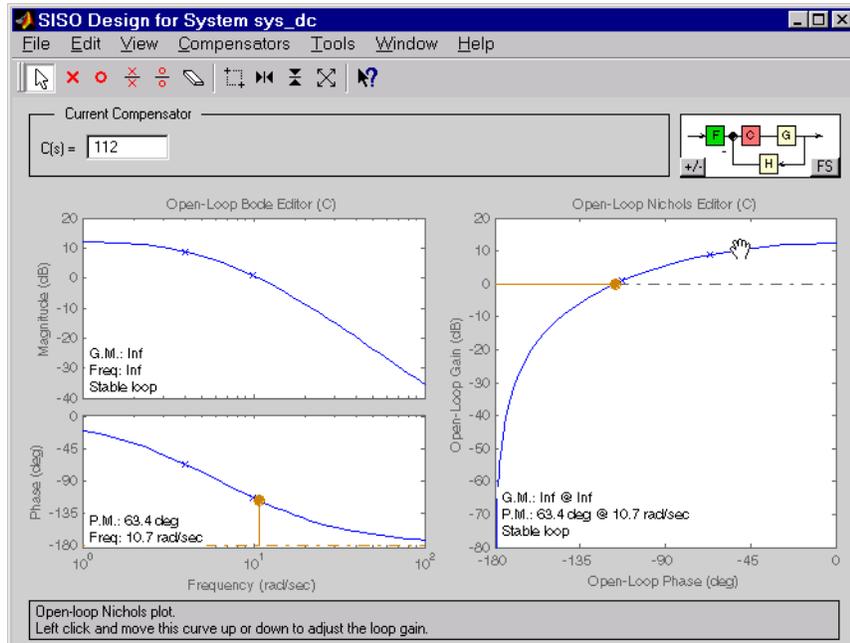
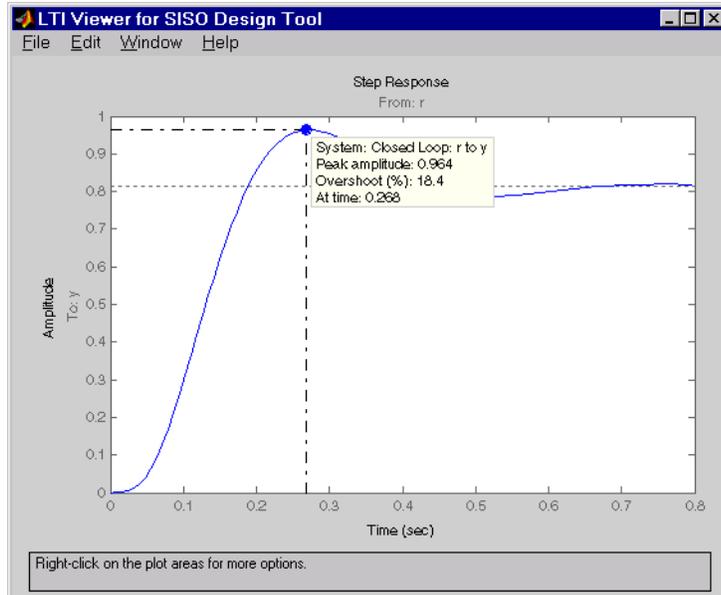


Figure 4-43: Adjusting the Compensator Gain by Moving the Nichols Curve

In this example, the new gain is 112. Select **Closed Loop Response** from the **Tools** menu to open a linked LTI Viewer.



**Figure 4-44: 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. You can do this by selecting **Add Pole/Zero** and then **Integrator** from the right-click menu.

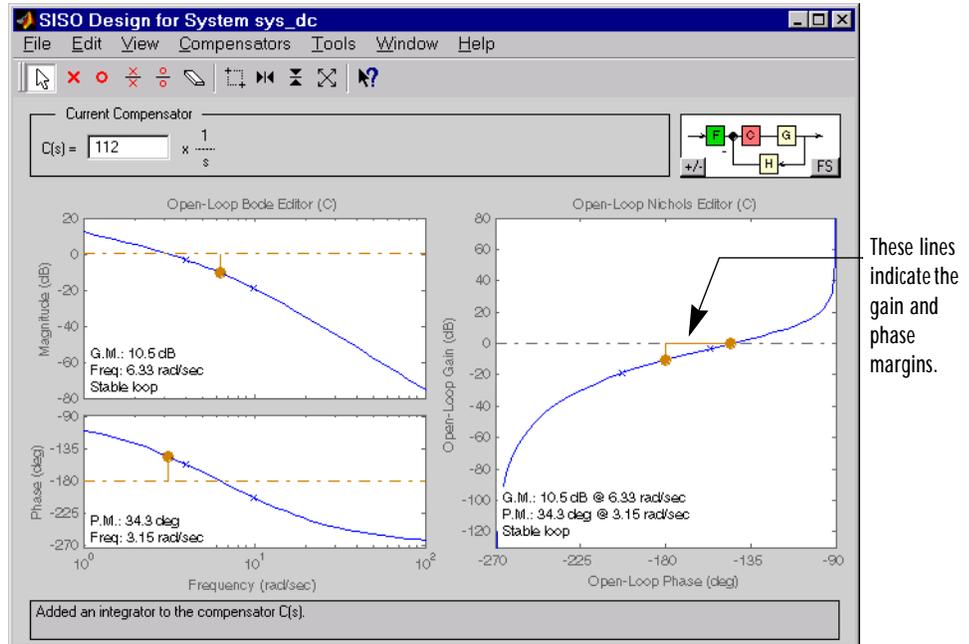
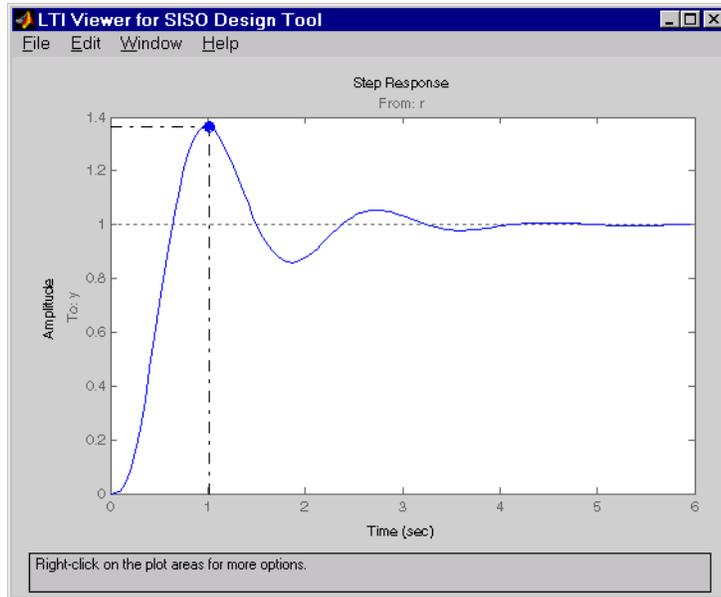


Figure 4-45: Adding an Integrator

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 displays 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.



**Figure 4-46: Step Response for a Compensator Consisting of a Gain and an Integrator**

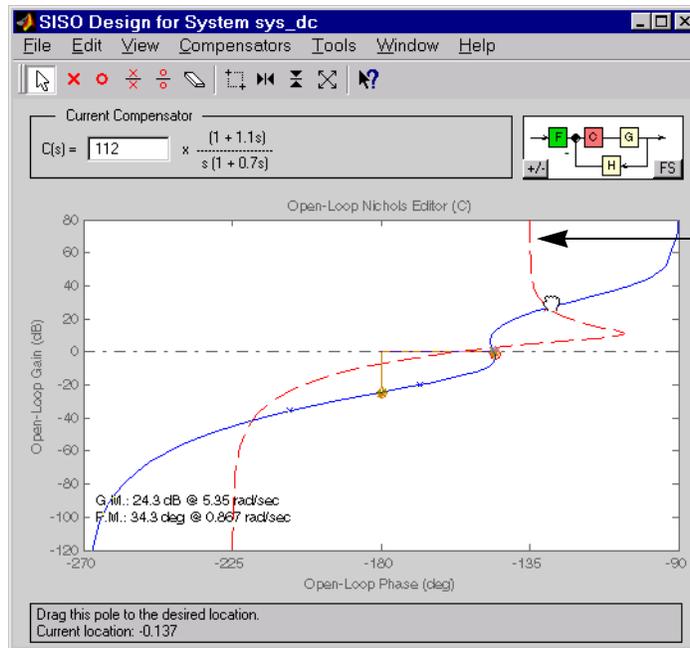
The steady-state value is now 1, which means the steady-state error has been eliminated, but the overshoot is 34% and the rise time is about 0.7 second. You must do more work to create a good design.

## 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 a lead network, select **Add Pole/Zero** and then **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 on the pole or zero and drag.

This figure shows the process of moving the lead network pole using the mouse (the open-loop Bode diagram has been removed for clarity).



This dashed line shows the location of possible pole positions.

**Figure 4-47: Moving the Lead Network's Pole**

You can track the pole's movement in the Status Panel at the bottom of the SISO Design Tool. The Status Panel tells you the current location of the pole.

For this design, move the lead network pole to -28 and the zero to -4.3. The zero should be almost on top of the rightmost pole of the plant (the DC motor model). Adjust the compensator gain to 84. This gives the final design.

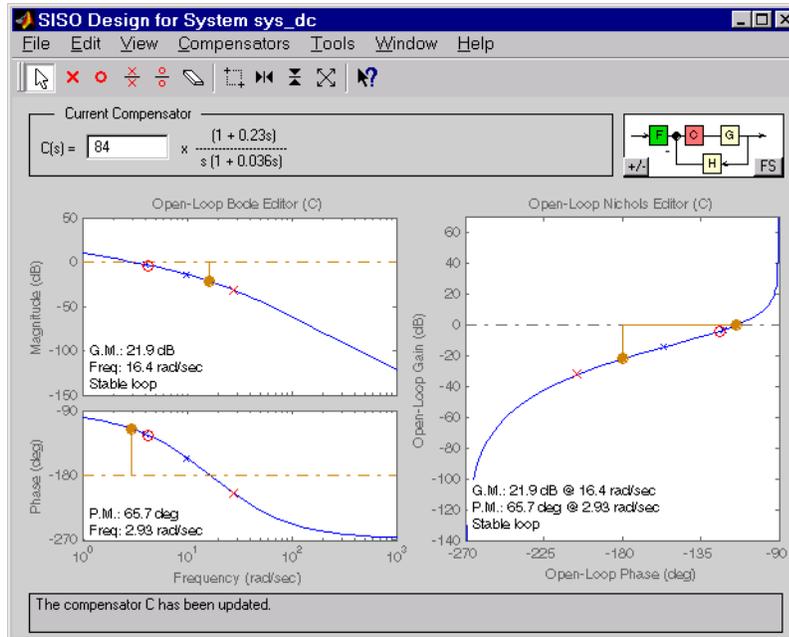
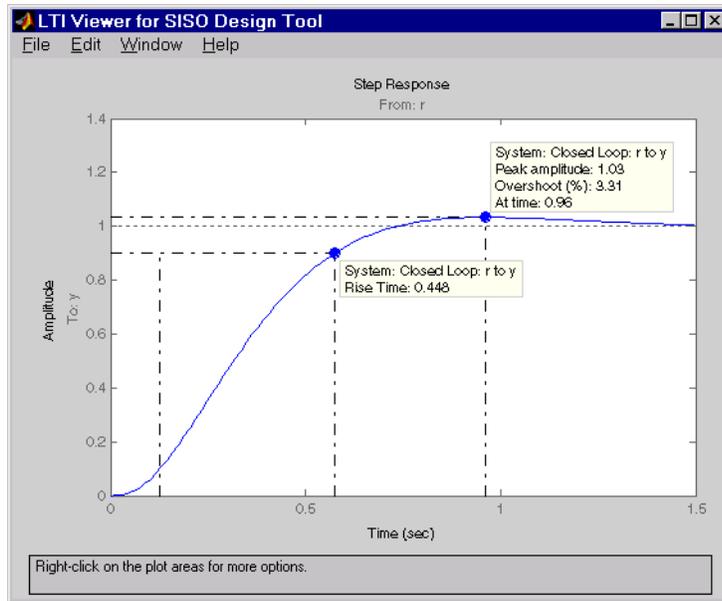


Figure 4-48: Final Nichols Plot Design for the DC Motor Compensator

The gain and phase margins are 21.9 dB and 65.7 degrees, respectively. Inspect the closed-loop step response to see if the rise time and overshoot requirements are met.



**Figure 4-49: 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.

## 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 commands for designing compensators using root locus design techniques.

Function	Description
pzmap	Pole-zero map
rlocus	Evans root locus plot
sgrid	Continuous $\omega_n$ $\zeta$ grid for root locus plots
sisotool	Root Locus Design GUI
zgrid	Discrete $\omega_n$ $\zeta$ 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

$$\begin{aligned} \dot{x} &= Ax + Bu \\ y &= Cx + Du \end{aligned}$$

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` command, 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  $\xi$  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. As a rule of thumb, the estimator dynamics should be faster than the controller dynamics (eigenvalues of  $A - BK$ ).

Use the `place` command 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  $\xi$  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 commands for pole placement.

Command	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`.

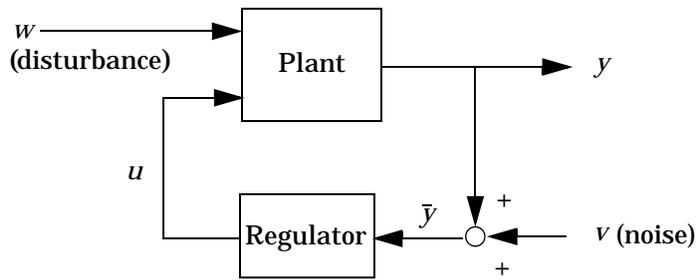
**Caution.** Pole placement can be badly conditioned if you choose unrealistic pole locations. In particular, you should avoid:

- 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 perturbations.

## Linear-Quadratic-Gaussian (LQG) Design

Linear-quadratic-Gaussian (LQG) control is a modern state-space technique for designing optimal dynamic regulators. It enables you to 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

$$J(u) = \int_0^{\infty} \{x^T Q x + 2x^T N u + u^T R u\} dt$$

The weighting matrices  $Q$ ,  $N$ , and  $R$  are user specified and define the trade-off 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` command 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

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

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

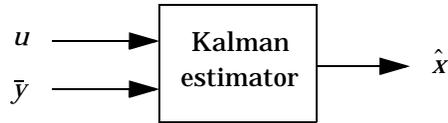
$$E(ww^T) = Q_n, \quad E(vv^T) = R_n, \quad E(wv^T) = N_n$$

determines the Kalman gain  $L$  through an algebraic Riccati equation.

The Kalman filter is an optimal estimator when dealing with Gaussian white noise. Specifically, it minimizes the asymptotic covariance

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

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



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

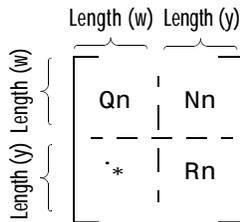
```
[kest,L,P] = kalman(sys_kf,Qn,Rn,Nn);
```

returns a state-space model `kest` of the Kalman estimator given the plant model `sys_kf` and the noise covariance data, `Qn`, `Rn`, and `Nn`. The plant model equations are the following.

$$\begin{aligned}x &= Ax + Bu + Gw \\ \bar{y} &= Cx + Du + Hw + v\end{aligned}$$

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

The figure below shows the required dimensions for `Qn`, `Rn`, and `Nn`. If `Nn` is 0, you can omit it.

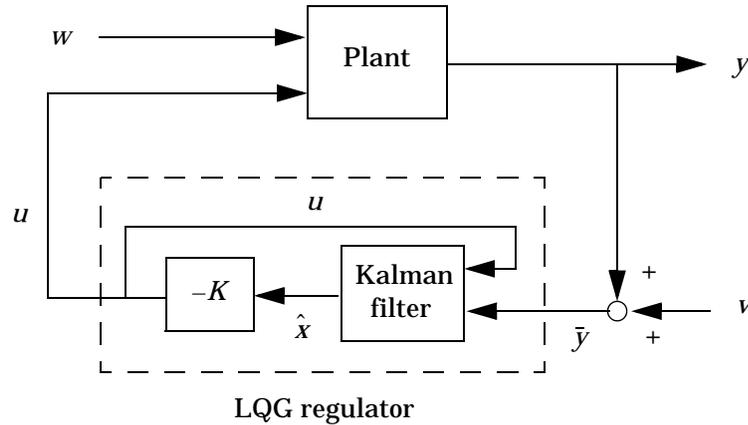


**Figure 4-50:** Required Dimensions for `Qn`, `Rn`, and `Nn`

For a complete example of a Kalman filter implementation, see [Kalman Filtering under Design Case Studies](#) online.

### LQG Regulator

To form the LQG regulator, simply connect the Kalman filter and LQ-optimal gain  $K$  as shown below.



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` command to create the LQG regulator.

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

See the example LQG Regulation under Design Case Studies online for a more complete discussion of construction LQG regulators.

### LQG Design Tools

The Control System Toolbox contains functions to perform the three LQG design steps outlined above. 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 commands.

<b>Command</b>	<b>Description</b>
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 regulation problem illustrated below.

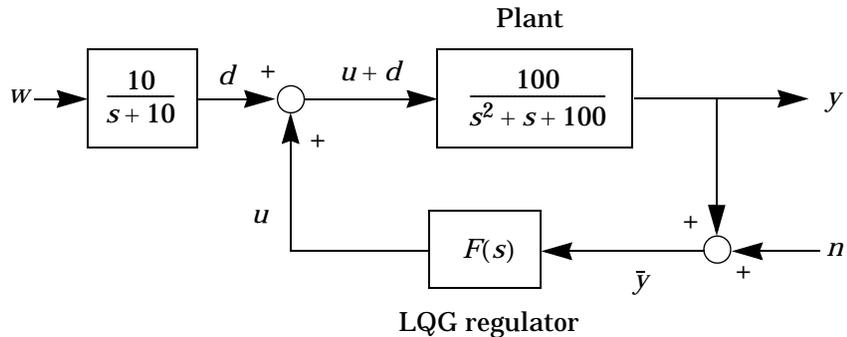
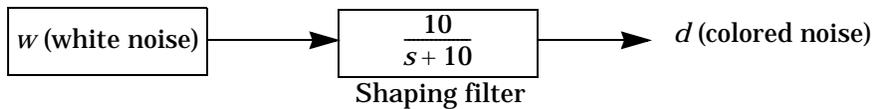
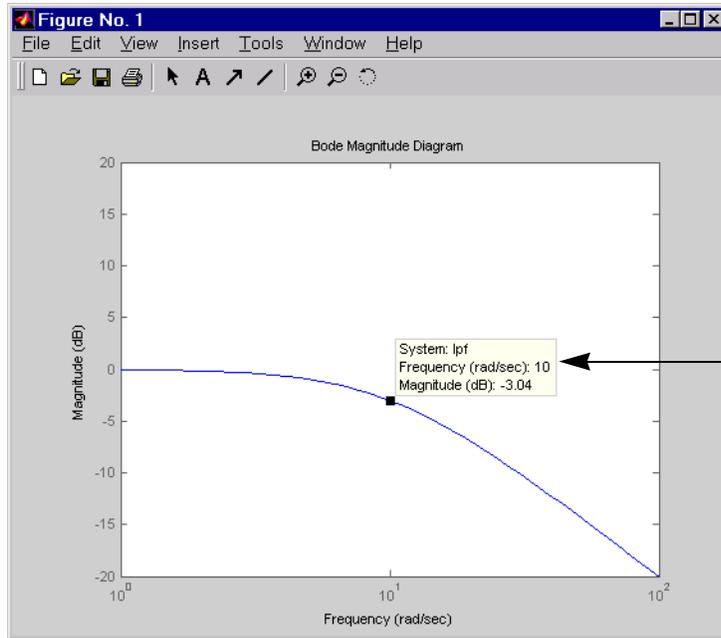


Figure 4-51: Simple Regulation Loop

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/sec. For LQG design purposes, it is modeled as white noise driving a low-pass filter with a cutoff at 10 rad/sec, as this picture shows.



This figure shows the Bode magnitude of the shaping filter.



**Figure 4-52: Bode Magnitude of the Low-Pass 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 trade-off 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.

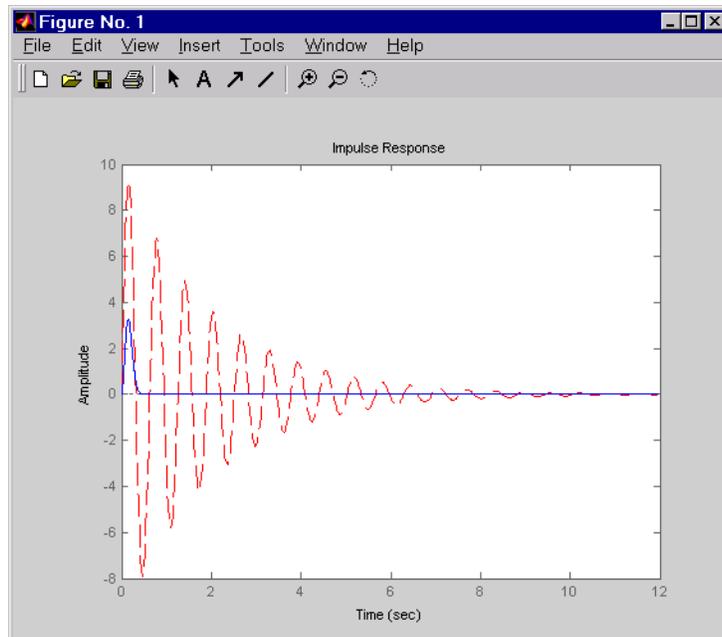
To validate the design, close the loop with feedback, create and add the lowpass filter in series with the closed-loop system, and compare the open- and closed-loop impulse responses by using the `impz` function.

```
% 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.



**Figure 4-53: 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. The previous section, “Example: LQG Design,” 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.

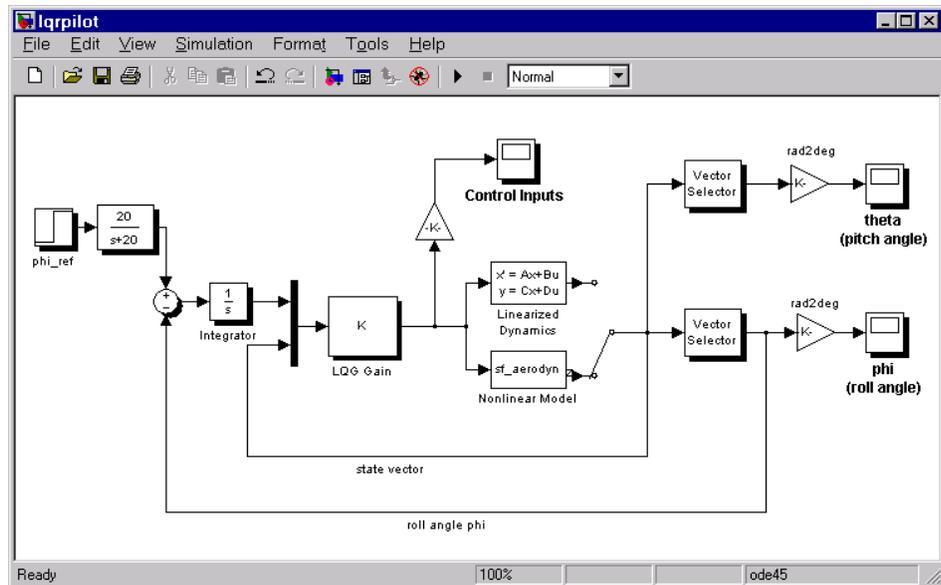


Figure 4-54: A Tracking Loop

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  $\phi$  and the  $\phi_{ref}$  is passed through an integrator. This aids in driving the error to zero.

### State-Space Equations for an Airframe

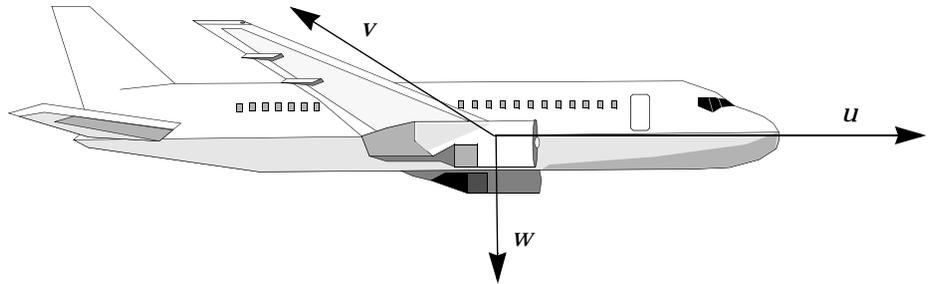
Beginning with the standard state-space equation

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

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, which is shown in the figure below.



**Figure 4-55: A Body Coordinate Frame for an Aircraft**

The variables  $\theta$  and  $\phi$  are roll and pitch, and  $p$ ,  $q$ , and  $r$  are the roll, pitch, and yaw rates, respectively.

The airframe dynamics are nonlinear. The equation below shows the nonlinear components added to the state space equation.

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

**Figure 4-56: Nonlinear Component of the State-Space Equation**

To see the numerical values for  $A$  and  $B$ , type

```
load lqrpilot
A, B
```

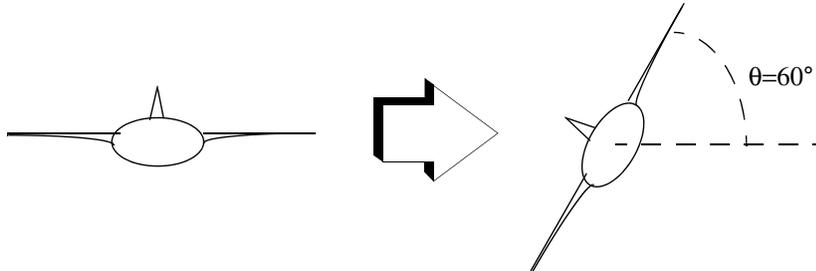
at the MATLAB prompt.

### Trimming

For LQG design purposes, the nonlinear dynamics are trimmed at  $\phi = 15^\circ$  and  $p$ ,  $q$ ,  $r$ , and  $\theta$  set to zero. Since  $u$ ,  $v$ , and  $w$  do not enter into the nonlinear term in Figure 4-56, 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  $A_{15}$ .

**Problem Definition**

The goal to perform a steady coordinated turn, as shown in this figure.



**Figure 4-57: An Aircraft Making a  $60^\circ$  Turn**

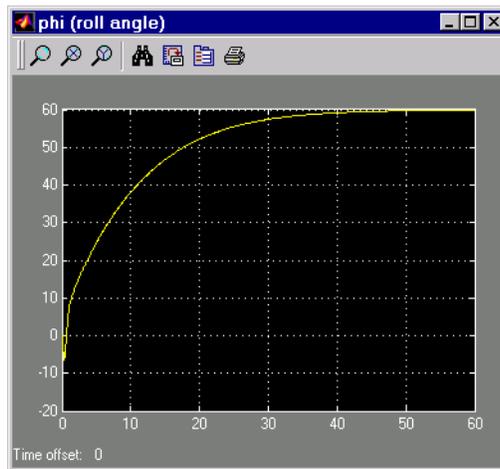
To achieve this goal, you must design a controller that commands a steady turn by going through a  $60^\circ$  roll. In addition, assume that  $\theta$ , the pitch angle, is required to stay as close to zero as possible.

### 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  $\phi$  to the  $60^\circ$  step command.



**Figure 4-58: Tracking the Roll Step Command**

As you can see, the system tracks the commanded  $60^\circ$  roll in about 60 seconds.

Another goal was to keep  $\theta$ , the pitch angle, relatively small. This figure shows how well the LQG controller did.

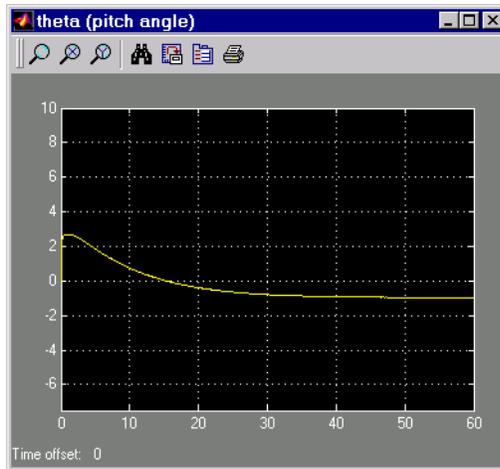


Figure 4-59: Minimizing the Displacement in the Pitch Angle, Theta

Finally, this figure shows the control inputs.

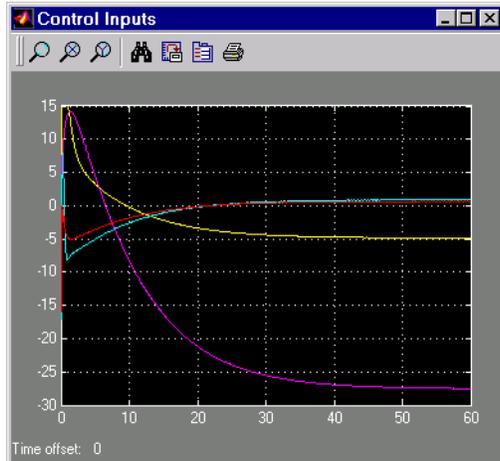


Figure 4-60: The Control Inputs for the LQG Tracking Problem

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

---

Demos . . . . .	5-2
Online Help . . . . .	5-2
Setting Plot Preferences and Properties . . . . .	5-3
The MathWorks Online . . . . .	5-4

## Demos

To see more Control System Toolbox examples, type

```
demo
```

at the MATLAB prompt. This opens the **MATLAB Demo** window. Select **Control System Toolbox** under **Toolboxes** to list the available demos.

Alternatively, you can access the demos from MATLAB's Launch Pad. Select "Demos" under "Control System Toolbox" to see a list of demos.

## 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.

Aside from this book, *Getting Started with the Control System Toolbox*, the online documentation contains the following topics:

- "Release Notes" — For 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" — 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 (GUI's) that you can use to analyze systems and design SISO compensators
- "Function Reference" — A complete guide to all the functions in the Control System Toolbox. Functions are listed by category and alphabetically.

## Setting Plot Preferences and Properties

The Control System Toolbox provides three graphical user interfaces (GUI's), 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 GUI's; see the online help system for a 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.

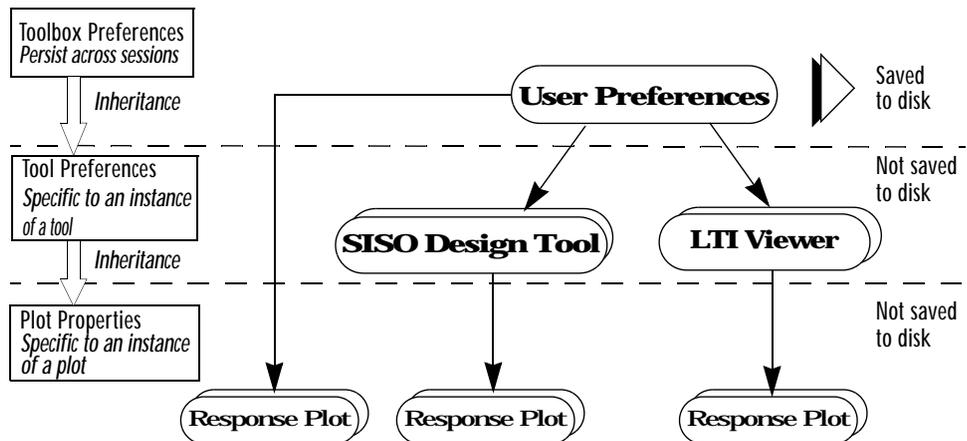


Figure 5-1: Preference and Property Inheritance Hierarchy

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 on 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** in the online help under Control System Toolbox.

## The MathWorks Online

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

`http://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 entitled *MATLAB Based Books* is available from The MathWorks and an up-to-date list is available on the Web site.

## B

- building models
  - MIMO 2-13
  - SISO 2-6

## C

- cell array 2-16
- changing
  - units on a plot 4-19
- comparing
  - models in plots 3-23
- compensators
  - editing in SISO Design Tool 4-40
  - exporting 4-45
  - feedback (state estimator design) 4-59
- concatenation, model 2-15
- covariance
  - noise 4-62

## D

- data markers 3-22
- denominator
  - specification 2-16
- design
  - Kalman estimator 4-62
  - LQG Tracker, example of 4-70
  - LQG, description of 4-61
  - LQG, example of 4-66
  - LQG, syntax 4-64
  - pole placement 4-57
  - regulators 4-64
  - root locus 4-57
  - state estimator 4-58
  - state estimator, Kalman 4-62

## E

- estimator
  - gain 4-58
  - Kalman 4-62
- exporting compensators and models 4-45

## F

- final time. *See* time response
- frequency response 3-19
  - customizing plots 3-26
  - plotting 3-23
- functions
  - time and frequency response 3-17

## G

- gain
  - estimator, Kalman 4-62
  - selection 4-27
  - state-feedback 4-62

## H

- Help Navigator 5-2

## I

- input point block 3-34
  - See also* Simulink LTI Viewer
- input vector 2-3
- installation 1-3

## K

- Kalman estimator
  - continuous 4-62

- gain 4-62
- steady-state 4-62
- Kalman filter. *See* Kalman estimator

**L**

- linear models
  - comparing multiple models in plots 3-23
  - exporting 4-45
  - frequency response. *See* frequency response
  - model order reduction 2-26
  - time response. *See* time response
- LQG (linear quadratic Gaussian) method
  - weighting matrices 4-62
- LQG (linear quadratic-gaussian) method
  - continuous LQ regulator 4-62
  - design *See* design, LQG
  - gain, optimal state-feedback 4-62
  - Kalman state estimator 4-62
  - LQ-optimal gain 4-62
  - regulator, designing 4-68
  - regulator, forming 4-64
- LTI Viewer 3-3
  - 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
  - models, importing 3-4
  - opening 3-4
  - plot options
    - rise time 3-7
    - settling time 3-14
  - right-click menus 3-5
  - Simulink models. *See* Simulink LTI Viewer

**M**

- markers, data 3-22
- MathWorks Web site 5-4
- MIMO 3-19
- Model Input and Output blocks 3-31
- model order reduction 2-26

**N**

- noise
  - covariance 4-62
  - white 3-17
- numerator
  - specification 2-16

**O**

- online help 5-2
- operations on LTI models
  - concatenation 2-15
- output point block 3-34
  - See also* Simulink LTI Viewer
- output vector 2-3
- overshoot 3-17

**P**

- plotting
  - changing units 4-19
  - customizing plots 3-26
  - frequency response. *See* frequency response
  - multiple systems 3-23
  - right-click menus 3-20
  - time responses 3-17
- pole placement 4-57
  - conditioning 4-60

**R**

- reduced-order models 2-26
- regulation 4-61, 4-67
  - performance 4-62
- related products 1-6
- Riccati equation 4-62
- right-click menus
  - LTI Viewer 3-5
  - plot responses 3-20
- rise time 3-17
- root locus
  - compensator gain 4-35
  - designing 4-57
- root locus design 4-27
  - functions for 4-57

**S**

- settling time 3-17
- Simulink LTI Viewer 3-29, 3-31
  - analysis models
    - exporting 3-41
    - specifying 3-34
  - opening 3-31
  - operating conditions, setting 3-42
  - specifying models for 3-31
- SISO Design Tool 4-3
  - adding a lead network 4-14
  - adding a notch filter 4-20
  - adding a prefilter 4-24
  - adding an integrator 4-12
  - adding poles and zeros 4-36
  - changing the compensator gain, Bode magnitude 4-8
  - changing the compensator gain, root locus 4-35
  - changing the feedback structure 4-6

- compensators, editing 4-40
- importing models 4-5
- opening 4-3
- root locus design 4-27

**state**

- estimator 4-58
- vector 2-3
- state estimator design
  - regulators/compensators 4-59
- state-space models 2-3
- steady state error 3-17

**T**

- time response 3-17
  - customizing plots 3-26
  - final time 3-19
  - MIMO 3-19
  - plotting 3-23
  - time range 3-19
  - to white noise 3-17
  - vector of time samples 3-19
- transfer functions
  - MIMO 2-15
- transfer functions (TF) 2-3
- typographical conventions (table) 1-8

**U**

- units, changing on a plot 4-19

**W**

- white noise 3-17
  - See also* noise

**Z**

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