

Control System Toolbox™ 9

User's Guide

MATLAB®

How to Contact MathWorks



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



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



508-647-7000 (Phone)



508-647-7001 (Fax)



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

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

Control System Toolbox™ User's Guide

© COPYRIGHT 2001–2011 by The MathWorks, Inc.

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

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

Trademarks

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

Patents

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

Revision History

| | | |
|----------------|-------------|---|
| June 2001 | Online only | New for Version 5.1 (Release 12.1) |
| July 2002 | Online only | Revised for Version 5.2 (Release 13) |
| June 2004 | Online only | Revised for Version 6.0 (Release 14) |
| March 2005 | Online only | Revised for Version 6.2 (Release 14SP2) |
| September 2005 | Online only | Revised for Version 6.2.1 (Release 14SP3) |
| March 2006 | Online only | Revised for Version 7.0 (Release 2006a) |
| September 2006 | Online only | Revised for Version 7.1 (Release 2006b) |
| March 2007 | Online only | Revised for Version 8.0 (Release 2007a) |
| September 2007 | Online only | Revised for Version 8.0.1 (Release 2007b) |
| March 2008 | Online only | Revised for Version 8.1 (Release 2008a) |
| October 2008 | Online only | Revised for Version 8.2 (Release 2008b) |
| March 2009 | Online only | Revised for Version 8.3 (Release 2009a) |
| September 2009 | Online only | Revised for Version 8.4 (Release 2009b) |
| March 2010 | Online only | Revised for Version 8.5 (Release 2010a) |
| September 2010 | Online only | Revised for Version 9.0 (Release 2010b) |
| April 2011 | Online only | Revised for Version 9.1 (Release 2011a) |

Linear System Modeling

Linear System Model Objects

1

| | |
|--|------|
| About Model Objects | 1-2 |
| What Are Model Objects? | 1-2 |
| Control System Modeling with Model Objects | 1-3 |
| | |
| Model Object Families | 1-6 |
| Dynamic vs. Static Models | 1-6 |
| Numeric Linear Time Invariant (LTI) Models | 1-7 |
| Control Design Blocks | 1-7 |
| Generalized Matrices | 1-8 |
| Generalized LTI Models | 1-9 |
| | |
| Models with Fixed Coefficients | 1-10 |
| Numeric LTI Models | 1-10 |
| Transfer Functions | 1-11 |
| State-Space Models | 1-12 |
| Frequency Response Data (FRD) Models | 1-14 |
| PID Controllers | 1-16 |
| | |
| Models with Tunable Coefficients | 1-19 |
| Generalized LTI Models | 1-19 |
| Modeling Tunable Components | 1-19 |
| Modeling Control Systems with Tunable Components | 1-20 |
| Internal Structure of Generalized Models | 1-21 |
| | |
| Using Linear System Model Objects | 1-23 |
| | |
| Converting Between Model Types | 1-24 |
| Explicit Conversion | 1-24 |
| Automatic Conversion | 1-24 |
| Caution About Model Conversions | 1-25 |

| | |
|---|-------------|
| Simulink Block for LTI Systems | 1-26 |
| References | 1-28 |

Model Creation

2

| | |
|--|-------------|
| Single-Input Single-Output (SISO) Model Creation ... | 2-2 |
| Transfer Function Model Using Numerator and Denominator Coefficients | 2-2 |
| Transfer Function Model Using Zeros, Poles, and Gain ... | 2-3 |
| State-Space Model | 2-4 |
| Frequency-Response Model | 2-5 |
| Proportional-Integral-Derivative (PID) Controller | 2-6 |
| | |
| Discrete-Time Model Creation | 2-10 |
| Discrete-Time Transfer Function Model | 2-10 |
| Discrete-Time Proportional-Integral-Derivative (PID) Controller | 2-11 |
| | |
| Multi-Input Multi-Output (MIMO) Model Creation | 2-14 |
| Multi-Input Multi-Output Transfer Function Model | 2-14 |
| Multi-Input Multi-Output State-Space Model | 2-15 |
| Multi-Input Multi-Output Descriptor State-Space Model | 2-17 |
| Multi-Input Multi-Output Frequency Response Data Model | 2-18 |
| Select Input/Output Pairs in Multi-Input Multi-Output (MIMO) Models | 2-20 |
| | |
| Models with Time Delays | 2-21 |
| About Modeling Time Delays | 2-21 |
| First Order Plus Dead Time Model | 2-21 |
| Input and Output Delay in State-Space Model | 2-23 |
| Transport Delay in Multi-Input, Multi-Output (MIMO) Transfer Function | 2-25 |
| Closing Feedback Loops with Time Delays | 2-26 |
| Discrete-Time Transfer Function with Time Delay | 2-29 |
| Time-Delay Approximation | 2-29 |

| | |
|---|------|
| Frequency Response Data (FRD) Model with Time Delay | 2-44 |
| About Internal Delays | 2-46 |
| Models with Parametric or Tunable Coefficients | 2-52 |
| Tunable Low-Pass Filter | 2-52 |
| Tunable Second-Order Filter | 2-53 |
| State-Space Model With Both Fixed and Tunable Parameters | 2-54 |
| Control System With Tunable Components | 2-55 |
| Model Arrays | 2-58 |
| About Model Arrays | 2-58 |
| One-Dimensional Model Array with Single Parameter Variation | 2-61 |
| Select Models from Array | 2-61 |
| Array With Variations in Two Parameters | 2-64 |
| Sample a Tunable (Parametric) Model for Parameter Studies | 2-66 |
| References | 2-69 |

Working with Linear Models

Data Manipulation

3

| | |
|---|-----|
| Model Properties | 3-2 |
| About Model Properties | 3-2 |
| Specify Model Properties at Model Creation | 3-2 |
| Examine and Specify Properties of an Existing Model | 3-3 |
| Extract Model Coefficients | 3-5 |
| Functions for Extracting Model Coefficients | 3-5 |
| Extracting Coefficients of Different Model Type | 3-5 |
| Extract Numeric Model Data and Time Delay | 3-6 |
| Extract Proportional-Integral-Derivative (PID) Gains from Transfer Function | 3-7 |

| | |
|---|-------------|
| Attach Metadata to Models | 3-9 |
| Specify Model Time Units | 3-9 |
| Interconnect Models with Different Time Units | 3-10 |
| Specify Frequency Units of Frequency-Response Data Model | 3-10 |
| Extract Subsystems of Multi-Input, Multi-Output (MIMO) Models | 3-11 |
| Specify and Select Input and Output Groups | 3-12 |
| | |
| Query Model Characteristics | 3-14 |
| Query Model Dynamics | 3-14 |
| Query Array Size | 3-15 |
| | |
| Customize Model Display | 3-17 |
| Configure Transfer Function Display Variable | 3-17 |
| Configure Display Format of Transfer Function in Factorized Form | 3-18 |

Model Interconnections

4

| | |
|--|-------------|
| About Model Interconnections | 4-2 |
| | |
| Catalog of Model Interconnections | 4-3 |
| | |
| Build a Model of a Single-Input, Single-Output (SISO) Feedback Loop | 4-5 |
| | |
| Build a Model of a Multi-Input, Multi-Output (MIMO) Feedback Loop | 4-8 |
| | |
| Build a Model of a Multi-Loop Control System | 4-9 |
| | |
| Build a Model of a Multi-Input, Multi-Output (MIMO) Control System | 4-12 |
| | |
| Results of Connecting Models | 4-16 |

| | |
|---|-------------|
| Property Inheritance in Connected Models | 4-16 |
| Result When Connecting Different Model Types | 4-16 |
| Recommended Model Type for Building Block Diagrams | 4-18 |

Operations on Models

5

| | |
|--|-------------|
| Overview | 5-2 |
| Precedence and Property Inheritance | 5-3 |
| Viewing LTI Systems as Matrices | 5-5 |
| Data Retrieval | 5-6 |
| Extracting and Modifying Subsystems | 5-8 |
| What is a Subsystem? | 5-8 |
| Basic Subsystem Concepts | 5-8 |
| Referencing FRD Models Through Frequencies | 5-11 |
| Referencing Channels by Name | 5-12 |
| Resizing LTI Systems | 5-13 |
| Arithmetic Operations on LTI Models | 5-15 |
| Supported Arithmetic Operations | 5-15 |
| Addition and Subtraction | 5-15 |
| Multiplication | 5-17 |
| Inversion and Related Operations | 5-18 |
| Transposition | 5-18 |
| Pertransposition | 5-19 |
| Model Interconnection Functions | 5-20 |
| Supported Interconnection Functions | 5-20 |
| Concatenation of LTI Models | 5-21 |
| Feedback and Other Interconnection Functions | 5-22 |

| | |
|---|-------------|
| Converting Between Continuous- and Discrete-Time Representations | 5-24 |
| Supported Conversion Functions and Methods | 5-24 |
| Zero-Order Hold Conversion Method | 5-25 |
| First-Order Hold Conversion Method | 5-27 |
| Impulse-Invariant Mapping | 5-28 |
| Tustin Approximation | 5-32 |
| Zero-Pole Matching Equivalents | 5-35 |
| | |
| Resampling of Discrete-Time Models | 5-37 |
| Available Commands for Resampling Discrete-Time Models | 5-37 |
| Example of Resampling a Discrete-Time Model | 5-37 |
| | |
| References | 5-41 |

Model Analysis Tools

6

| | |
|---|-------------|
| General Model Characteristics | 6-2 |
| | |
| Model Dynamics | 6-4 |
| | |
| State-Space Realizations | 6-7 |
| | |
| Analyzing Systems With Time Delays | 6-8 |
| Considerations to Keep in Mind when Analyzing Systems with Internal Time Delays | 6-11 |
| | |
| Sensitivity Analysis | 6-16 |
| | |
| Discretization | 6-19 |

Customization

Preliminaries

7

| | |
|--|-----|
| Terminology | 7-2 |
| Property and Preferences Hierarchy | 7-3 |
| Ways to Customize Plots | 7-5 |

Setting Toolbox Preferences

8

| | |
|--|-----|
| Toolbox Preferences Editor | 8-2 |
| Overview of the Toolbox Preferences Editor | 8-2 |
| Opening the Toolbox Preferences Editor | 8-2 |
| Units Pane | 8-4 |
| Style Pane | 8-7 |
| Options Pane | 8-8 |
| SISO Tool Pane | 8-9 |

Setting Tool Preferences

9

| | |
|-------------------------------------|-----|
| Introduction | 9-2 |
| LTI Viewer Preferences Editor | 9-3 |

| | |
|---|------------|
| Opening the LTI Viewer Preference Editor | 9-3 |
| Units Pane | 9-4 |
| Style Pane | 9-6 |
| Options Pane | 9-7 |
| Parameters Pane | 9-8 |
| Graphical Tuning Window Preferences Editor | 9-9 |
| Opening the Graphical Tuning Window Preferences Editor | 9-9 |
| Units Pane | 9-10 |
| Time Delays Pane | 9-11 |
| Style Pane | 9-12 |
| Options Pane | 9-15 |
| Line Colors Pane | 9-16 |

Customizing Response Plot Properties

10

| | |
|--|--------------|
| Introduction | 10-2 |
| Customizing Response Plots Using the Response Plots Property Editor | 10-3 |
| Opening the Property Editor | 10-3 |
| Overview of Response Plots Property Editor | 10-4 |
| Labels Pane | 10-6 |
| Limits Pane | 10-6 |
| Units Pane | 10-7 |
| Style Pane | 10-16 |
| Options Pane | 10-17 |
| Editing Subplots Using the Property Editor | 10-18 |
| Customizing Response Plots Using Plot Tools | 10-19 |
| Properties You Can Customize Using Plot Tools | 10-19 |
| Opening and Working with Plot Tools | 10-20 |
| Example of Changing Line Color Using Plot Tools | 10-20 |
| Customizing Response Plots from the Command Line | 10-23 |
| Overview of Customizing Plots from the Command Line .. | 10-23 |

| | |
|--|-------|
| Obtaining Plot Handles | 10-26 |
| Obtaining Plot Options Handles | 10-27 |
| Examples of Customizing Plots from the Command Line .. | 10-30 |
| Properties and Values Reference | 10-33 |
| Property Organization Reference | 10-47 |

| | |
|--|--------------|
| Customizing Plots Inside the SISO Design Tool | 10-48 |
| Overview of Customizing SISO Design Tool Plots | 10-48 |
| Root Locus Property Editor | 10-48 |
| Open-Loop Bode Property Editor | 10-52 |
| Open-Loop Nichols Property Editor | 10-55 |
| Prefilter Bode Property Editor | 10-57 |

Design Case Studies

11

| | |
|---|--------------|
| Yaw Damper for a 747 Jet Transport | 11-2 |
| Overview of this Case Study | 11-2 |
| Creating the Jet Model | 11-2 |
| Computing Open-Loop Eigenvalues | 11-4 |
| Open-Loop Analysis | 11-5 |
| Root Locus Design | 11-8 |
| Washout Filter Design | 11-13 |
| | |
| Hard-Disk Read/Write Head Controller | 11-19 |
| Overview of this Case Study | 11-19 |
| Creating the Read/Write Head Model | 11-19 |
| Model Discretization | 11-20 |
| Adding a Compensator Gain | 11-22 |
| Adding a Lead Network | 11-23 |
| Design Analysis | 11-26 |
| | |
| LQG Regulation: Rolling Mill Example | 11-30 |
| Overview of this Case Study | 11-30 |
| Process and Disturbance Models | 11-30 |
| LQG Design for the x-Axis | 11-33 |
| LQG Design for the y-Axis | 11-40 |
| Cross-Coupling Between Axes | 11-42 |
| MIMO LQG Design | 11-45 |

| | |
|-----------------------------------|--------------|
| Kalman Filtering | 11-49 |
| Overview of this Case Study | 11-49 |
| Discrete Kalman Filter | 11-50 |
| Steady-State Design | 11-51 |
| Time-Varying Kalman Filter | 11-57 |
| Time-Varying Design | 11-58 |
| References | 11-62 |

Reliable Computations

12

| | |
|--|-----------------|
| Scaling State-Space Models | 12-2 |
| Why Scaling Is Important | 12-2 |
| When to Scale Your Model | 12-2 |
| Manually Scaling Your Model | 12-3 |
| How To Get Accurate Results | 12-8 |

Using the SISO Design Tool and the LTI Viewer

SISO Design Tool

13

| | |
|--|------------------|
| Overview of the SISO Design Tool | 13-2 |
| Opening the SISO Design Tool | 13-3 |
| Using the SISO Design Task Node | 13-4 |
| The SISO Design Task Node | 13-4 |
| SISO Design Task Node Menu Bar | 13-4 |
| Using the SISO Design Task in the Controls & Estimation Tools Manager | 13-11 |

| | |
|--|--------------|
| Architecture | 13-11 |
| Compensator Editor | 13-18 |
| Graphical Tuning | 13-19 |
| Analysis Plots | 13-23 |
| Automated Tuning | 13-24 |
| SISO Design Task Graphical Tuning Window | 13-42 |
| Using the Graphical Tuning Window Menu Bar | 13-44 |
| Overview of the Graphical Tuning Window Menu Bar | 13-44 |
| File | 13-44 |
| Edit | 13-47 |
| View | 13-48 |
| Analysis | 13-49 |
| Tools | 13-50 |
| Window | 13-54 |
| Help | 13-54 |
| Using the Graphical Tuning Window Toolbar | 13-56 |
| Using the Right-Click Menus in the Graphical Tuning | |
| Window | 13-57 |
| Overview of the Right-Click Menus | 13-57 |
| Add Pole/Zero | 13-58 |
| Delete Pole/Zero | 13-61 |
| Edit Compensator | 13-62 |
| Gain Target | 13-62 |
| Show | 13-62 |
| Multimodel Display | 13-63 |
| Design Requirements | 13-63 |
| Grid | 13-76 |
| Full View | 13-76 |
| Properties | 13-77 |
| Select Compensator | 13-78 |
| Status Pane | 13-78 |
| LTI Viewer for SISO Design Task Design | |
| Requirements | 13-79 |
| Overview of LTI Viewer Design Requirements | 13-79 |
| Available Design Requirements in the LTI Viewer | 13-79 |
| Example: Time Domain Requirement | 13-80 |

| | |
|--|-------|
| Basic LTI Viewer Tasks | 14-2 |
| Using the Right-Click Menu in the LTI Viewer | 14-4 |
| Overview of the Right-Click Menu | 14-4 |
| Setting Characteristics of Response Plots | 14-4 |
| Adding Design Requirements | 14-9 |
| Importing, Exporting, and Deleting Models in the LTI Viewer | 14-12 |
| Importing Models | 14-12 |
| Exporting Models | 14-13 |
| Deleting Models | 14-14 |
| Selecting Response Types | 14-16 |
| Methods for Selecting Response Types | 14-16 |
| Right Click Menu: Plot Type | 14-16 |
| Plot Configurations Window | 14-16 |
| Line Styles Editor | 14-18 |
| Analyzing MIMO Models | 14-20 |
| Overview of Analyzing MIMO Models | 14-20 |
| Array Selector | 14-21 |
| I/O Grouping for MIMO Models | 14-23 |
| Selecting I/O Pairs | 14-24 |
| Customizing the LTI Viewer | 14-25 |
| Overview of Customizing the LTI Viewer | 14-25 |
| LTI Viewer Preferences Editor | 14-25 |

Linear System Modeling

- Chapter 1, “Linear System Model Objects”
- Chapter 2, “Model Creation”

Linear System Model Objects

- “About Model Objects” on page 1-2
- “Model Object Families” on page 1-6
- “Models with Fixed Coefficients” on page 1-10
- “Models with Tunable Coefficients” on page 1-19
- “Using Linear System Model Objects” on page 1-23
- “Converting Between Model Types” on page 1-24
- “Simulink Block for LTI Systems” on page 1-26
- “References” on page 1-28

About Model Objects

| In this section... |
|--|
| “What Are Model Objects?” on page 1-2 |
| “Control System Modeling with Model Objects” on page 1-3 |

What Are Model Objects?

Model Objects Represent Linear Systems

In Control System Toolbox™ software, you represent linear systems as model objects. *Model objects* are specialized data containers that encapsulate model data and other attributes in a structured way. Model objects allow you to manipulate linear systems as single entities rather than keeping track of multiple data vectors, matrices, or cell arrays.

Model objects can represent single-input, single-output (SISO) systems or multiple-input, multiple-output (MIMO) systems. You can represent both continuous- and discrete-time linear systems.

The main families of model objects are:

- **Numeric Linear Time Invariant (LTI) Models** — Represent linear systems with fixed numerical coefficients.
- **Control Design Blocks** — Represent tunable parameters or tunable linear models with certain predefined structures.
- **Generalized LTI Models** — Combine numeric and tunable coefficients to support tasks such as parameter studies or compensator tuning. Generalized LTI models arise from combining numeric LTI models with Control Design Blocks.

For more information about the families of model objects, see “Model Object Families” on page 1-6.

About Model Data

The data encapsulated in your model object depends on the model type you use. For example:

- Transfer functions store the numerator and denominator coefficients
- State-space models store the A , B , C , and D matrices that describe the dynamics of the system
- PID controller models store the proportional, integral, and derivative gains

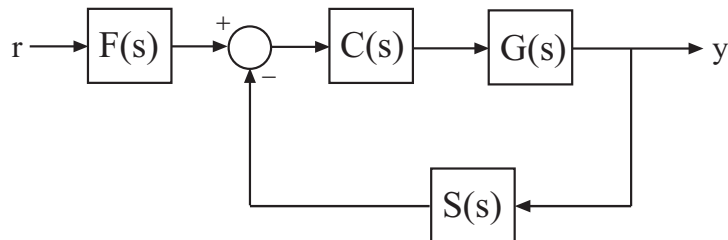
Other model attributes stored as model data include time units, names for the model inputs or outputs, and time delays. For more information about setting and retrieving model attributes, see “Model Properties” on page 3-2.

Note All model objects are MATLAB® objects, but working with them does not require a background in object-oriented programming. To learn more about objects and object syntax, see “MATLAB Classes” in the MATLAB documentation.

Control System Modeling with Model Objects

Model objects can represent individual components of a control architecture, such as the plant, actuators, sensors, or controllers. You can connect model objects to build aggregate models of block diagrams that represent the combined response of multiple elements.

For example, the following control system contains a prefilter F , a plant G , and a controller C , arranged in a single-loop configuration. The model also includes a representation of sensor dynamics, S .



You can represent each of the components as a model object. You do not need to use the same type of model object for each component. For example, represent the plant G as a zero-pole-gain (zpk) model with a double pole at $s = -1$; C as a PID controller, and F and S as transfer functions:

```
G = zpk([], [-1, -1], 1);  
C = pid(2, 1.3, 0.3, 0.5);  
S = tf(5, [1 4]);  
F = tf(1, [1 1]);
```

You can then combine these elements build models that represent your control system or the control system as a whole. For example, create the open-loop response SGC :

```
open_loop = S*G*C;
```

To build a model of the unfiltered closed-loop response, use the `feedback` command:

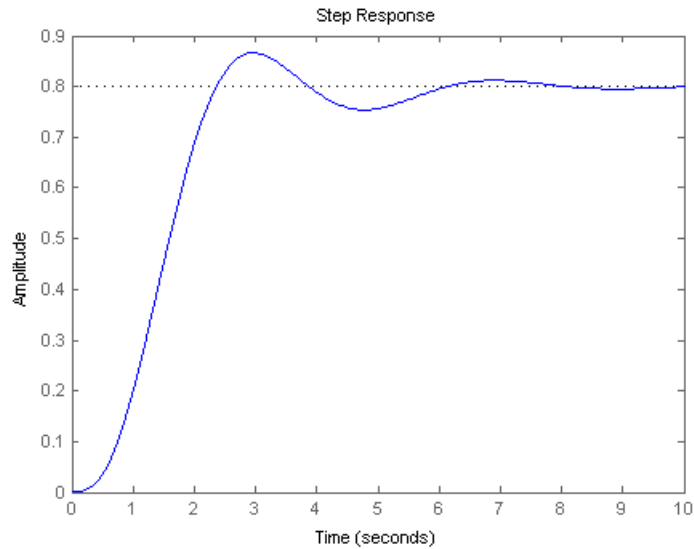
```
T = feedback(G*C, S);
```

To model the entire closed-loop system response from r to y , combine T with the filter transfer function:

```
Try = T*F;
```

The results `open_loop`, T , and `Try` are also linear model objects. You can operate on them with Control System Toolbox control design and analysis commands. For example, plot the step response of the entire system:

```
step(Try)
```



When you combine Numeric LTI models, the resulting Numeric LTI models represent the aggregate model. The resulting models do not retain the original data from the combined components. For example, T does not separately keep track of the dynamics of the components G, C, and S that are combined to create T.

To learn more about building control architectures by connecting model objects, see “Arithmetic Operations on LTI Models” on page 5-15 and “Model Interconnection Functions” on page 5-20.

Model Object Families

| In this section... |
|--|
| “Dynamic vs. Static Models” on page 1-6 |
| “Numeric Linear Time Invariant (LTI) Models” on page 1-7 |
| “Control Design Blocks” on page 1-7 |
| “Generalized Matrices” on page 1-8 |
| “Generalized LTI Models” on page 1-9 |

Dynamic vs. Static Models

Control System Toolbox model objects fall into one of the following two classes: Dynamic System Models and Static Models.

Dynamic System Models

Dynamic System Models represent systems that have internal dynamics or memory of past states (such as integrators). This family includes:

- All Numeric LTI models — Basic numeric representation of linear systems or components of linear systems
- Control Design Blocks that have dynamics, such as `ltiblock.tf`, `ltiblock.ss`, or `ltiblock.pid`
- All Generalized LTI models — Represent systems having both fixed and tunable (or parametric) coefficients

Most analysis commands work on any type of Dynamic System model object. For Control Design Blocks and Generalized LTI models, analysis commands use the current value of the tunable parameters.

Static Models

Static Models represent static input/output relationships and generalize the notions of matrix and numeric array to parametric arrays. You can use static models to create parametric expressions and to construct Generalized LTI

models whose coefficients are parametric expressions. The Static Models family includes:

- Tunable parameters (`realp` objects)
- Generalized matrices (`genmat` objects)

For more information about using these objects to create parametric models, see “Models with Tunable Coefficients” on page 1-19,

Numeric Linear Time Invariant (LTI) Models

Numeric LTI models are the basic numeric representation of linear systems or components of linear systems. Use numeric LTI models for modeling dynamic components, such as transfer functions or state-space models, whose coefficients are fixed, numeric values. You can use numeric LTI models for linear analysis or control design tasks. For more information about numeric LTI models and their applications, see “Models with Fixed Coefficients” on page 1-10.

The following table summarizes the available types of numeric LTI models.

| Model Type | Description |
|---------------------|---|
| <code>tf</code> | Transfer function model in polynomial form |
| <code>zpk</code> | Transfer function model in zero-pole-gain (factorized) form |
| <code>ss</code> | State-space model |
| <code>frd</code> | Frequency response data model |
| <code>pid</code> | Parallel-form PID controller |
| <code>pidstd</code> | Standard-form PID controller |

Control Design Blocks

Control Design Blocks are building blocks for constructing tunable models of control systems. Combine Control Design Blocks with Numeric LTI models to create Generalized LTI models that include both fixed and tunable

components. For more information about using Control Design Blocks, see “Models with Tunable Coefficients” on page 1-19.

Control Design Blocks include tunable parameter objects as well as tunable linear models with predefined structure. The following table summarizes the available types of Control Design Blocks.

| Family | Model Type | Description |
|----------------------------------|----------------------------|---|
| Tunable Linear Components | <code>ltiblock.gain</code> | Tunable static gain block |
| | <code>ltiblock.tf</code> | SISO fixed-order transfer function with tunable coefficients |
| | <code>ltiblock.ss</code> | Fixed-order state-space model with tunable coefficients |
| | <code>ltiblock.pid</code> | One-degree-of-freedom PID controller with tunable coefficients |
| Tunable Parameter | <code>realp</code> | Static control design block representing a tunable real parameter |

Generalized Matrices

Generalized Matrices extend the notion of numeric matrices to matrices that include parametric or tunable values. Create generalized matrices by building rational expressions involving `realp` parameters. You can use generalized matrices as inputs to `tf` or `ss` to create tunable linear models with structures other than the predefined structures of the Control Design Blocks. Use such models for parameter studies or some compensator tuning tasks.

| Model Type | Description |
|---------------------|--|
| <code>genmat</code> | Generalized matrix that includes parametric or tunable entries |

For more information about generalized matrices and their applications, see “Models with Tunable Coefficients” on page 1-19.

Generalized LTI Models

Generalized LTI models represent systems having both fixed and tunable (or parametric) coefficients. Generalized LTI models arise from combining numeric LTI models with Control Design Blocks.

For more information about Generalized LTI models and their applications, see “Models with Tunable Coefficients” on page 1-19.

| Model Type | Description |
|------------|---|
| genss | Generalized LTI model arising from combination of Numeric LTI models (except frd models) with Control Design Blocks |
| genfrd | Generalized LTI model arising from combination frd models with Control Design Blocks |

Models with Fixed Coefficients

| In this section... |
|---|
| “Numeric LTI Models” on page 1-10 |
| “Transfer Functions” on page 1-11 |
| “State-Space Models” on page 1-12 |
| “Frequency Response Data (FRD) Models” on page 1-14 |
| “PID Controllers” on page 1-16 |

Numeric LTI Models

Numeric LTI models are the basic representation of linear systems or components of linear systems whose coefficients are fixed numeric values.

Applications of Numeric LTI Models

You can use Numeric LTI models to represent block diagram components such as plant or sensor dynamics. By connecting Numeric LTI models together, you can derive Numeric LTI models of block diagrams. Use Numeric LTI models for most modeling, analysis, and control design tasks, including:

- Analyzing linear system dynamics using analysis commands such as `bode`, `step`, or `impz`.
- Designing controllers for linear systems using “SISO Design Tool” or the PID Tuner GUI.
- Designing controllers using control design commands such as `pidtune`, `rlocus`, or `lqr/lqg`.

Types of Numeric LTI Models

Control System Toolbox includes the following types of numeric LTI models:

- “Transfer Functions” on page 1-11
- “State-Space Models” on page 1-12
- “Frequency Response Data (FRD) Models” on page 1-14

- “PID Controllers” on page 1-16

Transfer Functions

- “Transfer Function Representations” on page 1-11
- “Commands for Creating Transfer Functions” on page 1-12

Transfer Function Representations

Control System Toolbox software supports transfer functions that are continuous-time or discrete-time, and SISO or MIMO. You can also have time delays in your transfer function representation.

A SISO transfer function is expressed as the ratio:

$$G(s) = \frac{N(s)}{D(s)},$$

of polynomials $N(s)$ and $D(s)$, called the numerator and denominator polynomials, respectively.

You can represent linear systems as transfer functions in polynomial or factorized (zero-pole-gain) form. For example, the polynomial-form transfer function:

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

can be rewritten in factorized form as:

$$G(s) = \frac{(s+1)(s-4)}{(s+2)(s+3)}.$$

The `tf` model object represents transfer functions in polynomial form. The `zpk` model object represents transfer functions in factorized form.

MIMO transfer functions are arrays of SISO transfer functions. For example:

$$G(s) = \begin{bmatrix} \frac{s-3}{s+4} \\ \frac{s+1}{s+2} \end{bmatrix}$$

is a one-input, two output transfer function.

Commands for Creating Transfer Functions

Use the commands described in the following table to create transfer functions.

| Command | Description |
|-------------------|---|
| <code>tf</code> | Create <code>tf</code> objects representing continuous-time or discrete-time transfer functions in polynomial form. |
| <code>zpk</code> | Create <code>zpk</code> objects representing continuous-time or discrete-time transfer functions in zero-pole-gain (factorized) form. |
| <code>filt</code> | Create <code>tf</code> objects representing discrete-time transfer functions using digital signal processing (DSP) convention. |

For examples of using these commands, see Chapter 2, “Model Creation” and the reference pages for each command.

State-Space Models

- “State-Space Model Representations” on page 1-13
- “Explicit State-Space Models” on page 1-13
- “Descriptor (Implicit) State-Space Models” on page 1-14
- “Commands for Creating State-Space Models” on page 1-14

State-Space Model Representations

State-space models rely on linear differential equations or difference equations to describe system dynamics. Control System Toolbox software supports SISO or MIMO state-space models in continuous or discrete time. State-space models can include time delays. You can represent state-space models in either explicit or descriptor (implicit) form.

State-space models can result from:

- Linearizing a set of ordinary differential equations that represent a physical model of the system.
- State-space model identification using System Identification Toolbox™ software.
- State-space realization of transfer functions. (See “Converting Between Model Types” on page 1-24 for more information.)

Use `ss` model objects to represent state-space models. For examples of creating state-space models, see Chapter 2, “Model Creation”.

Explicit State-Space Models

Explicit continuous-time state-space models have the following form:

$$\begin{aligned}\frac{dx}{dt} &= Ax + Bu \\ y &= Cx + Du\end{aligned}$$

where x is the state vector. u is the input vector, and y is the output vector. A , B , C , and D are the state-space matrices that express the system dynamics.

A discrete-time explicit state-space model takes the following form:

$$\begin{aligned}x[n+1] &= Ax[n] + Bu[n] \\ y[n] &= Cx[n] + Du[n]\end{aligned}$$

where the vectors $x[n]$, $u[n]$, and $y[n]$ are the state, input, and output vectors for the n th sample.

Descriptor (Implicit) State-Space Models

A *descriptor state-space model* is a generalized form of state-space model. In continuous time, a descriptor state-space model takes the following form:

$$E \frac{dx}{dt} = Ax + Bu$$

$$y = Cx + Du$$

where x is the state vector. u is the input vector, and y is the output vector. A , B , C , D , and E are the state-space matrices.

Commands for Creating State-Space Models

Use the commands described in the following table to create state-space models.

| Command | Description |
|---------|---|
| ss | Create explicit state-space model. |
| dss | Create descriptor (implicit) state-space model. |
| delayss | Create state-space models with specified time delays. |

For examples of using these commands, see Chapter 2, “Model Creation” and the reference pages for each command.

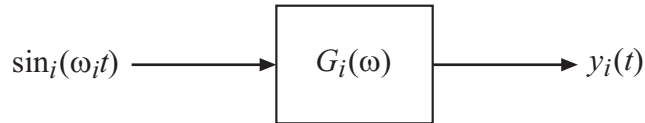
Frequency Response Data (FRD) Models

- “Frequency Response Data” on page 1-14
- “Commands for Creating FRD Models” on page 1-15

Frequency Response Data

In the Control System Toolbox software, you can use `frd` models to store, manipulate, and analyze frequency response data. An `frd` model stores a vector of frequency points with the corresponding complex frequency response data you obtain either through simulations or experimentally.

For example, suppose you measure frequency response data for the SISO system you want to model. You can measure such data by driving the system with a sine wave at a set of frequencies $\omega_1, \omega_2, \dots, \omega_n$, as shown:



At steady state, the measured response $y_i(t)$ to the driving signal at each frequency ω_i takes the following form:

$$y_i(t) = a \sin(\omega_i t + b), \quad i = 1, \dots, n.$$

The measurement yields the complex frequency response G at each input frequency:

$$G(j\omega_i) = ae^{jb}, \quad i = 1, \dots, n.$$

You can do most frequency-domain analysis tasks on frd models, but you cannot perform time-domain simulations with them. For information on frequency response analysis of linear systems, see Chapter 8 of [1].

Commands for Creating FRD Models

Use the following commands to create FRD models.

| Command | Description |
|------------|---|
| frd | Create frd objects from frequency response data. |
| frestimate | Create frd objects by estimating the frequency response of a Simulink® model. This approach requires Simulink® Control Design™ software. See “Frequency Response Estimation” in the <i>Simulink Control Design User’s Guide</i> for more information. |

For examples creating FRD models, see Chapter 2, “Model Creation” and the frd reference page.

PID Controllers

- “Continuous-Time PID Controller Representations” on page 1-16
- “Discrete-Time PID Controllers” on page 1-17

Continuous-Time PID Controller Representations

You can represent continuous-time Proportional-Integral-Derivative (PID) controllers in either parallel or standard form. The two forms differ in the parameters used to express the proportional, integral, and derivative actions and the filter on the derivative term, as shown in the following table.

| Form | Formula |
|----------|---|
| Parallel | $C = K_p + \frac{K_i}{s} + \frac{K_d s}{T_f s + 1},$ <p>where:</p> <ul style="list-style-type: none"> • K_p = proportional gain • K_i = integrator gain • K_d = derivative gain • T_f = derivative filter time |
| Standard | $C = K_p \left(1 + \frac{1}{T_i s} + \frac{T_d s}{N s + 1} \right),$ <p>where:</p> <ul style="list-style-type: none"> • K_p = proportional gain • T_i = integrator time • T_d = derivative time • N = derivative filter constant |

Use a controller form that is convenient for your application. For example, if you want to express the integrator and derivative actions in terms of time constants, use Standard form.

For examples of creating continuous-time PID Controllers, see Chapter 2, “Model Creation” and the `pid` and `pidstd` reference pages.

Discrete-Time PID Controllers

Discrete-time PID controllers are expressed by the following formulas.

| Form | Formula |
|----------|---|
| Parallel | $C = K_p + K_i IF(z) + \frac{K_d}{T_f + DF(z)},$ <p>where:</p> <ul style="list-style-type: none"> • K_p = proportional gain • K_i = integrator gain • K_d = derivative gain • T_f = derivative filter time |
| Standard | $C = K_p \left(1 + \frac{1}{T_i} IF(z) + \frac{T_d}{\frac{T_d}{N} + DF(z)} \right),$ <p>where:</p> <ul style="list-style-type: none"> • K_p = proportional gain • T_i = integrator time • T_d = derivative time • N = derivative filter constant |

$IF(z)$ and $DF(z)$ are the discrete integrator formulas for the integrator and derivative filter, respectively. Use the `IFormula` and `DFormula` properties of the `pid` or `pidstd` model objects to set the $IF(z)$ and $DF(z)$ formulas. The next table shows available formulas for $IF(z)$ and $DF(z)$. T_s is the sample time.

| IFormula or DFormula | IF(z) or DF(z) |
|-----------------------------|---------------------------------|
| ForwardEuler (default) | $\frac{T_s}{z-1}$ |
| BackwardEuler | $\frac{T_s z}{z-1}$ |
| Trapezoidal | $\frac{T_s}{2} \frac{z+1}{z-1}$ |

If you do not specify a value for `IFormula`, `DFormula`, or both, `ForwardEuler` is used by default.

For examples of creating discrete-time PID Controllers, see Chapter 2, “Model Creation” and the `pid` and `pidstd` reference pages.

Models with Tunable Coefficients

In this section...

“Generalized LTI Models” on page 1-19

“Modeling Tunable Components” on page 1-19

“Modeling Control Systems with Tunable Components” on page 1-20

“Internal Structure of Generalized Models” on page 1-21

Generalized LTI Models

Generalized LTI models represent systems having both fixed and tunable (or parametric) coefficients.

You can use Generalized LTI models to:

- Model a tunable (or parametric) component of a control system, such as a tunable low-pass filter.
- Model a control system that contains both:
 - Fixed components, such as plant dynamics and sensor dynamics
 - Tunable components, such as filters and compensators

You can use Generalized LTI models for parameter studies. For an example, see “Sample a Tunable (Parametric) Model for Parameter Studies” on page 2-66.

If you have Robust Control Toolbox™ software, you can use Generalized LTI models for tuning fixed control structures using `hinfstruct`. See “H-Infinity Tuning of Fixed Control Structures” in the *Robust Control Toolbox Getting Started Guide*.

Modeling Tunable Components

Control System Toolbox includes tunable components with predefined structure called “Control Design Blocks” on page 1-7. To create tunable components with a specific custom structure that is not covered by the Control Design Blocks:

- 1 Use the tunable real parameter `realp` or the generalized matrix `genmat` to represent the tunable coefficients of your component.
- 2 Use the resulting `realp` or `genmat` objects as inputs to `tf` or `ss` to model the component. The result is a generalized state-space (`genss`) model of the component.

For examples of creating such custom tunable components, see:

- “Tunable Low-Pass Filter” on page 2-52
- “Tunable Second-Order Filter” on page 2-53
- “State-Space Model With Both Fixed and Tunable Parameters” on page 2-54

Modeling Control Systems with Tunable Components

To construct a Generalized LTI model representing a control system with both fixed and tunable components:

- 1 Model the nontunable components of your system using Numeric LTI models.
- 2 Model each tunable component using Control Design Blocks or expressions involving such blocks. See “Modeling Tunable Components” on page 1-19.
- 3 Use model interconnection commands such as `series`, `parallel` or `connect`, or the arithmetic operators `+`, `-`, `*`, `/`, `\`, and `^`, to combine all the components of your system.

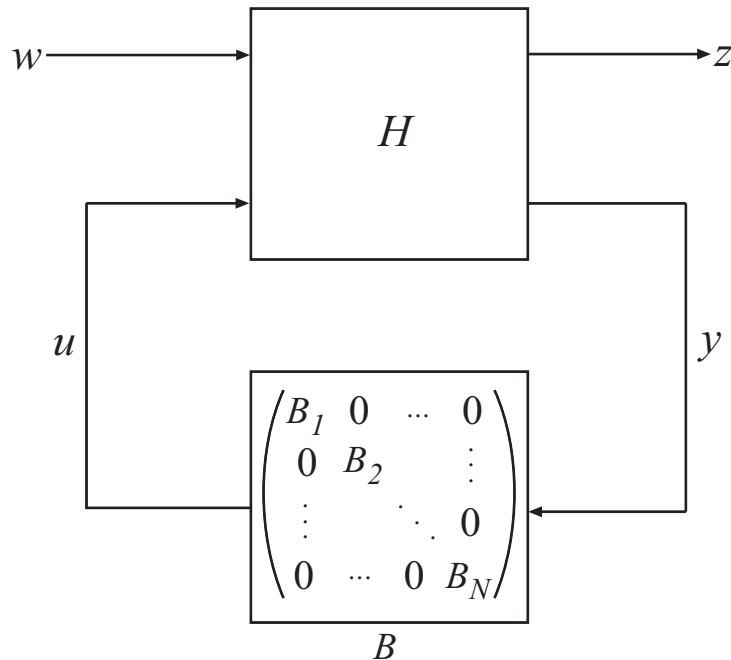
The resulting model is:

- A `genss` model, if none of the nontunable components is a frequency response data model (for example, `frd`)
- A `genfrd` model, if the nontunable component is a `frd` model

For an example of constructing a `genss` model of a control system with both fixed and tunable components, see “Control System With Tunable Components” on page 2-55.

Internal Structure of Generalized Models

A Generalized model separately stores the numeric and parametric portions of the model by structuring the model in *Standard Form*, as shown in the following illustration.



w and z represent the inputs and outputs of the Generalized model.

H represents all portions of the Generalized model that have fixed (non-parametric) coefficients. H is:

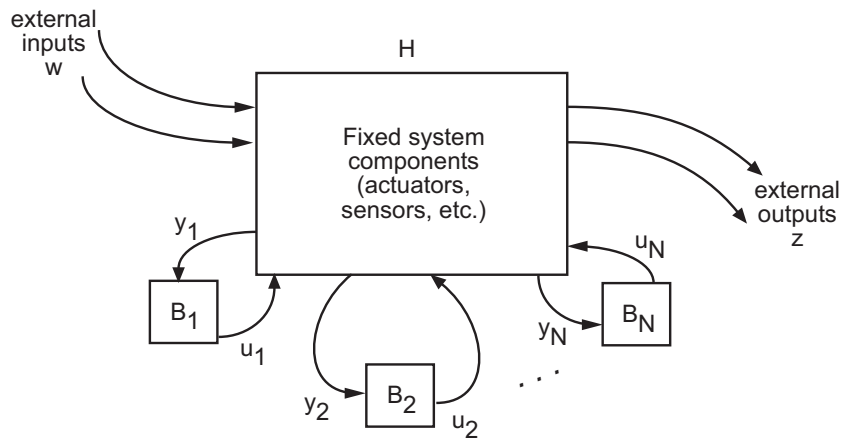
- A state-space (ss) model, for genss models
- A frequency response data (frd) model, for genfrd models
- A matrix, for genmat models

B represents the parametric components of the Generalized model, which are the Control Design Blocks B_1, \dots, B_N . The `Blocks` property of the Generalized model stores a list of the names of these blocks. If the Generalized model has

blocks that occur multiple times in B_1, \dots, B_N , these are only listed once in the `Blocks` property.

To access the internal representation of a Generalized model, including `H` and `B`, use the `getLFTModel` command.

This Standard Form can represent any control structure. To understand why, consider the control structure as an aggregation of fixed-coefficient elements interacting with the parametric elements:



To rewrite this in Standard Form, define

$$u := [u_1, \dots, u_N]$$

$$y := [y_1, \dots, y_N],$$

and group the tunable control elements B_1, \dots, B_N into the block-diagonal configuration C . P includes all the fixed components of the control architecture—actuators, sensors, and other nontunable elements—and their interconnections.

Using Linear System Model Objects

After you represent your dynamic system as a model object, you can:

- Attach additional information to the model using model attributes (properties). See “Model Properties” on page 3-2.
- Manipulate the model using arithmetic and model interconnection operations. See Chapter 5, “Operations on Models”.
- Analyze the model response using commands such as `bode` and `step`. See Chapter 6, “Model Analysis Tools”.
- Perform parameter studies using model arrays. See “Model Arrays” on page 2-58.
- Design compensators. You can:
 - Design compensators for systems specified as numeric LTI models. Available compensator design techniques include PID tuning, root locus analysis, pole placement, LQG optimal control, and frequency domain loop-shaping. See “Designing Compensators”.
 - Manually tune many control architectures using the Chapter 13, “SISO Design Tool”.
 - Use `hinfstruct` to automatically tune a control system that you represent as a `genss` model with tunable blocks (requires Robust Control Toolbox software). See “H-Infinity Tuning of Fixed Control Structures” in the *Robust Control Toolbox Getting Started Guide*.

Converting Between Model Types

In this section...

“Explicit Conversion” on page 1-24

“Automatic Conversion” on page 1-24

“Caution About Model Conversions” on page 1-25

Explicit Conversion

You can convert from one type of model to another using the model-creation command for the target model type (for example, `ss`, `tf`, `pid`, or `genss`). In general, you can convert from any model type to any other. However, there are a few limitations. For example:

- You cannot convert from `frd` to analytic model types such as `ss`, `tf`, or `zpk` (unless you perform system identification with System Identification Toolbox software)
- You cannot convert `ss` models with internal delays to `tf` or `zpk`

You can convert between Numeric LTI models and Generalized LTI models:

- Converting a Generalized LTI model to a Numeric LTI model evaluates any Control Design Blocks at their current (nominal) value.
- Converting a Numeric LTI model to a Generalized LTI model creates a Generalized LTI model with an empty `Blocks` property.

For information about converting to a particular model type, see the reference page for that model type.

Automatic Conversion

Some algorithms operate only on one type of model object. For example, the algorithm for zero-order-hold discretization with `c2d` can only be performed on state-space models. Similarly, commands such as `tfdata` expect one particular type of model (`tf`). For convenience, such commands automatically convert input models to the appropriate or required model type. For example, in

```
sys = ss(0,1,1,0)
[num,den] = tfdata(sys)
```

`tfdata` first converts the state-space model `sys` to an equivalent transfer function in order to return numerator and denominator data.

Conversions to state-space models are not uniquely defined. For this reason, automatic conversions to state space are disabled when the result depends on the choice of state coordinates, for example, in commands like `initial` or `kalman`.

Caution About Model Conversions

When manipulating or converting LTI models, keep in mind that:

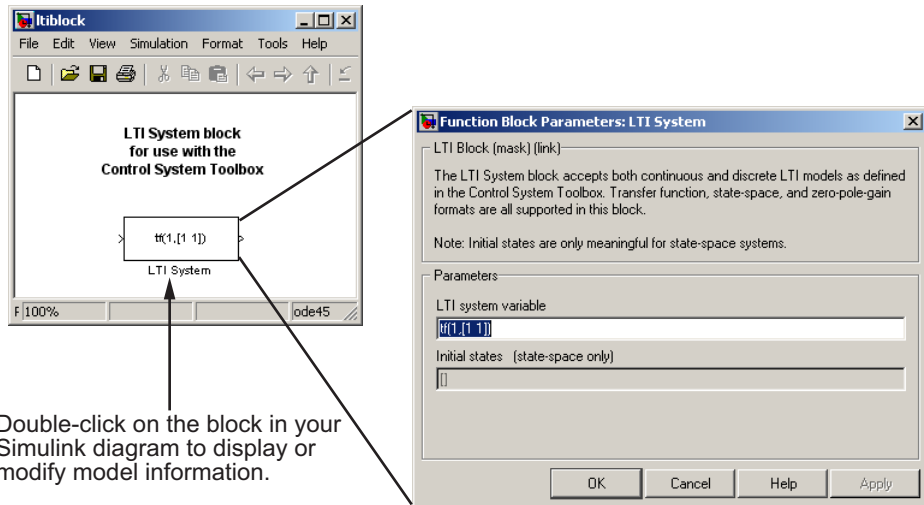
- The three LTI model types TF, ZPK, and SS, are not equally well-suited for numerical computations. In particular, the accuracy of computations using high-order transfer functions is often poor. Therefore, it is often preferable to work with the state-space representation. In addition, it is often beneficial to balance and scale state-space models. You get this type of balancing automatically when you convert any TF or ZPK model to state space using `ss`.
- Conversions to the transfer function representation using `tf` may incur a loss of accuracy. As a result, the transfer function poles may noticeably differ from the poles of the original zero-pole-gain or state-space model.
- Conversions to state space are not uniquely defined in the SISO case, nor are they guaranteed to produce a minimal realization in the MIMO case. For a given state-space model `sys`,

```
ss(tf(sys))
```

may return a model with different state-space matrices, or even a different number of states in the MIMO case. Therefore, if possible, it is best to avoid converting back and forth between state-space and other model types.

Simulink Block for LTI Systems

You can incorporate model objects into Simulink diagrams using the LTI System block shown below.



Double-click on the block in your Simulink diagram to display or modify model information.

The LTI System block can be accessed either by typing

```
ltiblock
```

at the MATLAB prompt or by selecting **Control System Toolbox** from the **Blocksets and Toolboxes** section of the main Simulink library.

The LTI System block consists of the dialog box shown on the right in the figure above. In the editable text box labeled **LTI system variable**, enter either the variable name of an LTI object located in the MATLAB workspace (for example, `sys`) or a MATLAB expression that evaluates to an LTI object (for example, `tf(1,[1 1])`). The LTI System block accepts both continuous and discrete LTI objects in either transfer function, zero-pole-gain, or state-space form. All types of delays are supported in the LTI block. Simulink converts the model to its state-space equivalent prior to initializing the simulation.

Use the editable text box labeled **Initial states** to enter an initial state vector for state-space models. The concept of "initial state" is not well-defined for transfer functions or zero-pole-gain models, as it depends on the choice of state coordinates used by the realization algorithm. As a result, you cannot enter nonzero initial states when you supply TF or ZPK models to LTI blocks in a Simulink diagram.

References

- [1] Dorf, R.C. and R.H. Bishop, *Modern Control Systems*, Addison-Wesley, Menlo Park, CA, 1998.

Model Creation

- “Single-Input Single-Output (SISO) Model Creation” on page 2-2
- “Discrete-Time Model Creation” on page 2-10
- “Multi-Input Multi-Output (MIMO) Model Creation” on page 2-14
- “Models with Time Delays” on page 2-21
- “Models with Parametric or Tunable Coefficients” on page 2-52
- “Model Arrays” on page 2-58
- “References” on page 2-69

Single-Input Single-Output (SISO) Model Creation

How to create continuous-time, single-input single-output (SISO) models.

In this section...

“Transfer Function Model Using Numerator and Denominator Coefficients” on page 2-2

“Transfer Function Model Using Zeros, Poles, and Gain” on page 2-3

“State-Space Model” on page 2-4

“Frequency-Response Model” on page 2-5

“Proportional-Integral-Derivative (PID) Controller” on page 2-6

Transfer Function Model Using Numerator and Denominator Coefficients

This example shows how to create continuous-time, single-input single-output (SISO) transfer functions from their numerator and denominator coefficients using `tf`.

Create the transfer function $G(s) = \frac{s}{s^2 + 3s + 2}$:

```
num = [1 0];  
den = [1 3 2];  
G = tf(num,den);
```

`num` and `den` are the numerator and denominator polynomial coefficients in descending powers of s . For example, `den = [1 3 2]` represents the denominator polynomial $s^2 + 3s + 2$.

`G` is a `tf` model object, which is a data container for representing transfer functions in polynomial form.

Tip Alternatively, you can specify the transfer function $G(s)$ as an expression in s :

1 Create a transfer function model for the variable s .

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

2 Specify $G(s)$ as a ratio of polynomials in s .

```
G = s/(s^2 + 3*s + 2);
```

More About

- “About Model Objects” on page 1-2
- “Transfer Functions” on page 1-11

Transfer Function Model Using Zeros, Poles, and Gain

This example shows how to create single-input, single-output (SISO) transfer functions in factored form using `zpk`.

Create the factored transfer function $G(s) = 5 \frac{s}{(s+1+i)(s+1-i)(s+2)}$:

```
Z = [0];
P = [-1-1i -1+1i -2];
K = 5;
G = zpk(Z,P,K);
```

Z and P are the zeros and poles (the roots of the numerator and denominator, respectively). K is the gain of the factored form. For example, $G(s)$ has a real pole at $s = -2$ and a pair of complex poles at $s = -1 \pm i$. The vector $P = [-1-1i -1+1i -2]$ specifies these pole locations.

G is a `zpk` model object, which is a data container for representing transfer functions in zero-pole-gain (factorized) form.

More About

- “About Model Objects” on page 1-2
- “Transfer Functions” on page 1-11

State-Space Model

This example shows how to create a continuous-time single-input, single-output (SISO) state-space model from state-space matrices using `ss`.

Create a model of an electric motor where the state-space equations are:

$$\begin{aligned}\frac{dx}{dt} &= Ax + Bu \\ y &= Cx + Du\end{aligned}$$

where the state variables are the angular position θ and angular velocity $d\theta/dt$:

$$x = \begin{bmatrix} \theta \\ \frac{d\theta}{dt} \end{bmatrix},$$

u is the electric current, the output y is the angular velocity, and the state-space matrices are:

$$A = \begin{bmatrix} 0 & 1 \\ -5 & -2 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 3 \end{bmatrix}, \quad C = [0 \ 1], \quad D = [0].$$

To create this model, enter:

```
A = [0 1; -5 -2];  
B = [0; 3];  
C = [0 1];  
D = 0;  
sys = ss(A,B,C,D);
```

`sys` is an `ss` model object, which is a data container for representing state-space models.

Tip To represent a system of the form:

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

use `dss`. This command creates a `ss` model with a nonempty `E` matrix, also called a descriptor state-space model. See “Multi-Input Multi-Output Descriptor State-Space Model” on page 2-17 for an example.

More About

- “About Model Objects” on page 1-2
- “State-Space Models” on page 1-12

Frequency-Response Model

This example shows how to create a single-input, single-output (SISO) frequency-response model using `frd`.

A frequency-response model stores a vector of frequency points with corresponding complex frequency response data you obtain either through simulations or experimentally. Thus, if you measure the frequency response of your system at a set of test frequencies, you can use the data to create a frequency response model:

1 Load the frequency response data in `AnalyzerData.mat`.

```
load AnalyzerData
```

This command loads the data into the MATLAB workspace as the column vectors `freq` and `resp`. The variables `freq` and `resp` contain 256 test frequencies and the corresponding complex-valued frequency response points, respectively.

Tip To inspect these variables, enter:

```
whos freq resp
```

2 Create a frequency response model.

```
sys = frd(resp,freq);
```

`sys` is an `frd` model object, which is a data container for representing frequency response data.

You can use `frd` models with many frequency-domain analysis commands. For example, visualize the frequency response data using `bode`.

Tip By default, the `frd` command assumes that the frequencies are in radians/second. To specify different frequency units, use the `TimeUnit` and `FrequencyUnit` properties of the `frd` model object. For example:

```
sys = frd(resp,freq,'TimeUnit','min','FrequencyUnit','rad/TimeUnit')
```

sets the frequency units to radians/minute.

More About

- “About Model Objects” on page 1-2
- “Frequency Response Data (FRD) Models” on page 1-14

Proportional-Integral-Derivative (PID) Controller

How to create models representing PID controllers.

- “Continuous-Time PID Controller Representations” on page 2-7
- “Create Continuous-Time Parallel-Form PID Controller” on page 2-8
- “Create Continuous-Time Standard-Form PID Controller” on page 2-8

Continuous-Time PID Controller Representations

You can represent continuous-time Proportional-Integral-Derivative (PID) controllers in either parallel or standard form. The two forms differ in the parameters used to express the proportional, integral, and derivative actions and the filter on the derivative term, as shown in the following table.

| Form | Formula |
|----------|---|
| Parallel | $C = K_p + \frac{K_i}{s} + \frac{K_d s}{T_f s + 1},$ <p>where:</p> <ul style="list-style-type: none"> • K_p = proportional gain • K_i = integrator gain • K_d = derivative gain • T_f = derivative filter time |
| Standard | $C = K_p \left(1 + \frac{1}{T_i s} + \frac{T_d s}{N s + 1} \right),$ <p>where:</p> <ul style="list-style-type: none"> • K_p = proportional gain • T_i = integrator time • T_d = derivative time • N = derivative filter constant |

Use a controller form that is convenient for your application. For example, if you want to express the integrator and derivative actions in terms of time constants, use Standard form.

See “Discrete-Time Proportional-Integral-Derivative (PID) Controller” on page 2-11 for information on creating PID Controllers in discrete time.

Create Continuous-Time Parallel-Form PID Controller

This example shows how to create a continuous-time Proportional-Integral-Derivative (PID) controller in parallel form using `pid`.

Create the following parallel-form PID controller: $C = 29.5 + \frac{26.2}{s} - \frac{4.3s}{0.06s + 1}$.

```
Kp = 29.5;
Ki = 26.2;
Kd = 4.3;
Tf = 0.06;
C = pid(Kp,Ki,Kd,Tf)
```

C is a `pid` model object, which is a data container for representing parallel-form PID controllers.

Create Continuous-Time Standard-Form PID Controller

This example shows how to create a continuous-time Proportional-Integral-Derivative (PID) controller in standard form using `pidstd`.

Create the following standard-form PID controller:

$$C = 29.5 \left(1 + \frac{1}{1.13s} + \frac{0.15s}{\frac{0.15}{2.3}s + 1} \right)$$

```
Kp = 29.5;
Ti = 1.13;
Td = 0.15;
N = 2.3;
C = pid(Kp,Ki,Kd,Tf)
```

C is a `pidstd` model object, which is a data container for representing standard-form PID controllers.

More About

- “About Model Objects” on page 1-2
- “PID Controllers” on page 1-16

Discrete-Time Model Creation

How to create discrete-time models.

| In this section... |
|--|
| “Discrete-Time Transfer Function Model” on page 2-10 “Discrete-Time Proportional-Integral-Derivative (PID) Controller” on page 2-11 |

Discrete-Time Transfer Function Model

This example shows how to create a discrete-time transfer function model using `tf`.

Create the transfer function $G(z) = \frac{z}{z^2 - 2z - 6}$ with a sampling time of 0.1 s.

```
num = [1 0];  
den = [1 -2 -6];  
Ts = 0.1;  
G = tf(num,den,Ts)
```

`num` and `den` are the numerator and denominator polynomial coefficients in descending powers of z . `G` is a `tf` model object.

Tip Create a discrete-time `zpk`, `ss`, and `frd` models in a similar way by appending a sampling period to the input arguments. For examples, see the reference pages for those commands.

The sampling time is stored in the `Ts` property of `G`. Access `Ts`, using dot notation:

```
G.Ts
```


More About

- “About Model Objects” on page 1-2
- “Models with Fixed Coefficients” on page 1-10

Discrete-Time Proportional-Integral-Derivative (PID) Controller**Discrete-Time PID Controller Representations**

Discrete-time PID controllers are expressed by the following formulas.

| Form | Formula |
|----------|---|
| Parallel | $C = K_p + K_i IF(z) + \frac{K_d}{T_f + DF(z)},$ <p>where:</p> <ul style="list-style-type: none"> • K_p = proportional gain • K_i = integrator gain • K_d = derivative gain • T_f = derivative filter time |
| Standard | $C = K_p \left(1 + \frac{1}{T_i} IF(z) + \frac{T_d}{\frac{T_d}{N} + DF(z)} \right),$ <p>where:</p> <ul style="list-style-type: none"> • K_p = proportional gain • T_i = integrator time • T_d = derivative time • N = derivative filter constant |

$IF(z)$ and $DF(z)$ are the discrete integrator formulas for the integrator and derivative filter, respectively. Use the `IFormula` and `DFormula` properties of the `pid` or `pidstd` model objects to set the $IF(z)$ and $DF(z)$ formulas. The next table shows available formulas for $IF(z)$ and $DF(z)$. T_s is the sample time.

| IFormula or DFormula | IF(z) or DF(z) |
|------------------------|---------------------------------|
| ForwardEuler (default) | $\frac{T_s}{z-1}$ |
| BackwardEuler | $\frac{T_s z}{z-1}$ |
| Trapezoidal | $\frac{T_s}{2} \frac{z+1}{z-1}$ |

If you do not specify a value for `IFormula`, `DFormula`, or both, `ForwardEuler` is used by default.

Create Discrete-Time Standard-Form PID Controller

This example shows how to create a standard-form discrete-time Proportional-Integral-Derivative (PID) controller that has $K_p = 29.5$, $T_i = 1.13$, $T_d = 0.15$, $N = 2.3$, and sample time $T_s = 0.1$:

```
C = pidstd(29.5,1.13,0.15,2.3,0.1,...
          'IFormula','Trapezoidal','DFormula','BackwardEuler')
```

This command creates a `pidstd` model with $IF(z) = \frac{T_s}{2} \frac{z+1}{z-1}$ and

$$DF(z) = \frac{T_s z}{z-1}.$$

You can set the discrete integrator formulas for a parallel-form controller in the same way, using `pid`.

More About

- “About Model Objects” on page 1-2
- “PID Controllers” on page 1-16

Multi-Input Multi-Output (MIMO) Model Creation

How to create multi-input multi-output (MIMO) models.

In this section...

“Multi-Input Multi-Output Transfer Function Model” on page 2-14
 “Multi-Input Multi-Output State-Space Model” on page 2-15
 “Multi-Input Multi-Output Descriptor State-Space Model” on page 2-17
 “Multi-Input Multi-Output Frequency Response Data Model” on page 2-18
 “Select Input/Output Pairs in Multi-Input Multi-Output (MIMO) Models”
 on page 2-20

Multi-Input Multi-Output Transfer Function Model

This example shows how to create a multi-input multi-output (MIMO) transfer function model by concatenating single-input single-output (SISO) transfer function models.

To create the two-input, one-output MIMO transfer function:

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

perform the following steps:

- 1 Create SISO transfer functions for each channel.

```
g11 = tf([1 -1],[1 1]);
g21 = tf([1 2],[1 4 5]);
```

Tip Use `zpk` instead of `tf` to create MIMO transfer functions in factorized form.

2 Concatenate the transfer functions.

$$G = [g_{11}; g_{21}];$$

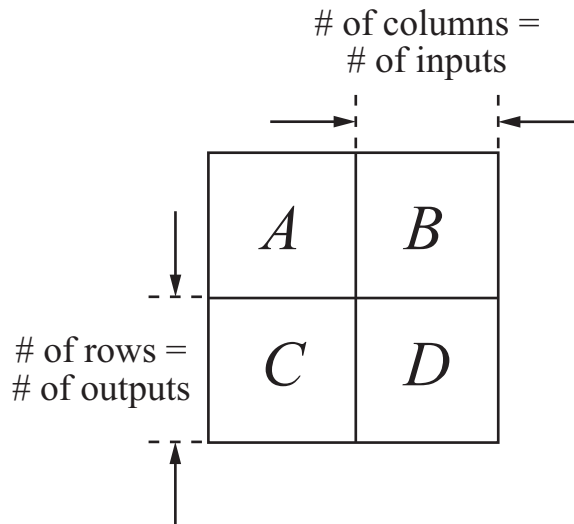
More About

- “About Model Objects” on page 1-2
- “Transfer Functions” on page 1-11

Multi-Input Multi-Output State-Space Model

This example shows how to create a MIMO state-space model using `ss`.

You create a MIMO state-space model in the same way as you create a single-input, single-output (SISO) state-space model. The only difference between the SISO and MIMO cases is the dimensions of the state-space matrices. The dimensions of the B , C , and D matrices increase with the numbers of inputs and outputs as shown in the following illustration.



In this example, you create a state-space model for a rotating body with inertia tensor J , damping force F , and three axes of rotation, related as:

$$J \frac{d\omega}{dt} + F\omega = T$$
$$y = \omega.$$

The system input T is the driving torque. The output y is the vector of angular velocities of the rotating body.

To express this system in state-space form:

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

rewrite it as:

$$\frac{d\omega}{dt} = -J^{-1}F\omega + J^{-1}T$$
$$y = \omega.$$

Then the state-space matrices are:

$$A = -J^{-1}F, \quad B = J^{-1}, \quad C = I, \quad D = 0.$$

To create this model, enter the following commands:

```
J = [8 -3 -3; -3 8 -3; -3 -3 8];  
F = 0.2*eye(3);  
A = -J\F;  
B = inv(J);  
C = eye(3);  
D = 0;  
sys_mimo = ss(A,B,C,D);
```

These commands assume that J is the inertia tensor of a cube rotating about its corner, and the damping force has magnitude 0.2.

`sys_mimo` is an ss model.

More About

- “About Model Objects” on page 1-2
- “State-Space Models” on page 1-12

Multi-Input Multi-Output Descriptor State-Space Model

This example shows how to create a continuous-time descriptor (implicit) state-space model using `dss`.

Note This example uses the same rotating-body system shown in “Multi-Input Multi-Output State-Space Model” on page 2-15, where you inverted the inertia matrix J to obtain the value of the B matrix. If J is poorly-conditioned for inversion, you can instead use a descriptor (implicit) state-space model.

A *descriptor (implicit) state-space model* is of the form:

$$E \frac{dx}{dt} = Ax + Bu$$

$$y = Cx + Du$$

Create a state-space model for a rotating body with inertia tensor J , damping force F , and three axes of rotation, related as:

$$J \frac{d\omega}{dt} + F\omega = T$$

$$y = \omega.$$

The system input T is the driving torque. The output y is the vector of angular velocities of the rotating body. You can write this system as a descriptor state-space model having the following state-space matrices:

$$A = -F, \quad B = I, \quad C = I, \quad D = 0, \quad E = J.$$

To create this system, enter:

```
J = [8 -3 -3; -3 8 -3; -3 -3 8];  
F = 0.2*eye(3);  
A = -F;  
B = eye(3);  
C = eye(3);  
D = 0;  
E = J;  
sys_mimo = dss(A,B,C,D,E)
```

These commands assume that J is the inertia tensor of a cube rotating about its corner, and the damping force has magnitude 0.2.

sys is an ss model with a nonempty E matrix.

More About

- “About Model Objects” on page 1-2
- “State-Space Models” on page 1-12

Multi-Input Multi-Output Frequency Response Data Model

This example shows how to create a MIMO frequency-response model using `frd`.

Frequency response data for a MIMO system includes a vector of complex response data for each of the input/output (I/O) pair of the system. Thus, if you measure the frequency response of each I/O pair your system at a set of test frequencies, you can use the data to create a frequency response model:

1 Load frequency response data in `AnalyzerDataMIMO.mat`.

```
load AnalyzerDataMIMO H11 H12 H21 H22 freq
```

This command loads the data into the MATLAB workspace as five column vectors `H11`, `H12`, `H21`, `H22`, and `freq`. The vector `freq` contains 100 test frequencies. The other four vectors contain the corresponding

complex-valued frequency response of each I/O pair of a two-input, two-output system.

Tip To inspect these variables, enter:

```
whos H11 H12 H21 H22 freq
```

2 Organize the data into a three-dimensional array.

```
Hresp = zeros(2,2,length(freq));  
Hresp(1,1,:) = H11;  
Hresp(1,2,:) = H12;  
Hresp(2,1,:) = H21;  
Hresp(2,2,:) = H22;
```

The dimensions of `Hresp` are the number of outputs, number of inputs, and the number of frequencies for which there is response data. `Hresp(i,j,:)` contains the frequency response from input `j` to output `i`.

3 Create a frequency-response model.

```
H = frd(Hresp,freq);
```

`sys` is an `frd` model object, which is a data container for representing frequency response data.

You can use `frd` models with many frequency-domain analysis commands. For example, visualize the response of this two-input, two-output system using `bode`.

Tip By default, the `frd` command assumes that the frequencies are in radians/second. To specify different frequency units, use the `TimeUnit` and `FrequencyUnit` properties of the `frd` model object. For example:

```
sys = frd(Hresp,freq,'TimeUnit','min','FrequencyUnit','rad/TimeUnit')
```

sets the frequency units to in radians/minute.

More About

- “About Model Objects” on page 1-2
- “Frequency Response Data (FRD) Models” on page 1-14

Select Input/Output Pairs in Multi-Input Multi-Output (MIMO) Models

This example shows how to select the response from the first input to the second output of a MIMO model.

- 1 Create a two-input, one-output transfer function.

```
N = {[1 -1],[1];[1 2],[3 1 4]};  
D = [1 1 10];  
H = tf(N,D)
```

Note For more information about using cell arrays to create MIMO transfer functions, see the `tf` reference page.

- 2 Select the response from the second input to the output of H.

To do this, use MATLAB array indexing.

```
H12 = H(1,2)
```

For any MIMO system `H`, the index notation `H(i,j)` selects the response from the `j`th input to the `i`th output.

Models with Time Delays

How to create models that include time delays.

In this section...

“About Modeling Time Delays” on page 2-21

“First Order Plus Dead Time Model” on page 2-21

“Input and Output Delay in State-Space Model” on page 2-23

“Transport Delay in Multi-Input, Multi-Output (MIMO) Transfer Function” on page 2-25

“Closing Feedback Loops with Time Delays” on page 2-26

“Discrete-Time Transfer Function with Time Delay” on page 2-29

“Time-Delay Approximation” on page 2-29

“Frequency Response Data (FRD) Model with Time Delay” on page 2-44

“About Internal Delays” on page 2-46

About Modeling Time Delays

Use the following model properties to represent time delays in linear systems.

- `InputDelay`, `OutputDelay` — Time delays at system inputs or outputs
- `ioDelay`, `InternalDelay` — Time delays that are internal to the system

In discrete-time models, these properties are constrained to integer values that represent delays expressed as integer multiples of the sampling time. To approximate discrete-time models with delays that are a fractional multiple of the sampling time, use `thiran`.

First Order Plus Dead Time Model

This example shows how to create a first order plus dead time model using the `InputDelay` or `OutputDelay` properties of `tf`.

To create the following first-order transfer function with a 2.1 s time delay:

$$G(s) = e^{-2.1s} \frac{1}{s+10},$$

enter:

```
G = tf(1,[1 10], 'InputDelay',2.1)
```

where InputDelay specifies the delay at the input of the transfer function.

Tip You can use InputDelay with zpk the same way as with tf:

```
G = zpk([],-10,1, 'InputDelay',2.1)
```

For SISO transfer functions, a delay at the input is equivalent to a delay at the output. Therefore, the following command creates the same transfer function:

```
G = tf(1,[1 10], 'OutputDelay',2.1)
```

Use dot notation to examine or change the value of a time delay. For example, change the time delay to 3.2 as follows:

```
G.OutputDelay = 3.2;
```

To see the current value, enter:

```
G.OutputDelay
```

```
ans =
```

```
3.2000
```

Tip An alternative way to create a model with a time delay is to specify the transfer function with the delay as an expression in s :

1 Create a transfer function model for the variable s .

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

2 Specify $G(s)$ as an expression in s .

```
G = exp(-2.1*s)/(s+10);
```

More About

- “About Model Objects” on page 1-2
- “Transfer Functions” on page 1-11

Input and Output Delay in State-Space Model

This example shows how to create state-space models with delays at the inputs and outputs, using the `InputDelay` or `OutputDelay` properties of `ss`.

Create a state-space model describing the following one-input, two-output system:

$$\frac{dx(t)}{dt} = -2x(t) + 3u(t-1.5)$$
$$y(t) = \begin{bmatrix} x(t-0.7) \\ -x(t) \end{bmatrix}.$$

This system has an input delay of 1.5. The first output has an output delay of 0.7, and the second output is not delayed.

Note In contrast to SISO transfer functions, input delays are not equivalent to output delays for state-space models. Shifting a delay from input to output in a state-space model requires introducing a time shift in the model states. For example, in the model of this example, defining $T = t - 1.5$ and $X(T) = x(T + 1.5)$ results in the following equivalent system:

$$\frac{dX(T)}{dT} = -2X(T) + 3u(T)$$
$$y(T) = \begin{bmatrix} X(T - 2.2) \\ -X(T - 1.5) \end{bmatrix}.$$

All of the time delays are on the outputs, but the new state variable X is time-shifted relative to the original state variable x . Therefore, if your states have physical meaning, or if you have known state initial conditions, consider carefully before shifting time delays between inputs and outputs.

To create this system:

1 Define the state-space matrices.

$$\begin{aligned} A &= -2; \\ B &= 3; \\ C &= [1; -1]; \\ D &= 0; \end{aligned}$$

2 Create the model.

$$G = \text{ss}(A,B,C,D, 'InputDelay', 1.5, 'OutputDelay', [0.7;0])$$

G is a `ss` model.

Tip Use `delays` to create state-space models with more general combinations of input, output, and state delays, of the form:

$$\frac{dx}{dt} = Ax(t) + Bu(t) + \sum_{j=1}^N (A_j x(t - t_j) + B_j u(t - t_j))$$

$$y(t) = Cx(t) + Du(t) + \sum_{j=1}^N (C_j x(t - t_j) + D_j u(t - t_j))$$

More About

- “About Model Objects” on page 1-2
- “State-Space Models” on page 1-12

Transport Delay in Multi-Input, Multi-Output (MIMO) Transfer Function

This example shows how to create a MIMO transfer function with different transport delays for each input-output (I/O) pair.

Create the MIMO transfer function:

$$H(s) = \begin{bmatrix} e^{-0.1} \frac{2}{s} & e^{-0.3} \frac{s+1}{s+10} \\ 10 & e^{-0.2} \frac{s-1}{s+5} \end{bmatrix}.$$

Time delays in MIMO systems can be specific to each I/O pair, as in this example. You cannot use `InputDelay` and `OutputDelay` to model I/O-specific transport delays. Instead, use `ioDelay` to specify the transport delay across each I/O pair.

To create this MIMO transfer function:

- 1 Create a transfer function model for the variable `s`.

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

- Use the variable `s` to specify the transfer functions of `H` without the time delays.

```
H = [2/s (s+1)/(s+10); 10 (s-1)/(s+5)];
```

- Specify the `ioDelay` property of `H` as an array of values corresponding to the transport delay for each I/O pair.

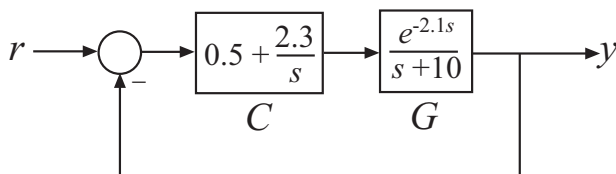
```
H.ioDelay = [0.1 0.3; 0 0.2];
```

`H` is a two-input, two-output `tf` model. Each I/O pair in `H` has the time delay specified by the corresponding entry in `tau`.

Closing Feedback Loops with Time Delays

This example shows how internal delays arise when you interconnect models that have input, output, or transport time delays.

Create a model of the following control architecture:



`G` is the plant model, which has an input delay. `C` is a proportional-integral (PI) controller.

To create a model representing the closed-loop response of this system:

- Create the plant `G` and the controller `C`.

```
G = tf(1,[1 10], 'InputDelay',2.1)
C = pid(0.5,2.3);
```

`C` has a proportional gain of 0.5 and an integral gain of 2.3.

- Use `feedback` to compute the closed-loop response from `r` to `y`.


```
T = feedback(C*G,1);
```

The time delay in T is not an input delay as it is in G . Because the time delay is internal to the closed-loop system, the software returns T as an `ss` model with an *internal time delay* of 2.1 seconds.

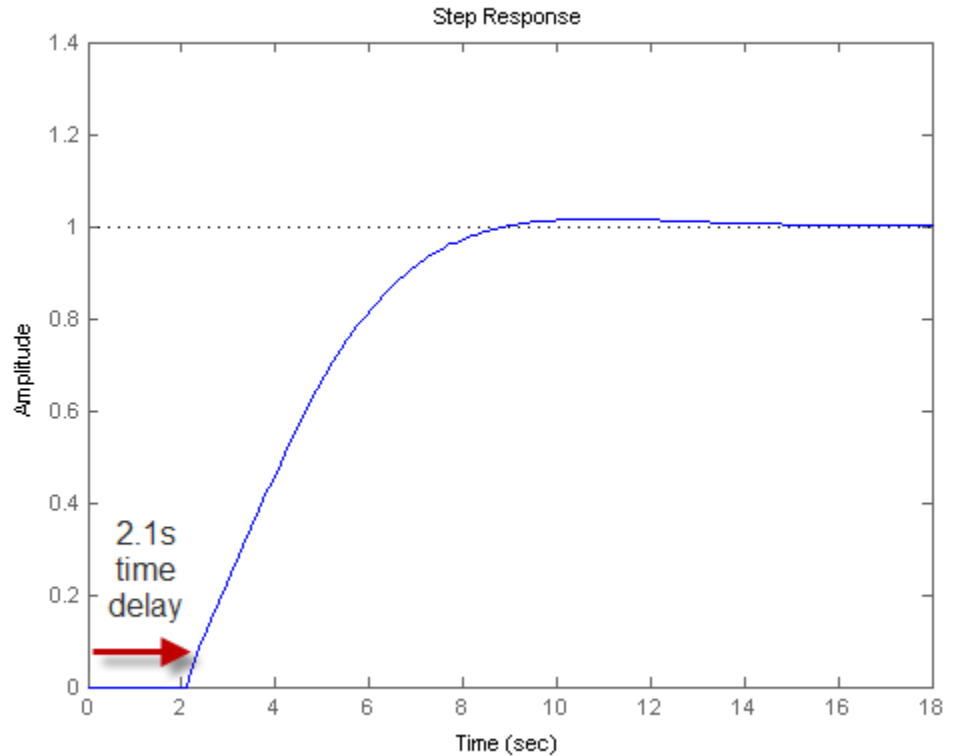
Note In addition to `feedback`, any “System Interconnections” function (including `parallel` and `series`) can give rise to internal delays.

T is an exact representation of the closed-loop response, not an approximation. To access the internal delay value, enter:

```
T.InternalDelay
```

A step plot of T confirms the presence of the time delay:

```
step(T)
```



Note Most analysis commands, such as `step`, `bode` and `margin`, support models with internal delays.

The internal time delay is stored in the `InternalDelay` property of `T`. Use dot notation to access `InternalDelay`. For example, to change the internal delay to 3.5 seconds, enter:

```
T.InternalDelay = 3.5
```

You cannot modify the number of internal delays because they are structural properties of the model.

More About

- “About Internal Delays” on page 2-46
- “Model Interconnection Functions” on page 5-20

Discrete-Time Transfer Function with Time Delay

This example shows how to create a discrete-time transfer function with a time delay.

Specify time delays for discrete-time models in the same way as for continuous-time models, except for discrete-time models, delay values must be integer multiples of the sampling time. For example, create the following first-order transfer function, with a sampling time of $T_s = 0.1$ s:

$$H(z) = z^{-25} \frac{2}{z - 0.95}.$$

```
H = tf(2,[1 -0.95],0.1,'InputDelay',25)
```

Setting `InputDelay` to 25 results in a delay of 25 sampling periods.

Tip Use `thiran` to approximate discrete-time models with delays that are a fractional multiple of the sampling time.

Time-Delay Approximation

- “About Time-Delay Approximation” on page 2-30
- “Time-Delay Approximation in Continuous-Time Open-Loop Model” on page 2-30
- “Time-Delay Approximation in Continuous-Time Closed-Loop Model” on page 2-34

- “Approximate Different Delays with Different Approximation Orders” on page 2-39
- “Convert Time Delay to Factors of $1/z$ in Discrete-Time Model” on page 2-40
- “Fractional Time-Delay Approximation in Discrete-Time Model” on page 2-42

About Time-Delay Approximation

Many control design algorithms cannot handle time delays directly. For example, techniques such as root locus, LQG, and pole placement do not work properly if time delays are present. A common technique is to replace delays with all-pass filters that approximate the delays.

To approximate time delays in continuous-time models, use the `pade` command to compute a Padé approximation. The Padé approximation is valid only at low frequencies, and provides better frequency-domain approximation than time-domain approximation. It is therefore important to compare the true and approximate responses to choose the right approximation order and check the approximation validity.

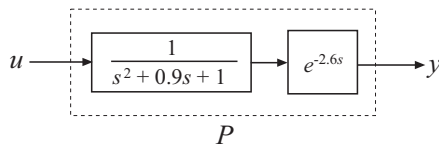
For discrete-time models, use `delay2z` to convert a time delay to factors of $1/z$ where the time delay is an integer multiple of the sampling time. Use `thiran` to approximate fractional time delays.

Time-Delay Approximation in Continuous-Time Open-Loop Model

This example shows how to approximate delays in a continuous-time open-loop system using `pade`.

Padé approximation is helpful when using analysis or design tools that do not support time delays.

- 1 Create sample open-loop system with an output delay.



```
s = tf('s');
P = exp(-2.6*s)/(s^2+0.9*s+1);
```

P is a second-order transfer function (tf) object with a time delay.

- 2** Compute the first-order Padé approximation of P.

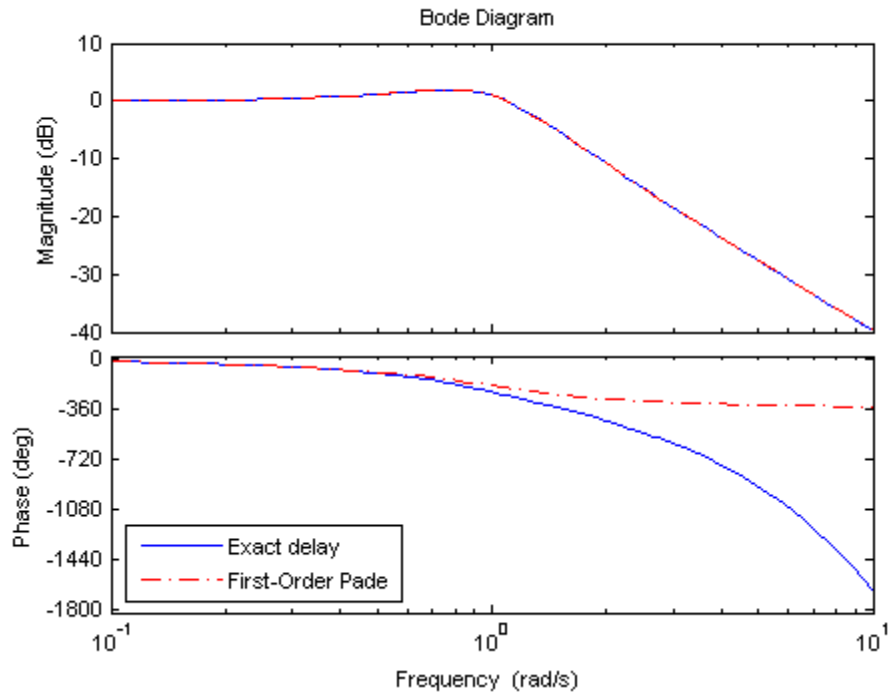
```
Pnd1 = pade(P,1)
```

This command replaces all time delays in P with a first-order approximation. Therefore, Pnd1 is a third-order transfer function with no delays:

$$\frac{-s + 0.7692}{s^3 + 1.669 s^2 + 1.692 s + 0.7692}$$

- 3** Compare the frequency response of the original and approximate models using bodeplot.

```
h = bodeoptions;
h.PhaseMatching = 'on';
bodeplot(P, '-b', Pnd1, '-.r', {0.1, 10}, h)
legend('Exact delay', 'First-Order Pade', 'Location', 'SouthWest')
```



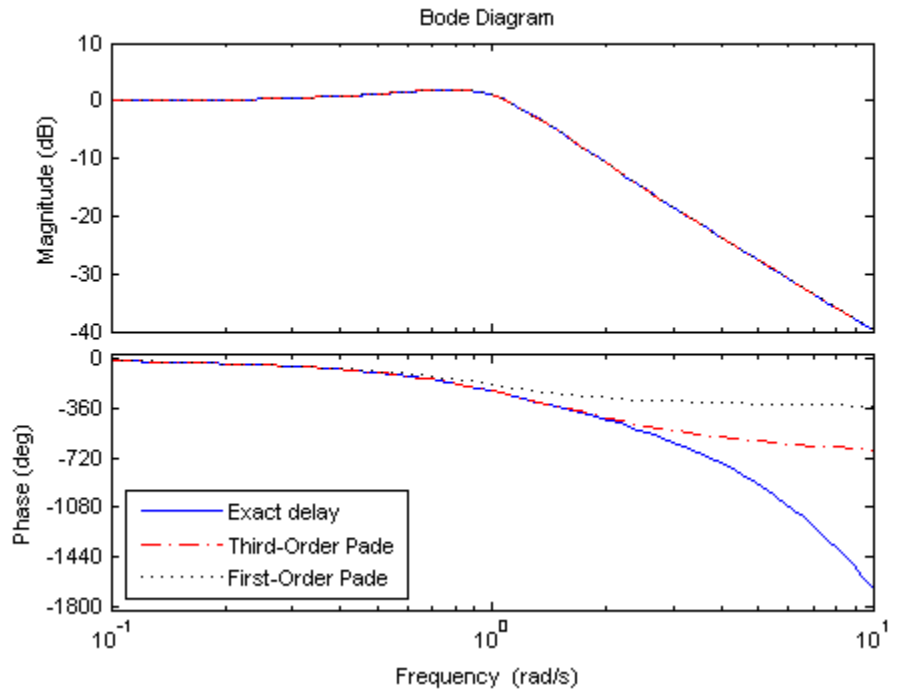
The magnitude of P and Pnd1 match exactly. However, the phase of Pnd1 deviates from the phase of P beyond approximately 1 rad/s.

- 4 Increase the Padé approximation order to extend the frequency band in which the phase approximation is good.

```
Pnd3 = pade(P,3);
```

- 5 Compare the frequency response of P, Pnd1 and Pnd3.

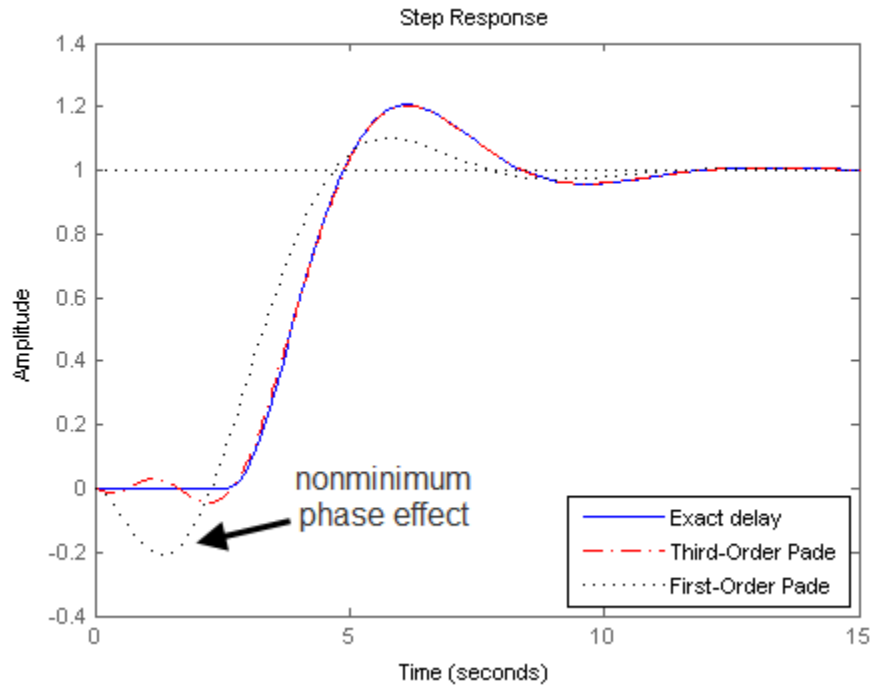
```
bodeplot(P, '-b', Pnd3, '-.r', Pnd1, ':k', {0.1 10}, h)
legend('Exact delay', 'Third-Order Pade', 'First-Order Pade', ...
      'Location', 'SouthWest')
```



The phase approximation error is reduced by using a third-order Padé approximation.

- 6** Compare the time domain responses of the original and approximated systems using step.

```
step(P, '-b', Pnd3, '-.r', Pnd1, ':k')
legend('Exact delay', 'Third-Order Padé', 'First-Order Padé', ...
      'Location', 'Southeast')
```



Using the Padé approximation introduces a nonminimum phase artifact (“wrong way” effect) in the initial transient response. The effect is reduced in the higher-order approximation.

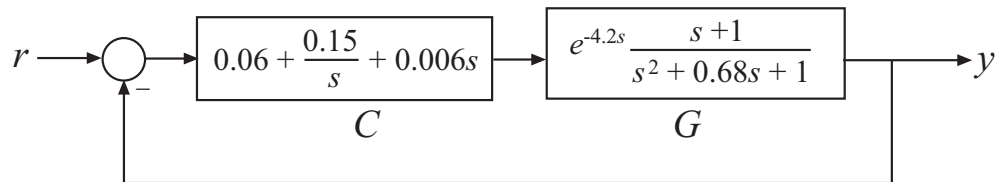
Note Using too high an approximation order may result in numerical issues and possibly unstable poles. Therefore, avoid Padé approximations with order $N > 10$.

Time-Delay Approximation in Continuous-Time Closed-Loop Model

This example shows how to approximate delays in a continuous-time closed-loop system with internal delays, using pade.

Padé approximation is helpful when using analysis or design tools that do not support time delays.

- 1 Create sample continuous-time closed-loop system with an internal delay.



Construct a model Tc1 of the closed-loop transfer function from r to y.

```
s = tf('s');
G = (s+1)/(s^2+.68*s+1)*exp(-4.2*s);
C = pid(0.06,0.15,0.006);
Tc1 = feedback(G*C,1);
```

Tc1 contains an internal delay.

```
Tc1.InternalDelay
ans =
    4.2000
```

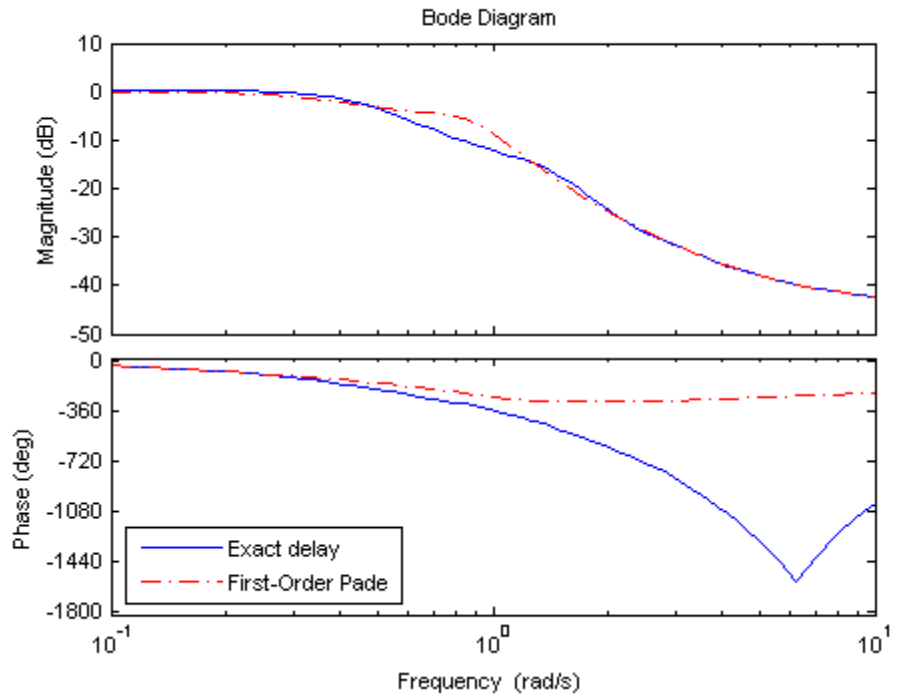
- 2 Compute the first-order Padé approximation of Tc1.

```
Tnd1 = pade(Tc1,1);
```

Tnd1 is a fourth-order state-space (ss) model with no delays.

- 3 Compare the frequency response of the original and approximate models using bodeplot.

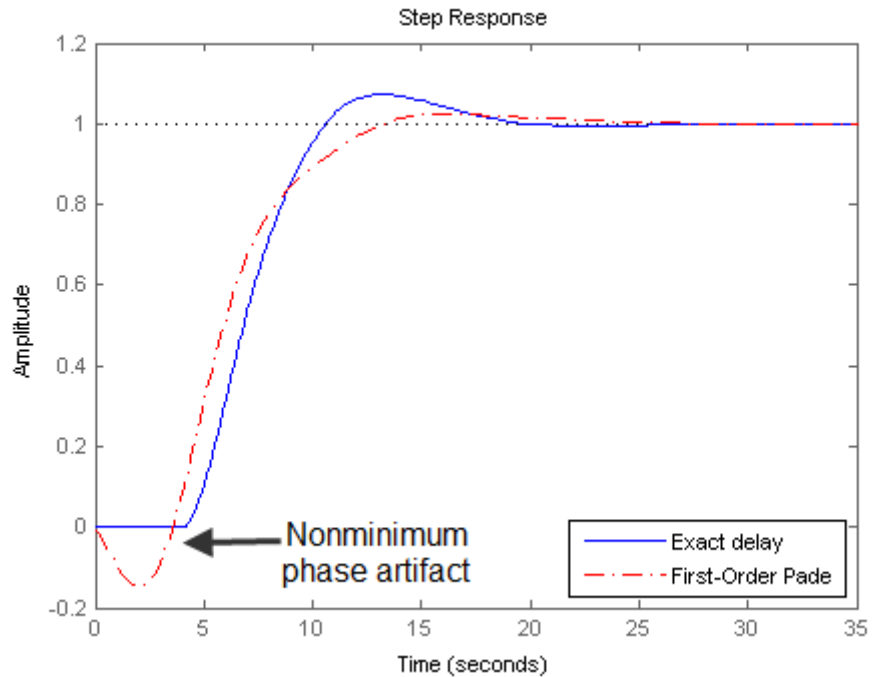
```
h = bodeoptions;
h.PhaseMatching = 'on';
bodeplot(Tc1, '-b', Tnd1, '-.r', {1, 10}, h);
legend('Exact delay', 'First-Order Pade', 'Location', 'SouthWest');
```



The magnitude and phase approximation errors are significant beyond 1 rad/s.

4 Compare the time domain response of Tc1 and Tnd1 using step.

```
step(Tc1, '-b', Tnd1, '-.r');
legend('Exact delay', 'First-Order Pade', 'Location', 'SouthEast');
```



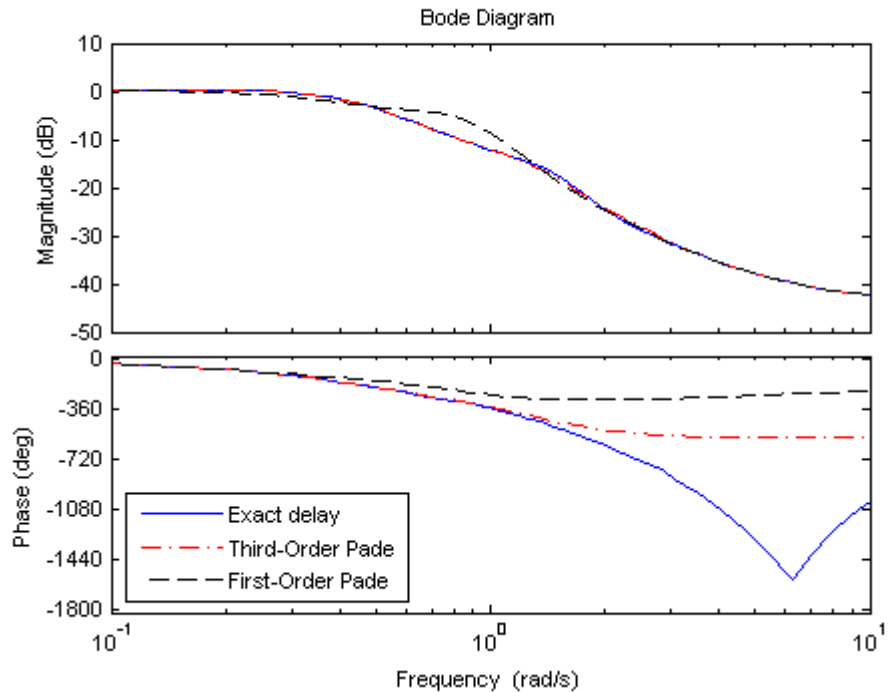
Using the Padé approximation introduces a nonminimum phase artifact (“wrong way” effect) in the initial transient response.

- 5** Increase the Padé approximation order to see if this will extend the frequency with good phase and magnitude approximation.

```
Tnd3 = pade(Tc1,3);
```

- 6** Observe the behavior of the third-order Padé approximation of Tc1. Compare the frequency response of Tc1 and Tnd3.

```
bode(Tc1,'-b',Tnd3,'-.r',Tnd1,'-k',{.1,10},h);
legend('Exact delay','Third-Order Padé','First-Order Padé',...
'Location','SouthWest');
```



The magnitude and phase approximation errors are reduced when a third-order Padé approximation is used.

Increasing the Padé approximation order extends the frequency band where the approximation is good. However, too high an approximation order may result in numerical issues and possibly unstable poles. Therefore, avoid Padé approximations with order $N > 10$.

More About

- “About Internal Delays” on page 2-46

Approximate Different Delays with Different Approximation Orders

This example shows how to specify different Padé approximation orders to approximate internal and output delays in a continuous-time open-loop system.

- 1 Load sample continuous-time open-loop system that contains internal and output time delays.

```
load PadeApproximation1 sys;
```

`sys` is a second-order continuous-time ss model with internal delay 3.4 s and output delay 1.5 s.

Tip Enter `get(sys)` for more properties of `sys`.

- 2 Use `pade` to compute a third-order approximation of the internal delay and a first-order approximation of the output delay.

```
P13 = pade(sys,inf,1,3);
```

The three input arguments following `sys` specify the approximation orders of any input, output, and internal delays of `sys`, respectively. `inf` specifies that a delay is not to be approximated. The approximation orders for the output and internal delays are one and three respectively.

`P13` is a sixth-order continuous-time ss model with no delays.

- 3 (Optional) For comparison, approximate only the internal delay of `sys`, leaving the output delay intact.

```
P3 = pade(sys,inf,inf,3);
```

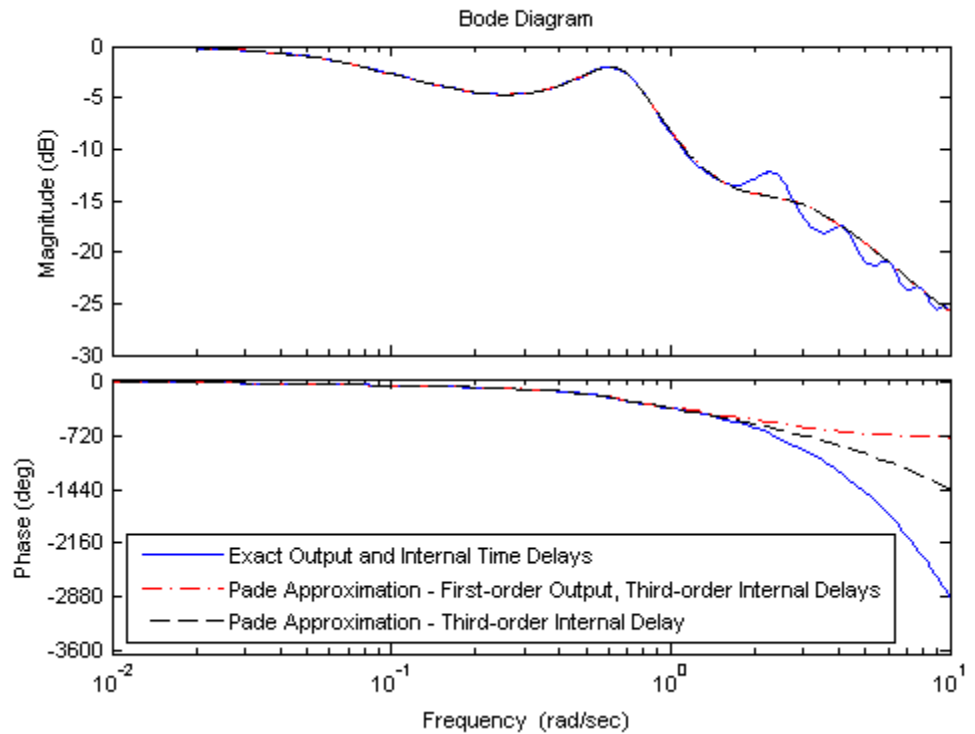
`P3` is a fifth-order continuous-time ss model with an output delay of 1.5 s. The internal delay is approximated and absorbed into the state-space matrices.

- 4 (Optional) Compare the frequency response of the exact and approximated systems. `sys`, `P13`, `P3`.

```

h = bodeoptions;
h.PhaseMatching = 'on';
bode(sys, 'b-', P13, 'r-.', P3, 'k--', h, {.01, 10});
legend('Exact Output and Internal Time Delays',...
      'Pade Approximation - First-order Output, ...
      Third-order Internal Delays',...
      'Pade Approximation - Third-order Internal Delay',...
      'location', 'SouthWest')

```

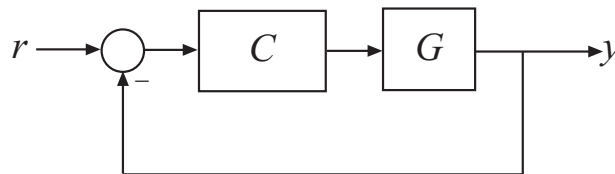


Convert Time Delay to Factors of $1/z$ in Discrete-Time Model

This example shows how to convert a time delay in a discrete-time model to factors of $1/z$ using `delay2z`.

In a discrete-time model, a time delay of one sampling interval is equivalent to a factor of $1/z$ (a pole at $z = 0$) in the model. Therefore, time delays stored in the `InputDelay`, `OutputDelay`, or `ioDelay` properties of a discrete-time model can be rewritten in the model dynamics by rewriting them as poles at $z = 0$. However, the additional poles increase the order of the system. Particularly for large time delays, this can yield systems of very high order, leading to long computation times or numerical inaccuracies.

To illustrate how to eliminate time delays in a discrete-time closed-loop model, and to observe the effects of doing so, create the following closed-loop system:



where G is a first-order discrete-time system with an input delay, and C is a PI controller.

```
G = ss(0.9,0.125,0.08,0,'Ts',0.01,'InputDelay',7);
C = pid(6,90,0,0,'Ts',0.01);
T = feedback(C*G,1);
```

T is a second-order state-space model with an internal delay, as the following commands show:

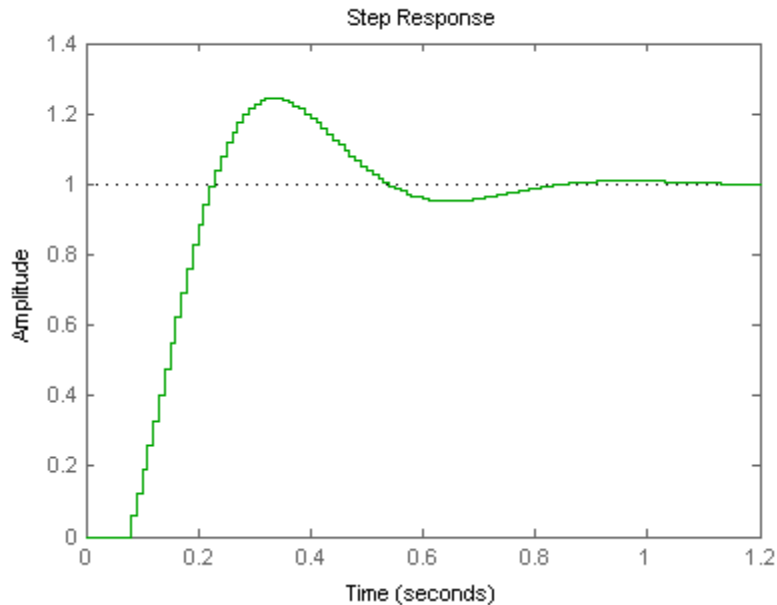
```
order(T)
T.InternalDelay
```

Use `delay2z` to replace the internal delay by z^{-7} .

```
Tnd = delay2z(T);
Tnd.InternalDelay
```

Tnd has `InternalDelay = 0`, and the step response of Tnd exactly matches that of T :

```
step(T,Tnd)
```



However, `Tnd` is a ninth-order model, due to the seven extra poles introduced by absorbing the seven-unit delay into the model. To see this, as the following command shows:

```
order(Tnd)
```

Fractional Time-Delay Approximation in Discrete-Time Model

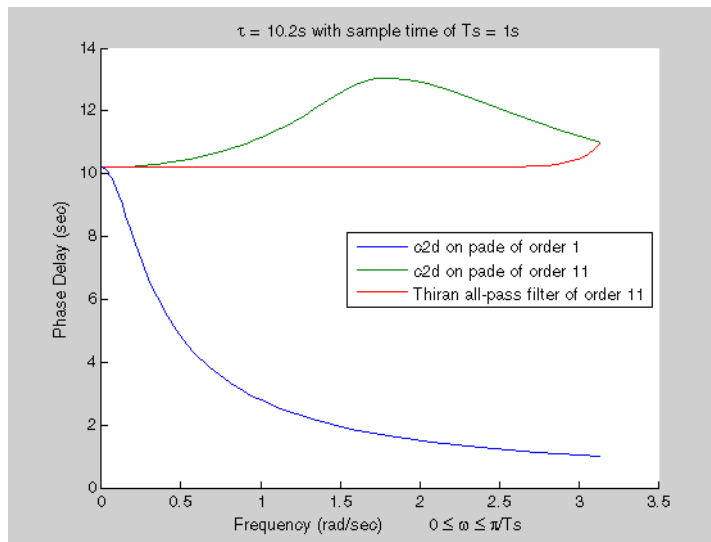
Use the `thiran` command to approximate a time delay that is a fractional multiple of the sampling time as a Thiran all-pass filter.

For a time delay of `tau` and a sampling time of `Ts`, the syntax `thiran(tau, Ts)` creates a discrete-time transfer function that is the product of two terms:

- A term representing the integer portion of the time delay as a pure line delay, $(1/z)^N$, where $N = \text{ceil}(\text{tau}/T_s)$.
- A term approximating the fractional portion of the time delay ($\text{tau} - NT_s$) as a Thiran all-pass filter.

Discretizing a Padé approximation does not guarantee good phase matching between the continuous-time delay and its discrete approximation. Using `thiran` to generate a discrete-time approximation of a continuous-time delay can yield much better phase matching. For example, the following figure shows the phase delay of a 10.2-second time delay discretized with a sample time of 1 s, approximated in three ways:

- a first-order Padé approximation, discretized using the `tustin` method of `c2d`
- an 11th-order Padé approximation, discretized using the `tustin` method of `c2d`
- an 11th-order Thiran filter



The Thiran filter yields the closest approximation of the 10.2-second delay.

See the `thiran` reference page for more information about Thiran filters.

Frequency Response Data (FRD) Model with Time Delay

This example shows that absorbing time delays into frequency response data can cause undesirable phase wrapping at high frequencies.

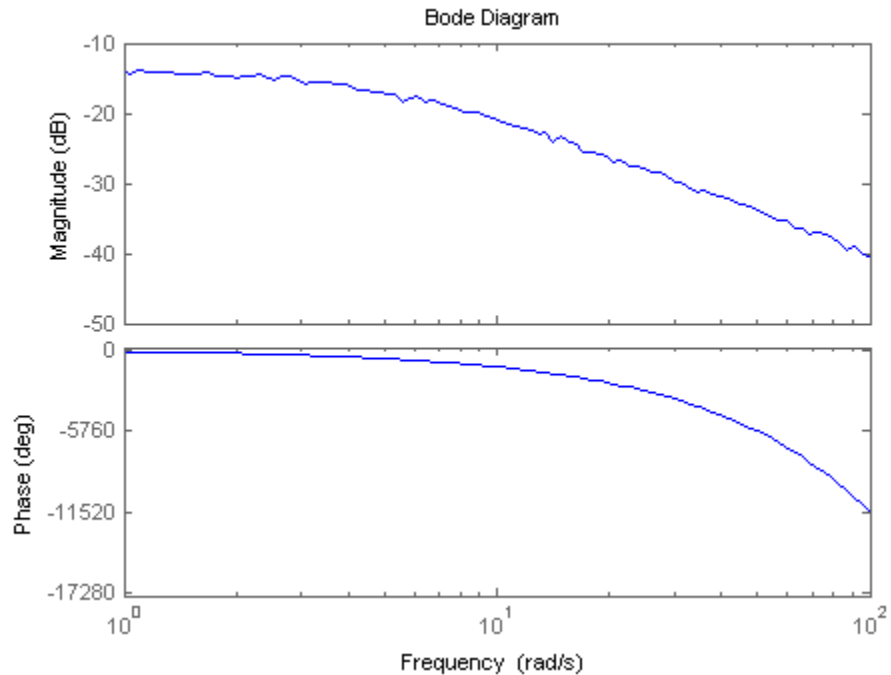
When you collect frequency response data for a system that includes time delays, you can absorb the time delay into the frequency response as a phase shift. Alternatively, if you are able to separate time delays from your measured frequency response, you can represent the delays using the `InputDelay`, `OutputDelay`, or `ioDelay` properties of the `frd` model object. The latter approach can give better numerical results, as this example illustrates.

The `frd` model `fsys` includes a transport delay of 2 s. Load the model into the MATLAB workspace and inspect the time delay with the following commands:

```
load frdexample fsys
fsys.ioDelay
```

A Bode plot of `fsys` shows the effect of the transport delay:

```
bode(fsys)
```



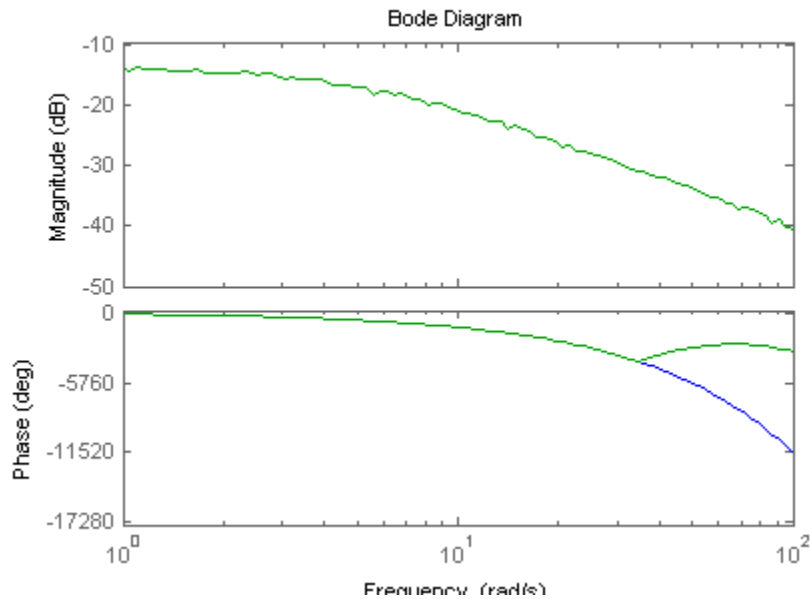
The transport delay `ioDelay = 2` introduces a rapid accumulation of phase with increasing frequency.

The `delay2z` command absorbs all time delays directly into the frequency response, resulting in an `frd` model with `ioDelay = 0`:

```
fsys2 = delay2z(fsys);
fsys2.ioDelay
```

Comparing the two ways of representing the delay shows that absorbing the delay into the frequency response causes phase-wrapping.

```
bode(fsys, fsys2)
```



Phase wrapping can introduce numerical inaccuracy at high frequencies or where the frequency grid is sparse. For that reason, if your system takes the form $e^{-\tau s}G(s)$, you might get better results by measuring frequency response data for $G(s)$ and using `InputDelay`, `OutputDelay`, or `ioDelay` to model the time delay τ .

More About

- “Frequency Response Data (FRD) Models” on page 1-14

About Internal Delays

Using the `InputDelay`, `OutputDelay`, and `ioDelay` properties, you can model simple processes with transport delays. However, these properties cannot model more complex situations, such as feedback loops with delays. In addition to the `InputDelay` and `OutputDelay` properties, state-space (ss) models have an `InternalDelay` property. This property lets you model the interconnection of systems with input, output, or transport delays, including feedback loops with delays. You can use `InternalDelay` property to

accurately model and analyze arbitrary linear systems with delays. Internal delays can arise from the following:

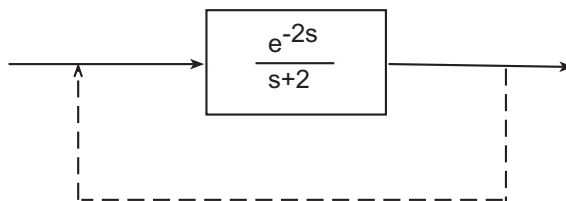
- Concatenating state-space models with input and output delays
- Feeding back a delayed signal
- Converting MIMO tf or zpk models with transport delays to state-space form

Using internal time delays, you can do the following:

- In continuous time, generate approximate-free time and frequency simulations, because delays do not have to be replaced by a Padé approximation. In continuous time, this allows for more accurate analysis of systems with long delays.
- In discrete time, keep delays separate from other system dynamics, because delays are not replaced with poles at $z = 0$, which boosts efficiency of time and frequency simulations for discrete-time systems with long delays.
- Use most Control System Toolbox functions.
- Test advanced control strategies for delayed systems. For example, you can implement and test an accurate model of a Smith predictor. See the Smith predictor demo.

Why Internal Delays Are Necessary

This example illustrates why input, output, and transport delays not enough to model all types of delays that can arise in dynamic systems. Consider the simple feedback loop with a 2 s. delay:



The closed-loop transfer function is

$$\frac{e^{-2s}}{s + 2 + e^{-2s}}$$

The delay term in the numerator can be represented as an output delay. However, the delay term in the denominator cannot. In order to model the effect of the delay on the feedback loop, the `InternalDelay` property is needed to keep track of internal coupling between delays and ordinary dynamics.

Typically, you do not create state-space models with internal delays directly, by specifying the *A*, *B*, *C*, and *D* matrices together with a set of internal delays. Rather, such models arise when you interconnect models having delays. There is no limitation on how many delays are involved and how the models are connected. For an example of creating an internal delay by closing a feedback loop, see “Closing Feedback Loops with Time Delays” on page 2-26.

Behavior of Models With Internal Delays

When you work with models having internal delays, be aware of the following behavior:

- When a model interconnection gives rise to internal delays, the software returns an `ss` model regardless of the interconnected model types. This occurs because only `ss` supports internal delays.
- The software fully supports feedback loops. You can wrap a feedback loop around any system with delays.
- When displaying the *A*, *B*, *C*, and *D* matrices, the software sets all delays to zero (creating a zero-order Padé approximation). This approximation occurs for the display only, and not for calculations using the model.

For some systems, setting delays to zero creates singular algebraic loops, which result in either improper or ill-defined, zero-delay approximations. For these systems:

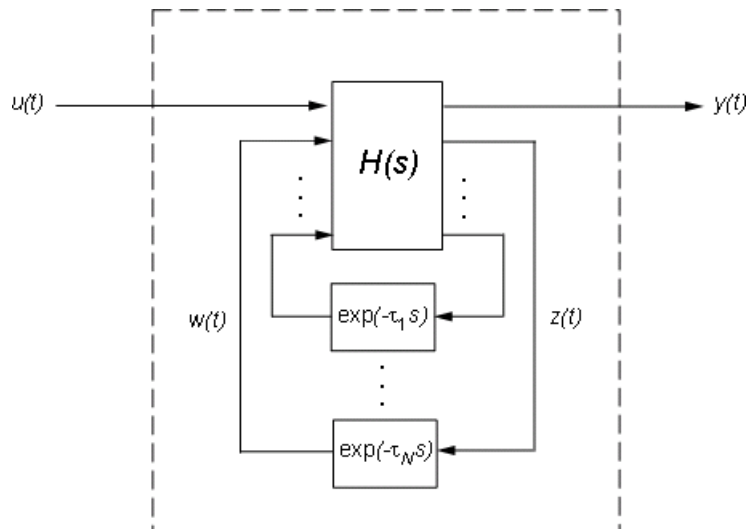
- Entering `sys` returns only sizes for the matrices of a system named `sys`.
- Entering `sys.a` produces an error.

The limited display and the error do not imply a problem with the model `sys` itself.

Inside Time Delay Models

State-space objects use generalized state-space equations to keep track of internal delays. Conceptually, such models consist of two interconnected parts:

- An ordinary state-space model $H(s)$ with an augmented I/O set
- A bank of internal delays.



The corresponding state-space equations are:

$$\begin{aligned}\dot{x} &= Ax(t) + B_1u(t) + B_2w(t) \\ y(t) &= C_1x(t) + D_{11}u(t) + D_{12}w(t) \\ z(t) &= C_2x(t) + D_{21}u(t) + D_{22}w(t) \\ w_j(t) &= z(t - \tau_j), \quad j = 1, \dots, N\end{aligned}$$

You need not bother with this internal representation to use the tools. If, however, you want to extract H or the matrices A, B_1, B_2, \dots , you can use `getDelayModel`. For the example:

$$P = 5 \cdot \exp(-3.4 \cdot s) / (s+1);$$

```
C = 0.1 * (1 + 1/(5*s));  
T = feedback(ss(P*C),1);  
[H,tau] = getDelayModel(T,'lft'); size(H)
```

Note that H is a two-input, two-output model whereas T is SISO. The inverse operation (combining H and tau to construct T) is performed by `setDelayModel`.

See [1], [2] for details.

Functions That Support Internal Time Delays

The following commands support internal delays for both continuous- and discrete-time systems:

- All interconnection functions
- Time domain response functions—except for `impulse` and `initial`
- Frequency domain functions—except for `norm`

Limitations on Functions that Support Internal Time Delays. The following commands support internal delays for both continuous- and discrete-time systems and have certain limitations:

- `allmargin`, `margin`—Uses interpolation, therefore these commands are only as precise as the fineness of the specified grid.
- `pole`, `zero`—Returns poles and zeros of the system with all delays set to zero.
- `ssdata`, `get`—If an SS model has internal delays, these commands return the A, B, C, and D matrices of the system with all internal delays set to zero. Use `getDelayModel` to access the internal state-space representation of models with internal delays.

Functions That Do Not Support Internal Time Delays

The following commands do not support internal time delays:

- System dynamics—`norm` and `lti/isstable`
- Time-domain analysis—`initial` and `initialplot`

- Model simplification—`balreal`, `balred`, and `modred`
- Compensator design—`rlocus`, `lqg`, `lqry`, `lqrd`, `kalman`, `kalmd`, `lqgreg`, `lqgtrack`, `lqi`, and `augstate`.

To use these functions on a system with internal delays, use `pade` to approximate the internal delays. See “Time-Delay Approximation” on page 2-29.

Models with Parametric or Tunable Coefficients

In this section...

“Tunable Low-Pass Filter” on page 2-52

“Tunable Second-Order Filter” on page 2-53

“State-Space Model With Both Fixed and Tunable Parameters” on page 2-54

“Control System With Tunable Components” on page 2-55

Tunable Low-Pass Filter

This example shows how to create the low-pass filter $F = a/(s + a)$ with one tunable parameter a .

You cannot use `ltiblock.tf` to represent F , because the numerator and denominator coefficients of an `ltiblock.tf` block are independent. Instead, construct F using the tunable real parameter object `realp`.

- 1 Create a tunable real parameter.

```
a = realp('a',10);
```

The `realp` object `a` is a tunable parameter with initial value 10.

- 2 Use `tf` to create the tunable filter `F`:

```
F = tf(a,[1 a]);
```

`F` is a `genss` object which has the tunable parameter `a` in its `Blocks` property. You can connect `F` with other tunable or numeric models to create more complex models of control systems. For an example, see “Control System With Tunable Components” on page 2-55.

More About

- “Models with Tunable Coefficients” on page 1-19

Tunable Second-Order Filter

This example shows how to create a parametric model of the second-order filter:

$$F(s) = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2},$$

where the damping ζ and the natural frequency ω_n are tunable parameters.

- 1 Define the tunable parameters using `realp`.

```
wn = realp('wn',3);
zeta = realp('zeta',0.8);
```

`wn` and `zeta` are `realp` parameter objects, with initial values 3 and 0.8, respectively.

- 2 Create a model of the filter using the tunable parameters.

```
F = tf(wn^2,[1 2*zeta*wn wn^2])
```

The inputs to `tf` are the vectors of numerator and denominator coefficients expressed in terms of `wn` and `zeta`.

`F` is a `genss`. The property `F.Blocks` lists the two tunable parameters `wn` and `zeta`.

- 3 (Optional) Examine the number of tunable blocks in the model using `nblocks`.

```
nblocks(F)
```

This command returns the result:

```
ans =
```

```
6
```

`F` has two tunable parameters, but the parameter `wn` appears five times — twice in the numerator and three times in the denominator.

4 (Optional) Rewrite F for fewer occurrences of ω_n .

The second-order filter transfer function can be expressed as follows:

$$F(s) = \frac{1}{\left(\frac{s}{\omega_n}\right)^2 + 2\zeta\left(\frac{s}{\omega_n}\right) + 1}.$$

Use this expression to create the tunable filter:

```
F = tf(1, [(1/wn)^2 2*zeta*(1/wn) 1])
```

5 (Optional) Examine the number of tunable blocks in the new filter model.

```
nblocks(F)
```

This command returns the result:

```
ans =
```

```
4
```

In the new formulation, there are only three occurrences of the tunable parameter ω_n . Reducing the number of occurrences of a block in a model can improve performance time of calculations involving the model. However, the number of occurrences does not affect the results of tuning the model or sampling it for parameter studies.

More About

- “Models with Tunable Coefficients” on page 1-19

State-Space Model With Both Fixed and Tunable Parameters

This example shows how to create a state-space (`genss`) model having both fixed and tunable parameters.

Create a state-space model having the following state-space matrices:

$$A = \begin{bmatrix} 1 & a+b \\ 0 & ab \end{bmatrix}, \quad B = \begin{bmatrix} -3.0 \\ 1.5 \end{bmatrix}, \quad C = [0.3 \ 0], \quad D = 0,$$

where a and b are tunable parameters, whose initial values are -1 and 3 , respectively.

1 Create the tunable parameters using `realp`.

```
a = realp('a', -1);
b = realp('b', 3);
```

2 Define a generalized matrix using algebraic expressions of a and b .

```
A = [1 a+b; 0 a*b]
```

A is a generalized matrix whose `Blocks` property contains a and b . The initial value of A is $M = [1 \ 2; 0 \ -3]$, from the initial values of a and b .

3 Create the fixed-value state-space matrices.

```
B = [-3.0; 1.5];
C = [0.3 0];
D = 0;
```

4 Use `ss` to create the state-space model.

```
sys = ss(A,B,C,D)
```

`sys` is a generalized LTI model (`genss`) with tunable parameters a and b .

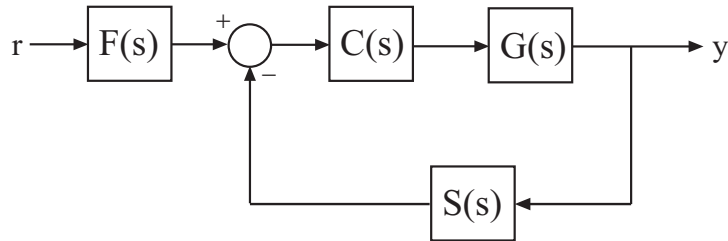
More About

- “Models with Tunable Coefficients” on page 1-19

Control System With Tunable Components

This example shows how to model a control system having a mixture of numeric and tunable components by creating models representing each component and interconnecting them to create a `genss` model.

Create a model of the following control system:



The plant response $G(s) = 1/(s + 1)^2$. The model of sensor dynamics is $S(s) = 5/(s + 4)$. The controller C is a tunable PID controller, and the prefilter $F = a/(s + a)$ is a low-pass filter with one tunable parameter, a .

- 1 Create models representing the plant and sensor dynamics.

Because the plant and sensor dynamics are fixed, represent them using numeric LTI models:

```
G = zpk([], [-1, -1], 1);
S = tf(5, [1 4]);
```

- 2 Create a tunable representation of the controller C using `ltiblock.pid`.

```
C = ltiblock.pid('C', 'PID');
```

`ltiblock.pid` is a control design block with a predefined proportional-integral-derivative (PID) structure. For more information about predefined control design blocks, see “Control Design Blocks” on page 1-7.

- 3 Create a model of the filter F with one tunable parameter.

- a Create the tunable parameter using `realp`:

```
a = realp('a', 10);
```

`a` is a `realp` (real tunable parameter) object with initial value 10.

- b Create the tunable model of the filter using `tf` with the tunable parameter `a`:

```
F = tf(a, [1 a]);
```

F is a `genss` model object.

- 4 Use model interconnection commands to connect the models together to construct a model of the closed-loop response from r to y :

```
T = feedback(G*C,S)*F
```

T is a `genss` model object. In contrast to an aggregate model formed by connecting only numeric LTI models, T keeps track of the tunable elements of the control system. The tunable elements are stored in the `Blocks` property of the `genss` model object. Entering:

```
T.Blocks
```

displays the tunable elements of T:

```
ans =
```

```
  C: [1x1 ltiblock.pid]  
  a: [1x1 realp]
```

More About

- “Models with Tunable Coefficients” on page 1-19

Model Arrays

In this section...

“About Model Arrays” on page 2-58

“One-Dimensional Model Array with Single Parameter Variation” on page 2-61

“Select Models from Array” on page 2-61

“Array With Variations in Two Parameters” on page 2-64

“Sample a Tunable (Parametric) Model for Parameter Studies” on page 2-66

About Model Arrays

What Are Model Arrays?

In many applications, it is useful to consider collections multiple model objects. For example, you may want to consider a model with a parameter that varies across a range of values, such as

```
sys1 = tf(1, [1 1 1]);  
sys2 = tf(1, [1 1 2]);  
sys3 = tf(1, [1 1 3]);
```

and so on. Model arrays are a convenient way to store and analyze such a collection. Model arrays are collections of multiple linear models, stored as elements in a single MATLAB array.

For all models collected in a single model array, the following attributes must be the same:

- The number of inputs and outputs
- The sample time `Ts`
- The time unit `TimeUnit`

Uses of Model Arrays

Uses of model arrays include:

- Representing and analyzing sensitivity to parameter variations
- Validating a controller design against several plant models
- Representing linear models arising from the linearization of a nonlinear system at several operating points
- Storing models obtained from several system identification experiments applied to one plant

Using model arrays, you can apply almost all of the basic model operations that work on single model objects to entire sets of models at once. Functions operate on arrays model by model, allowing you to manipulate an entire collection of models in a vectorized fashion. You can also use analysis functions such as `bode`, `nyquist`, and `step` to model arrays to analyze multiple models simultaneously. You can access the individual models in the collection through MATLAB array indexing.

Visualizing Model Arrays

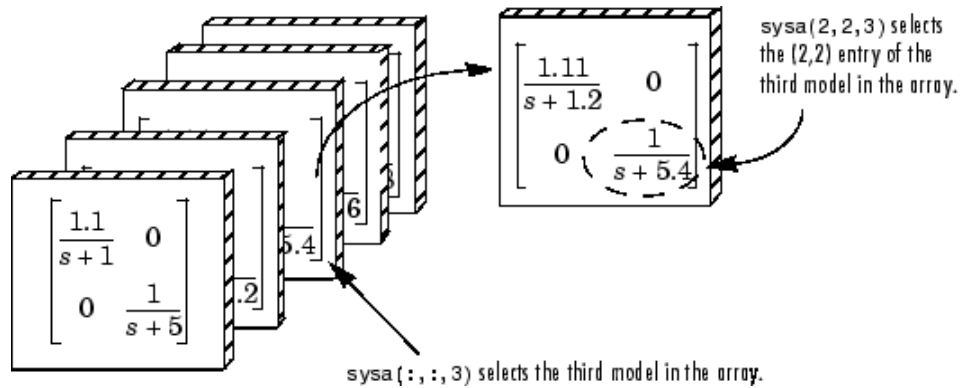
To visualize the concept of a model array, consider the set of five transfer function models shown below. In this example, each model has two inputs and two outputs. They differ by parameter variations in the individual model components.

$$\left[\begin{array}{cc} \frac{1.1}{s+1} & 0 \\ 0 & \frac{1}{s+5} \end{array} \right] \quad \left[\begin{array}{cc} \frac{1.3}{s+1.1} & 0 \\ 0 & \frac{1}{s+5.2} \end{array} \right] \quad \left[\begin{array}{cc} \frac{1.11}{s+1.2} & 0 \\ 0 & \frac{1}{s+5.4} \end{array} \right] \quad \left[\begin{array}{cc} \frac{1.15}{s+1.3} & 0 \\ 0 & \frac{1}{s+5.6} \end{array} \right] \quad \left[\begin{array}{cc} \frac{1.09}{s+1.4} & 0 \\ 0 & \frac{1}{s+5.8} \end{array} \right]$$

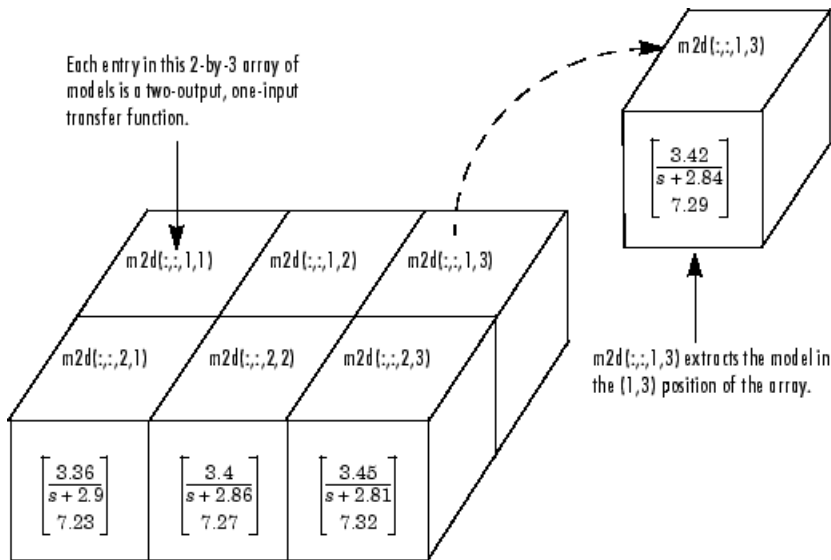
Just as you might collect a set of two-by-two matrices in a multidimensional array, you can collect this set of five transfer function models as a list in a model array under one variable name, say, `sys`. Each element of the model array is a single model object.

Visualizing Selection of Models From Model Arrays

The following illustration shows how indexing selects models from a one-dimensional model array. The illustration shows a 1-by-5 array `sysa` of 2-input, 2-output transfer functions.



The following illustration shows selection of models from the two-dimensional model array `m2d`.



One-Dimensional Model Array with Single Parameter Variation

This example shows how to create a one-dimensional array of transfer functions using `stack`. One parameter of the transfer function varies from model to model in the array.

Create an array of transfer functions representing a low-pass filter

$F(s) = \frac{a}{s+a}$, at three values of the roll-off frequency a :

- 1 Create transfer function models representing the filter with roll-off frequency at $a = 3, 5,$ and 7 .

```
F1 = tf(3,[1 3]);
F2 = tf(5,[1 5]);
F3 = tf(7,[1 7]);
```

- 2 Use `stack` to build an array.

```
Farray = stack(1,F1,F2,F3)
```

The first argument to `stack` specifies the array dimension along which `stack` builds an array. The remaining arguments specify the models to arrange along that dimension.

`Farray` is a 3-by-1 array of transfer functions.

Note Concatenating models with MATLAB array concatenation commands, instead of with `stack`, creates multi-input multi-output (MIMO) models rather than model arrays. For example:

```
G = [F1;F2;F3]
```

creates a one-input, three-output transfer function model, not a 3-by-1 array.

Select Models from Array

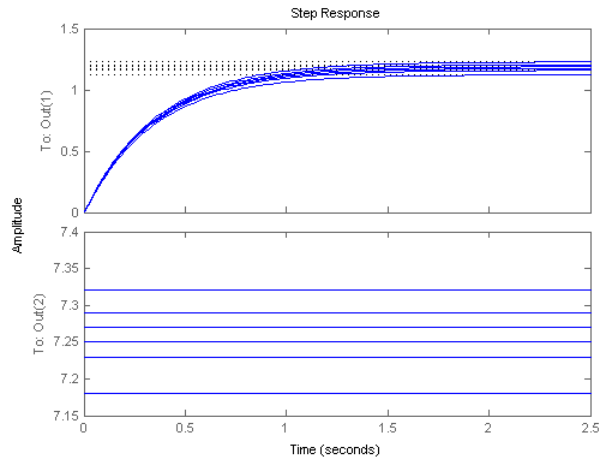
This example shows how to select individual models or sets of models from a model array using array indexing.

- 1 Load the transfer function array `m2d` into the MATLAB workspace.

```
load LTIexamples m2d
```

- 2 (Optional) Plot the step response of `m2d`.

```
step(m2d)
```



The step response shows that `m2d` contains six one-input, two-output models. The `step` command plots all of the models in an array on a single plot.

- 3 (Optional) Examine the dimensions of `m2d`.

```
arraydim = size(m2d)
```

This command produces the result:

```
arraydim =  
  
      2      1      2      3
```

- The first entries of `arraydim`, 2 and 1, show that `m2d` is an array of two-output, one-input transfer functions.

- The remaining entries in `arraydim` give the array dimensions of `m2d`, 2-by-3.

In general, the dimensions of a model array are `[Ny,Nu,S1,...,Sk]`. `Ny` and `Nu` are the numbers of outputs and inputs of each model in the array. `S1,...,Sk` are the array dimensions. Thus, `Si` is the number of models along the *i*th array dimension.

- 4 Select the transfer function in the second row, first column of `m2d`.

To do so, use MATLAB array indexing.

```
sys = m2d(:, :, 2, 1)
```

Tip You can also access models using single index referencing of the array dimensions. For example,

```
sys = m2d(:, :, 4)
```

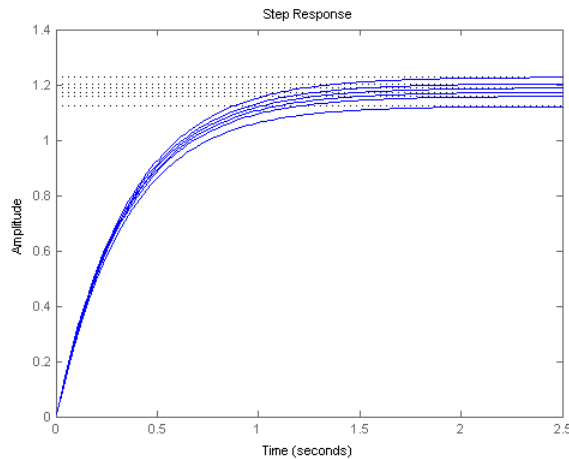
selects the same model as `m2d(:, :, 2, 1)`.

- 5 Select the array of subsystems from the first input to the first output of each model in `m2d`.

```
m11 = m2d(1, 1, :, :)
```

- 6 (Optional) Plot the step response of `m11`.

```
step(m11)
```



The step response shows that `m11` is an array of six single-input, single-output (SISO) models.

Note For frequency response data (FRD) models, the array indices can be followed by the keyword 'frequency' and some expression selecting a subset of the frequency points as in

```
sys (outputs,inputs,n1,...,nk,'frequency',SelectedFreqs)
```

See “Referencing FRD Models Through Frequencies” on page 5-11 for details on frequency point selection in FRD models.

More About

- “Visualizing Selection of Models From Model Arrays” on page 2-60

Array With Variations in Two Parameters

This example shows how to create a two-dimensional (2-D) array of transfer functions using for loops. One parameter of the transfer function varies in each dimension of the array. You can use the technique of this example to create higher-dimensional arrays with variations of more parameters.

The second-order single-input, single-output (SISO) transfer function

$$H(s) = \frac{\omega^2}{s^2 + 2\zeta\omega s + \omega^2}.$$

depends on two parameters—the damping ratio ζ and natural frequency ω .

If both ζ and ω vary, you obtain multiple transfer functions of the form:

$$H_{ij}(s) = \frac{\omega_j^2}{s^2 + 2\zeta_i\omega_j s + \omega_j^2},$$

where ζ_i and ω_j represent different measurements or sampled values of the variable parameters.

You can collect all of these transfer functions in a single variable to create a two-dimensional model array.

1 (Optional) Preallocate memory for the model array.

You can enhance computation efficiency by preallocating memory. To do this, create an array of the required size and initialize its entries to zero.

```
H = tf(zeros(1,1,3,3));
```

In this example, there are three values for each parameter in the transfer function H . Therefore, this command creates a 3-by-3 array of single-input, single-output (SISO) zero transfer functions.

2 Create arrays containing the parameter values.

```
zeta = [0.66,0.71,0.75];
w = [1.0,1.2,1.5];
```

3 Build the array by looping through all combinations of parameter values.

```
for i = 1:length(zeta)
    for j = 1:length(w)
        H(:, :, i, j) = tf(w(j)^2, [1 2*zeta(i)*w(j) w(j)^2]);
```

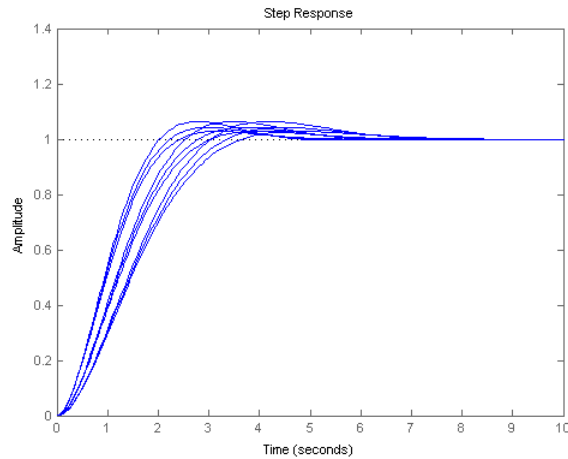
```

end
end

```

H is a 3-by-3 array of transfer functions. ζ varies as you move from model to model along a single column of H. The parameter ω varies as you move along a single row. Plotting the step response of H shows how the parameter variation affects the step response.

```
step(H)
```



Sample a Tunable (Parametric) Model for Parameter Studies

This example shows how to sample a parametric model of a second-order filter across a grid of parameter values using `replaceBlock`.

- 1 Create a tunable (parametric) model of the second-order filter:

$$F(s) = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2},$$

where the damping ζ and the natural frequency ω_n are the parameters.


```
wn = realp('wn',3);  
zeta = realp('zeta',0.8);  
F = tf(wn^2,[1 2*zeta*wn wn^2]);
```

F is a genss model with two tunable Control Design Blocks, the `realp` blocks `wn` and `zeta`. The blocks `wn` and `zeta` have initial values of 3 and 0.8, respectively.

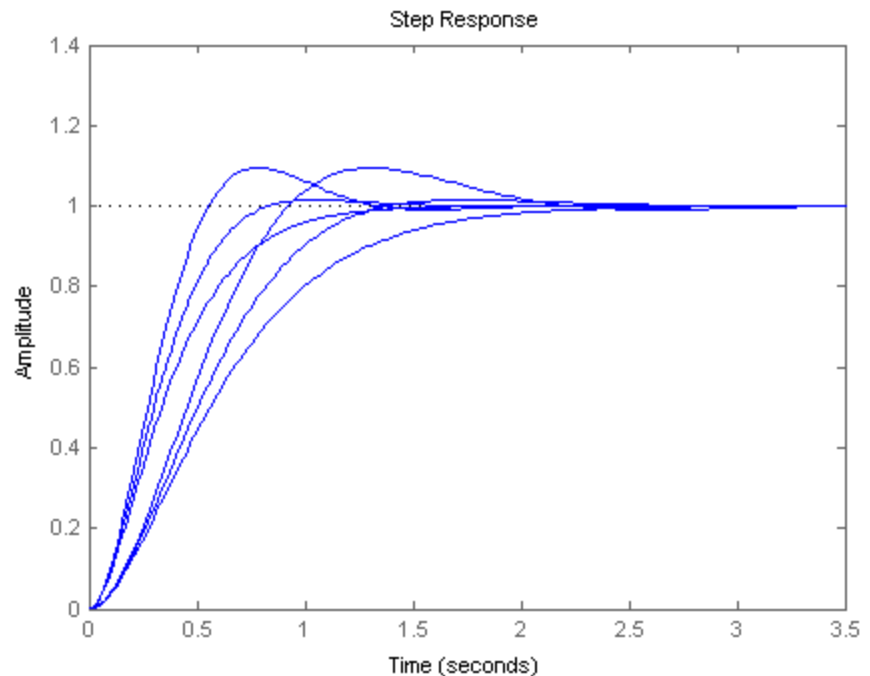
- 2** Sample F over a 2-by-3 grid of (wn,zeta) values.

```
Fsample = replaceBlock(F,'wn',[3;5],'zeta',[0.6 0.8 1.0]);
```

Fsample is 2-by-3 array of state-space models.

- 3** (Optional) Plot the step response of Fsample.

```
step(Fsample)
```



The step response plot show the variation in the natural frequency and damping constant across the six models in the array `Fsample`.

More About

- “Models with Tunable Coefficients” on page 1-19

References

[1] P. Gahinet and L.F. Shampine, "Software for Modeling and Analysis of Linear Systems with Delays," *Proc. American Control Conf.*, Boston, 2004, pp. 5600-5605

[2] L.F. Shampine and P. Gahinet, Delay-differential-algebraic Equations in Control Theory, *Applied Numerical Mathematics*, 56 (2006), pp. 574-588

Working with Linear Models

- Chapter 3, “Data Manipulation”
- Chapter 4, “Model Interconnections”
- Chapter 5, “Operations on Models”
- Chapter 6, “Model Analysis Tools”

Data Manipulation

- “Model Properties” on page 3-2
- “Extract Model Coefficients” on page 3-5
- “Attach Metadata to Models” on page 3-9
- “Query Model Characteristics” on page 3-14
- “Customize Model Display” on page 3-17

Model Properties

| In this section... |
|---|
| “About Model Properties” on page 3-2 |
| “Specify Model Properties at Model Creation” on page 3-2 |
| “Examine and Specify Properties of an Existing Model” on page 3-3 |

About Model Properties

Model properties are data fields for specifying model attributes, such as coefficients, sampling time, and metadata such as channel names.

For a list of all properties associated with each model type, see the corresponding reference page.

For help on any particular model property, see the model reference page or enter help *modeltype.propertyname* at the command line. For example:

```
help ss.StateName
```

Related Examples

- “Examine and Specify Properties of an Existing Model” on page 3-3
- “Specify Model Properties at Model Creation” on page 3-2

Specify Model Properties at Model Creation

This example shows how to specify transfer function model properties when you create the model. You can use the techniques of this example to access and set properties of any type of model.

To specify model properties at model creation, use *Name, Value* pair syntax. For example, assign a transport delay and an output name to a new transfer function model.

```
H = tf(1,[1 10], 'ioDelay',6.5, 'OutputName', 'velocity')
```


Examine and Specify Properties of an Existing Model

This example shows how to examine and specify properties of an existing ss model using `get`, `set`, and dot notation. You can use the techniques of this example to access and set properties of any type of model.

- 1 Load an existing model object.

```
load PadeApproximation1 sys;
```

- 2 Examine the current property values.

```
get(sys)
```

This command displays all the properties of `sys`.

```
    a: [2x2 double]
    b: [2x1 double]
    c: [0.5000 0.1000]
    d: 0
    e: []
    Scaled: 0
    StateName: {2x1 cell}
    StateUnit: {2x1 cell}
InternalDelay: 3.4000
    InputDelay: 0
    OutputDelay: 1.5000
    Ts: 0
    TimeUnit: 'seconds'
    InputName: {''}
    InputUnit: {''}
    InputGroup: [1x1 struct]
    OutputName: {''}
    OutputUnit: {''}
    OutputGroup: [1x1 struct]
    Name: ''
    Notes: {}
    UserData: []
```

- 3 Specify input delay and channel names.

Using dot notation allows you to directly reference property values.

```
sys.InputDelay = 4.2;  
sys.InputName = 'thrust';  
sys.OutputName = 'velocity';
```

Alternatively, you can use the `set` command to change property values.

```
set(sys, 'InputDelay', 4.2, 'InputName', 'thrust', 'OutputName', 'velocity')
```

Caution Changing properties such as `Ts` and `TimeUnits` can cause undesirable changes in system behavior. See the property descriptions in the model reference pages for more information.

Extract Model Coefficients

| In this section... |
|---|
| “Functions for Extracting Model Coefficients” on page 3-5 |
| “Extracting Coefficients of Different Model Type” on page 3-5 |
| “Extract Numeric Model Data and Time Delay” on page 3-6 |
| “Extract Proportional-Integral-Derivative (PID) Gains from Transfer Function” on page 3-7 |

Functions for Extracting Model Coefficients

Control System Toolbox includes several commands for extracting model coefficients such as transfer function numerator and denominator coefficients, state-space matrices, and proportional-integral-derivative (PID) gains.

The following commands are available for data extraction.

| Command | Result |
|------------|---|
| tfdata | Extract transfer function coefficients |
| zpkdata | Extract zero and pole locations and system gain |
| ssdata | Extract state-space matrices |
| dssdata | Extract descriptor state-space matrices |
| frdata | Extract frequency response data from frd model |
| piddata | Extract parallel-form proportional-integral-derivative (PID) data |
| pidstddata | Extract standard-form PID data |
| get | Access all model property values |

Extracting Coefficients of Different Model Type

When you use a data extraction command on a model of a different type, the software computes the coefficients of the target model type. For example, if

you use `zpkdata` on a `ss` model, the software converts the model to `zpk` form and returns the zero and pole locations.

Extract Numeric Model Data and Time Delay

This example shows how to extract transfer function numerator and denominator coefficients using `tfdata`.

- 1 Create a first-order plus dead time transfer function model.

```
H = exp(-2.5*s)/(s+12);
```

- 2 Extract the numerator and denominator coefficients.

```
[num,den] = tfdata(H,'v')
```

`num` and `den` are numerical arrays. Without the `'v'` flag, `tfdata` returns cell arrays.

Note For SISO transfer function models, you can also extract coefficients using:

```
num = H.num{1};  
den = H.den{1};
```

- 3 Extract the time delay.

- a Determine which property of `H` contains the time delay.

In a SISO `tf` model, a time delay can be expressed as an input delay, an output delay, or a transport delay (I/O delay).

```
get(H)
```

This command returns:

```
num: {[0 1]}  
den: {[1 12]}  
Variable: 's'
```

```

        ioDelay: 0
        InputDelay: 0
        OutputDelay: 2.5000
        Ts: 0
        TimeUnit: 'seconds'
        InputName: {''}
        InputUnit: {''}
        InputGroup: [1x1 struct]
        OutputName: {''}
        OutputUnit: {''}
        OutputGroup: [1x1 struct]
        Name: ''
        Notes: {}
        UserData: []

```

The time delay is stored in the `OutputDelay` property.

- b** Extract the output delay.

```
delay = H.OutputDelay;
```

Extract Proportional-Integral-Derivative (PID) Gains from Transfer Function

This example shows how to extract PID gains from a transfer function using `piddata`.

You can use the same steps to extract PID gains from a model of any type that represents a PID controller, using `piddata` or `pidstddata`.

- 1** Create a transfer function that represents a PID controller with a first-order filter on the derivative term.

```
Czpk = zpk([-6.6, -0.7], [0, -2], 0.2)
```

- 2** Obtain the PID gains and filter constant.

```
[Kp, Ki, Kd, Tf] = piddata(Czpk)
```

This command returns the proportional gain K_p , integral gain K_i , derivative gain K_d , and derivative filter time constant T_f . Because `piddata`

automatically computes the PID controller parameters, you can extract the PID coefficients without creating a `pid` model.

More About

- “PID Controllers” on page 1-16

Attach Metadata to Models

In this section...

- “Specify Model Time Units” on page 3-9
- “Interconnect Models with Different Time Units” on page 3-10
- “Specify Frequency Units of Frequency-Response Data Model” on page 3-10
- “Extract Subsystems of Multi-Input, Multi-Output (MIMO) Models” on page 3-11
- “Specify and Select Input and Output Groups” on page 3-12

Specify Model Time Units

This example shows how to specify time units of a transfer function model.

The `TimeUnit` property of the `tf` model object specifies units of the time variable, time delays (for continuous-time models), and the sampling time `Ts` (for discrete-time models). The default time units is seconds.

Create a SISO transfer function model $sys = \frac{4s + 2}{s^2 + 3s + 10}$ with time units in milliseconds:

```
num = [4 2];
den = [1 3 10];
sys = tf(num,den,'TimeUnit','milliseconds');
```

You can specify the time units of any dynamic system in a similar way.

The system time units appear on the time- and frequency-domain plots. For multiple systems with different time units, the units of the first system are used if the time and frequency units in the “Toolbox Preferences Editor” on page 8-2 are auto.

Note Changing the `TimeUnit` property changes the system behavior. If you want to use different time units without modifying system behavior, use `chgTimeUnit`.

Interconnect Models with Different Time Units

This example shows how to interconnect transfer function models with different time units.

To interconnect models using arithmetic operations or interconnection commands, the time units of all models must match.

- 1 Create two transfer function models with time units of milliseconds and seconds, respectively.

```
sys1 = tf([1 2],[1 2 3],'TimeUnit','milliseconds');  
sys2 = tf([4 2],[1 3 10]);
```

- 2 Change the time units of `sys2` to milliseconds.

```
sys2 = chgTimeUnit(sys2,'milliseconds');
```

- 3 Connect the systems in parallel.

```
sys = sys1+sys2;
```

Specify Frequency Units of Frequency-Response Data Model

This example shows how to specify units of the frequency points of a frequency-response data model.

The `FrequencyUnit` property specifies units of the frequency vector in the `Frequency` property of the `frd` model object. The default frequency units are `rad/TimeUnit`, where `TimeUnit` is the time unit specified in the `TimeUnit` property.

Create a SISO frequency-response data model with frequency data in GHz.

```
load AnalyzerData;
```



```
sys = frd(resp,freq,'FrequencyUnit','GHz');
```

You can independently specify the units in which you measure the frequency points and sample time in the `FrequencyUnit` and `TimeUnit` properties, respectively. You can also specify the frequency units of a `genfrd` in a similar way.

The frequency units appear on the frequency-domain plots. For multiple systems with different frequency units, the units of the first system are used if the frequency units in the “Toolbox Preferences Editor” on page 8-2 is `auto`.

Note Changing the `FrequencyUnit` property changes the system behavior. If you want to use different frequency units without modifying system behavior, use `chgFreqUnit`.

Extract Subsystems of Multi-Input, Multi-Output (MIMO) Models

This example shows how to extract subsystems of a MIMO model using channel names.

Extracting subsystems is useful when, for example, you want to analyze a portion of a complex system. Using channel names, you can use MATLAB indexing to extract all the dynamics relating to a particular channel, without keeping track of channel order in a complex MIMO model.

1 Create a MIMO transfer function.

```
G1 = tf(3,[1 10]);  
G2 = tf([1 2],[1 0]);  
G = [G1,G2];
```

2 Assign names to the model inputs.

```
G.InputName = {'temperature';'pressure'};
```

Because `G` has two inputs, use a cell array of two strings to specify the channel names.

- 3** Extract the subsystem of `G` that contains all dynamics from the 'temperature' input to all outputs.

```
Gt = G(:, 'temperature');
```

Specify and Select Input and Output Groups

This example shows how to specify groups of input and output channels in a model object and extract subsystems using the groups.

Channel groups are useful for keeping track of inputs and outputs in complex MIMO models.

- 1** Create a state-space model with three inputs and four outputs.

```
H = rss(3,4,3);
```

- 2** Group the inputs as follows:

- Inputs 1 and 2 in a group named `controls`
- Outputs 1 and 3 to a group named `temperature`
- Outputs 1, 3, and 4 to a group named `measurements`

```
H.InputGroup.controls = [1 2];  
H.OutputGroup.temperature = [1 3];  
H.OutputGroup.measurements = [1 3 4];
```

`InputGroup` and `OutputGroup` are structures. The name of each field in the structure is the name of the input or output group. The value of each field is a vector that identifies the channels in that group.

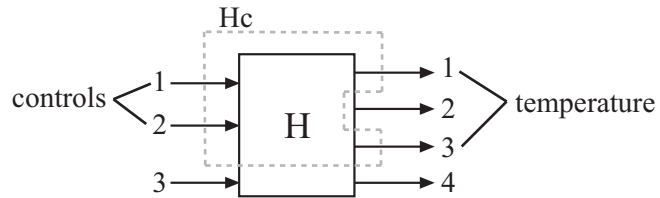
- 3** Extract the subsystem corresponding to the `controls` inputs and the `temperature` outputs.

You can use group names to index into subsystems.

```
Hc = H('temperature', 'controls')
```

`Hc` is a two-input, three-output ss model containing the I/O channels from the 'controls' input the 'temperature' outputs.

The following illustration shows the relationship between H and the subsystem Hc.



Query Model Characteristics

| In this section... |
|-------------------------------------|
| “Query Model Dynamics” on page 3-14 |
| “Query Array Size” on page 3-15 |

How to query model characteristics such as stability, time domain, and number of inputs and outputs.

Query Model Dynamics

This example shows how to query model dynamics such as stability, time domain, and model order.

- 1 Load the saved state-space model T.

```
load queryexample.mat T;
```

- 2 Query whether T has stable dynamics.

```
isstable(T)
```

`isstable` returns a Boolean value of 1 (true) if all system poles are in the open left-half plane (for continuous-time models) or inside the open unit disk (for discrete-time models). Otherwise, `isstable` returns 0 (false).

- 3 Query whether T has any time delays.

```
hasdelay(T)
```

This command returns 1, which indicates that T has input delay, output delay, or internal delay. Use `get(T)` to determine which property of T holds the time delay.

- 4 Query whether T is proper.

```
isproper(T)
```

This command returns 1 if the system has relative degree ≤ 0 .

5 Query the order of T.

```
order(T)
```

For a state-space model, `order` returns the number of states. For a `tf` or `zpk` model, the order is the number of states required for a state-space realization of the system.

6 Query whether T is a discrete-time model.

```
isdt(T)
```

This command returns 1, indicating that T is a discrete-time model. Similarly, you can use `isct` to query whether T is in continuous time.

Query Array Size

This example shows how to query the size of model arrays, including the number of inputs and outputs and the number of models in an array.

You can also use these commands on individual models.

1 Load a saved model array, `sysarr`.

```
load queryexample sysarr
```

2 Query the array dimensions.

```
size(sysarr)
```

This command displays the array size, and the number of inputs, outputs, and states of the models in the array.

Alternatively, to obtain the array dimensions as a numeric array, use `size` with an output argument.

```
dims = size(sysarr)
```

This command produces the result:

```
dims =
     3     1     2     4
```

The first two entries 3, 1 are the number of outputs and inputs, respectively. The remaining entries 2, 4 are the array dimensions.

Tip To query the number of array dimensions, use `ndims`.

- 3** Query the number of models in the array.

```
N = nmodels(sysarr)
```

Because `sysarr` is a 2-by-4 array, this command returns a result of $2 \times 4 = 8$.

- 4** Query whether the models in `sysarr` are single-input, single-output (SISO).

```
issiso(sysarr)
```

`issiso` returns a Boolean value of 1 (true) if the models in the array are SISO, and 0 (false) if the models are not SISO.

- 5** Query the number of inputs and outputs in the models in the array using `iosize`.

```
ios = iosize(sysarr)
```

This command returns a numeric array containing the number of outputs and inputs of the models in the array.

Related Examples

- “Select Models from Array” on page 2-61

Customize Model Display

How to customize the MATLAB screen display of transfer function models.

In this section...

“Configure Transfer Function Display Variable” on page 3-17

“Configure Display Format of Transfer Function in Factorized Form” on page 3-18

Configure Transfer Function Display Variable

This example shows how to configure the display variable of transfer function (tf) models.

You can use the same steps to configure the display variable of transfer function models in factorized form (zpk models).

By default, tf and zpk models are displayed in terms of s in continuous time and z in discrete time. Use the `Variable` property change the display variable to 'p' (equivalent to 's'), 'q' (equivalent to 'z'), 'z⁻¹', or 'q⁻¹'.

- 1 Create the discrete-time transfer function $H(z) = \frac{z-1}{z^2-3z+2}$ with a sampling time of 1 s.

```
H = tf([1 -1],[1 -3 2],0.1)
```

- 2 Change the display variable to 'q⁻¹'.

```
H.Variable = 'q^-1'
```

MATLAB computes new coefficients and displays the same transfer function in terms of the new variable:

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

```
Sampling time (seconds): 0.1
```

Tip Alternatively, you can directly create the same transfer function expressed in terms of 'q⁻¹'.

```
H = tf([0 1 -1],[1 -3 2],0.1,'Variable','q^-1');
```

For the inverse variables 'z⁻¹' and 'q⁻¹', `tf` interprets the numerator and denominator arrays as coefficients of ascending powers of 'z⁻¹' or 'q⁻¹'.

Configure Display Format of Transfer Function in Factorized Form

This example shows how to configure the display of transfer function models in factorized form (zpk models).

You can configure the display of the numerator and denominator polynomials to highlight:

- The numerator and denominator roots
- The natural frequencies and damping ratios of each root
- The time constants and damping ratios of each root

See the `DisplayFormat` property on the zpk reference page for more information about these quantities.

- 1 Create a zpk model having a zero at $s = 5$, a pole at $s = -10$, and a pair of complex poles at $s = -2 \pm 5i$.

```
H = zpk(5,[-10,-3-5*i,-3+5*i],10)
```

This command displays the result:

```
Zero/pole/gain:
      10 (s-5)
-----
(s+10) (s^2 + 6s + 34)
```


The default display format, 'roots', displays the standard factorization of the numerator and denominator polynomials.

- 2 Configure the display format to display the natural frequency of each polynomial root.

Set the DisplayFormat property of H to 'frequency'.

```
H.DisplayFormat = 'frequency'
```

This command displays the result:

```
Zero/pole/gain:
      -0.14706 (1-s/5)
-----
(1+s/10) (1 + 1.029(s/5.831) + (s/5.831)^2)
```

You can read the natural frequencies and damping ratios for each pole and zero from the display as follows:

- Factors corresponding to real roots are displayed as $(1 - s/\omega_0)$, where ω_0 is the natural frequency of the root. Thus, for example, the natural frequency of the zero of H is 5.
 - Factors corresponding to complex pairs of roots are displayed as $1 - 2\zeta(s/\omega_0) + (s/\omega_0)^2$, where ω_0 is the natural frequency and ζ is the damping ratio of the root. Thus, for example, the natural frequency of the complex pole pair is 5.831, and the damping ratio is 1.029/2.
- 3 Configure the display format to display the time constant of each pole and zero.

To do this, set the DisplayFormat property of H to 'time constant'.

```
H.DisplayFormat = 'time constant'
```

This command displays the result:

```
      -0.14706 (1-0.2s)
-----
(1+0.1s) (1 + 1.029(0.1715s) + (0.1715s)^2)
```

You can read the time constants and damping ratios from the display as follows:

- Factors corresponding to real roots are displayed as (τs) , where τ is the time constant of the root. Thus, for example, the time constant of the zero of H is 0.2.
- Factors corresponding to complex pairs of roots are displayed as $1 - 2\zeta(\tau s) + (\tau s)^2$, where τ is the time constant and ζ is the damping ratio of the root. Thus, for example, the time constant of the complex pole pair is 0.1715, and the damping ratio is 1.029/2.

Model Interconnections

- “About Model Interconnections” on page 4-2
- “Catalog of Model Interconnections” on page 4-3
- “Build a Model of a Single-Input, Single-Output (SISO) Feedback Loop” on page 4-5
- “Build a Model of a Multi-Input, Multi-Output (MIMO) Feedback Loop” on page 4-8
- “Build a Model of a Multi-Loop Control System” on page 4-9
- “Build a Model of a Multi-Input, Multi-Output (MIMO) Control System” on page 4-12
- “Results of Connecting Models” on page 4-16
- “Recommended Model Type for Building Block Diagrams” on page 4-18

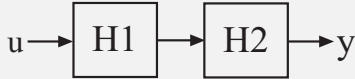
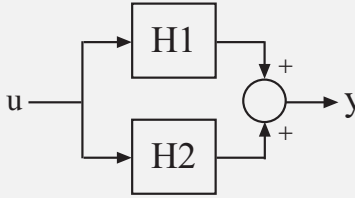
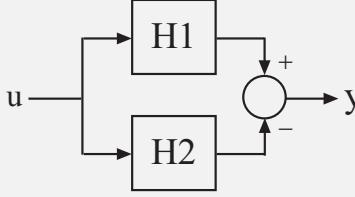
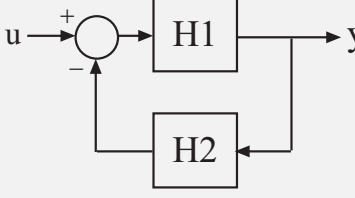

About Model Interconnections

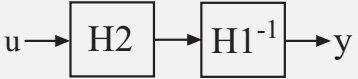
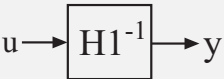
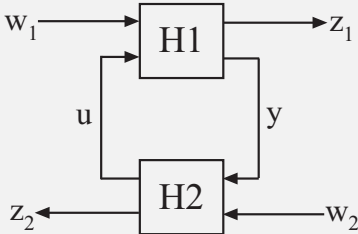
You can use model arithmetic and model interconnection commands to construct models of control systems. You can conceptualize your control system as a block diagram containing multiple interconnected components. You use model arithmetic or interconnection commands to combine models of each component into a single model representing the entire block diagram. For example, you can construct a single model that represents an entire closed-loop control system having a plant, a controller, sensor dynamics, or other components.

Model interconnection commands also let you combine Numeric LTI models with Control Design Blocks to create Generalized LTI models that represent systems with tunable components.

Catalog of Model Interconnections

Each of the model interconnection commands corresponds to a type of connection in a block diagram. Many of the commands also correspond to arithmetic combinations of models. The following table lists the interconnection commands, the corresponding algebraic expression, and the corresponding block diagram connection.

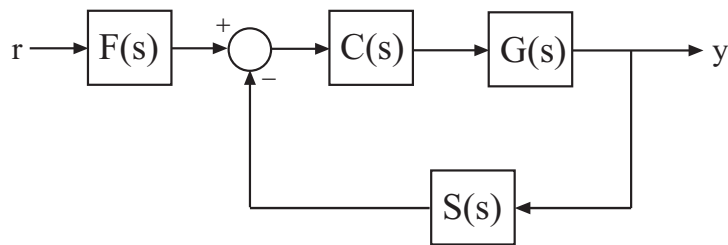
| Command | Arithmetic Equivalent | Block Diagram Connection |
|-------------------------------|-----------------------|--|
| <code>series(H1,H2)</code> | $H2 * H1$ |  |
| <code>parallel(H1,H2)</code> | $H1 + H2$ |  |
| <code>parallel(H1,-H2)</code> | $H1 - H2$ |  |
| <code>feedback(H1,H2)</code> | $H1 / (1 + H2 * H1)$ |  |
| N/A | $H1 / H2$ (division) |  |

| Command | Arithmetic Equivalent | Block Diagram Connection |
|---|-----------------------------------|--|
| N/A | $H1 \setminus H2$ (left division) |  |
| inv(H1) | N/A |  |
| lft(H1, H2, nu, ny) — Linear fractional transformation (LFT) | N/A |  |

Build a Model of a Single-Input, Single-Output (SISO) Feedback Loop

This example shows how to use arithmetic and model connection commands to combine models representing multiple system components into a single model representing the entire closed-loop SISO control system.

Construct a model of the following control system, which contains a plant $G(s)$, a controller $C(s)$, and a representation of sensor dynamics, $S(s)$, in a single feedback loop. The system also includes a prefilter $F(s)$.



1 Create model objects representing each of the components.

```

G = zpk([], [-1, -1], 1);
C = pid(2, 1.3, 0.3, 0.5);
S = tf(5, [1 4]);
F = tf(1, [1 1]);
  
```

In this example, the plant G is a zero-pole-gain (zpk) model with a double pole at $s = -1$. C is a PID controller, and F and S are transfer functions. You can use model arithmetic and interconnection commands on any type of dynamic system model or static model.

2 Construct the combined controller-plant system $L(s) = C(s)G(s)$.

```
L = G*C;
```

When you combine models using the multiplication operator $*$, the models are in reverse order compared to the block diagram.

Tip Alternatively, construct $L(s)$ using the `series` command:

```
L = series(C,G);
```

- 3** Construct the unfiltered closed-loop response $T(s) = \frac{L}{1+LS}$.

You can use model arithmetic to construct T algebraically:

```
T = L / (1+S*L)
```

However, arithmetically computing this model duplicates the poles of L . This duplication inflates the model order and can lead to computational inaccuracy.

Instead, use `feedback` to compute the closed-loop model.

```
T = feedback(ss(L),S);
```

Converting L to state-space (`ss`) causes `feedback` to convert all components to `ss` models and return T as a `ss` model. Converting to `ss` is not required, but for complex interconnections with many components, converting to state-space form yields better numeric accuracy. See “Recommended Model Type for Building Block Diagrams” on page 4-18 for more information.

- 4** Combine T with the prefilter transfer function to obtain the entire closed-loop system response from r to y .

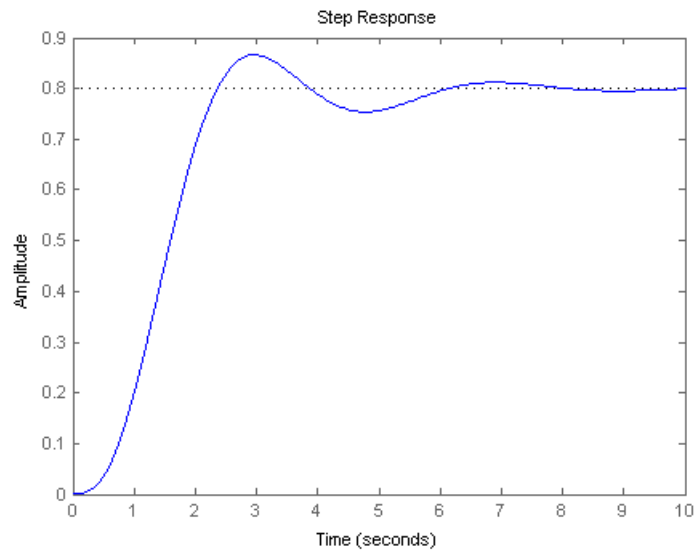
```
Try = T*F;
```

`Try` is a `ss` object representing the aggregate closed-loop system. `Try` does not retain the original data from the combined components G , C , F , and S .

- 5** (Optional) Plot the step response of the closed-loop system.

When you build models by combining components, you can operate on those models with Control System Toolbox control design and analysis commands

```
step(Try)
```

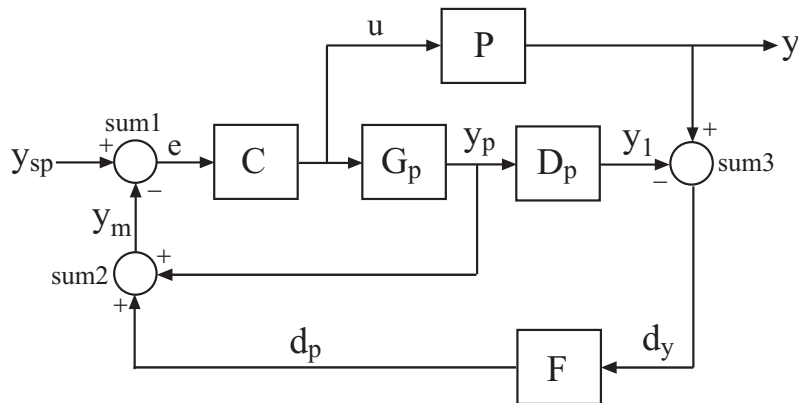



Build a Model of a Multi-Input, Multi-Output (MIMO) Feedback Loop

This example shows how to build a MIMO closed-loop model using feedback.

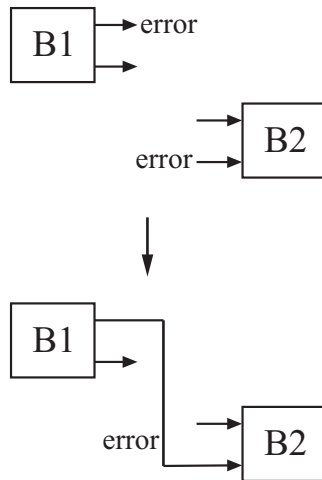
Build a Model of a Multi-Loop Control System

This example shows how to combine models to construct an arbitrary block diagram using `connect`. The system of this example is a Smith Predictor, the single-input, single-output (SISO) multi-loop control system illustrated in the following block diagram.



For more information about the Smith Predictor system, see the Control System Toolbox demo `Control of Processes with Long Dead Time: The Smith Predictor`.

The `connect` command lets you construct the complex interconnections of this system. To use `connect`, you name the input and output channels of the components of the block diagram. `connect` joins ports that have the same name. For example, the following illustration shows how `connect` joins an output of model B1 named `error` with an input of model B2 that has the same name.



To build the closed loop model of the Smith Predictor system from y_{sp} to y :

- 1 Create model objects representing the process model P , the predictor model G_p , and the delay model D_p .

```

s = tf('s');

P = exp(-93.9*s) * 5.6/(40.2*s+1);
P.InputName = 'u'; P.OutputName = 'y';

Gp = 5.6/(40.2*s+1);
Gp.InputName = 'u'; Gp.OutputName = 'yp';

Dp = exp(-93.9*s);
Dp.InputName = 'yp'; Dp.OutputName = 'y1';

```

Use the `InputName` and `OutputName` properties to specify signal names for the input and output of each model. When you connect these models to build the control system, these names tell connect how the models are interconnected in your block diagram. Therefore, for example, because P and G_p receive the same input signal u in the block diagram, assign them the same `InputName`, u . Likewise, because the output of G_p is the input of the delay D_p , set `Gp.OutputName` and `Dp.InputName` to yp .

2 Create models representing the filter F and the PI controller C.

```
F = 1/(20*s+1);  
F.InputName = 'dy'; F.OutputName = 'dp';  
  
C = pidstd(0.574,40.1);  
C.Inputname = 'e'; C.OutputName = 'u';
```

These models also have specified input and output names to enable connect to combine the models.

3 Create the summing junctions needed to complete the block diagram.

Use the `sومblk` command to create each summing junction.

```
sum1 = سومblk('e = ysp - ym');  
sum2 = سومblk('ym = yp + dp');  
sum3 = سومblk('dy = y - y1');
```

The argument to `sومblk` is a formula string that specifies the output signal and summed input signals. For example, in `sum1`, the output is the signal `e`, which is the difference `ysp-ym`.

4 Assemble the complete model from y_{sp} to y .

```
T = connect(P,Gp,Dp,C,F,sum1,sum2,sum3,'ysp','y');
```

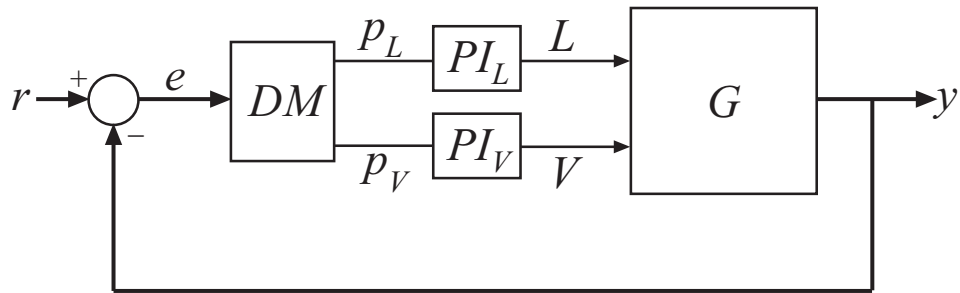
To use `connect`, list all the models and summing junctions in any order. `connect` automatically combines the models and summing junctions by matching each argument's output name with another argument's input name. The final two arguments to `connect` specify the input and output signals of the aggregate model.

Thus, the resulting model `T` is a `ss` model with input `ysp` and output `y`. `T` represents the aggregate multi-loop control structure.

Build a Model of a Multi-Input, Multi-Output (MIMO) Control System

This example shows how to construct a model of a MIMO control system using connect.

Consider the following two-input, two-output control system.



The plant G represents a distillation column with two inputs, reflux L and boilup V , and two outputs, the concentrations of two chemicals, represented by the vector signal $y = [y_D, y_B]$. The vector setpoint signal $r = [r_D, r_B]$ represents the desired concentrations of the two chemicals. The vector error signal e is input to DM , a static 2-by-2 decoupling matrix. PI_L and PI_V are independent PI controllers that control the two inputs of G . For more information about this system, see the Robust Control Toolbox demo Decoupling Controller for a Distillation Column.

To create a two-input, two-output model representing this closed-loop control system:

- 1 Create a model representing the 2-by-2 plant G .

```
s = tf('s', 'TimeUnit', 'minutes');
G = [87.8 -86.4 ; 108.2 -109.6]/(75*s+1);
G.InputName = {'L', 'V'};
G.OutputName = {'yD', 'yB'};
```

When you construct the closed-loop model, connect uses the input and output names to form connections between the block diagram components.

Therefore, assign names to the inputs and outputs of the transfer function G .

2 Create a model representing the 2-by-2 decoupling matrix DM .

```
DM = tf([4.31, -2.51; 3.92, -2.97], 'TimeUnit', 'minutes');
DM.u = 'e';
DM.y = {'pL', 'pV'};
```

$DM.u$ and $DM.y$ are shorthand for $DM.InputName$ and $DM.OutputName$, respectively. Therefore, the above commands assign the names 'pL' and 'pV' to the outputs of DM .

Additionally, when you use a single string to name a vector-valued input or output, the name is automatically expanded into a vector of the necessary length. Therefore, the command $DM.u = 'e'$ assigns the names 'e(1)' and 'e(2)' to the inputs of DM .

3 Create models representing the PI controllers PI_L and PI_V .

```
PI_L = pid(1, 0.0361, 'TimeUnit', 'minutes');
PI_L.u = 'pL'; PI_L.y = 'L';

PI_V = pid(1, 0.0365, 'TimeUnit', 'minutes');
PI_V.u = 'pV'; PI_V.y = 'V';
```

PI_L and PI_V are `pid` controller objects. The input and output signals of each controller are named as in the block diagram.

Tip To create a tunable (`genss`) model of the control system, use `ltiblock.gain` to represent DM and `ltiblock.pid` to represent PI_L and PI_V . See the Robust Control Toolbox demo Decoupling Controller for a Distillation Column for an example.

4 Create the summing junction.

The summing junction produces the error signals e by taking the difference between r and y . All three signals are vector signals with two values.

```
Sum = sumblk('e = %r - %y', {'rD', 'rB'}, G.y);
```

Sum is the transfer function for the summing junction described by the formula string 'e = %r - %y'. The alias entries %r and %y in the formula refer to subsequent arguments to sumblk. Thus, sumblk uses {'rD', 'rB'} for %r, and the signal names of G.OutputName for %y. Because %r and %y are vector signals of length 2, sumblk automatically expands e into the vector signals {'e(1)', 'e(2)'}

Therefore, Sum is the vector summing junction:

$$e(1) = r_D - y_D$$
$$e(2) = r_B - y_B.$$

- 5** Join all components to construct the closed-loop system from r to y .

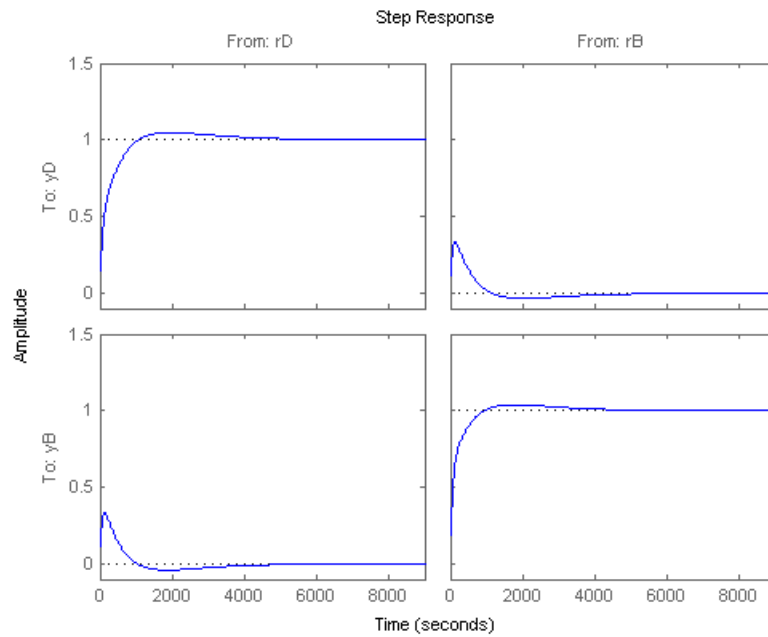
```
CLry = connect(DM,G,PI_L,PI_V,Sum,{'rD','rB'},G.y);
```

The arguments to connect include all the components of the closed-loop system, in any order. connect automatically combines the components using the input and output names to join signals.

The final two arguments to connect specify the output and input signals of the closed-loop model, respectively. The inputs are the vector setpoint signals $r = [r_D, r_B]$. The outputs are the outputs $y = [y_D, y_B]$ of the plant G . Therefore, CLry is a two-input, two-output model.

- 6** (Optional) Plot the step response of CLry.

```
step(CLry)
```

The step response plot confirms that CLry has the expected inputs and outputs.

Results of Connecting Models

In this section...

“Property Inheritance in Connected Models” on page 4-16

“Result When Connecting Different Model Types” on page 4-16

Property Inheritance in Connected Models

When you connect two or more models, the resulting model’s properties are determined in part by the operation and in part by the properties of the connected models. Some general rules governing property inheritance are:

- In operations combining discrete-time models, all models must have identical or unspecified (`sys.Ts = -1`) sampling times. Models resulting from such operations inherit the specified sampling time, if there is one.
- All models must have identical `TimeUnit` properties.
- In general, when you connect two dynamic systems `sys1` and `sys2` using operations such as `+`, `*`, `[,]`, `[;]`, `append`, and `feedback`, the resulting model inherits its I/O names and I/O groups from `sys1` and `sys2`. However, conflicting I/O names or I/O groups are not inherited. For example, the `InputName` property for `sys1 + sys2` is left unspecified if `sys1` and `sys2` have different `InputName` property values.
- A model resulting from operations on `tf` or `zpk` models inherits its `Variable` property value from the operands. Conflicts are resolved according the following rules:
 - For continuous-time models, 'p' has precedence over 's'.
 - For discrete-time models, 'q⁻¹' and 'z⁻¹' have precedence over 'q' and 'z', while 'q' has precedence over 'z'.
- Most operations ignore the `Notes` and `UserData` properties. These properties of the resulting model are empty.

Result When Connecting Different Model Types

When you use model interconnection commands to combine models of different types, the resulting model type is determined by rules internal to the software. In general:

- When combining Numeric LTI models other than `frd` models, the resulting model is determined by the following order of precedence:

```
ss > zpk > tf > pid > pidstd
```

For example, if you combine an `ss` model, a `pid` model, and a `zpk` model, the result is an `ss` model.

- Any combination that includes a frequency-response model (`frd` or `genfrd`) results in a frequency-response model.
- Combining numeric models with generalized models or with Control Design Blocks results in generalized models. For example, if you combine a `genss` model with a `tf` model, the result is a `genss` model. Likewise, combining a `zpk` model with an `ltiblock.pid` model results in a `genss` model.

Note The software automatically converts all models to the resulting model type before performing the connection operation.

For example, consider the following `ss` model `P`, `ltiblock.tf` model `F`, and `pid` model `C`.

```
P = ss([-0.8,0.4;0.4,-1.0],[-3.0;1.4],[0.3,0],0);  
C = pid(-0.13,-0.61);  
F = ltiblock.tf('F',0,1);
```

The `ss` model has the highest precedence among Numeric LTI models. Therefore, combining `P` and `C` with any model interconnection command returns an `ss` model. For example, entering:

```
CL = feedback(P*C,1)
```

returns a state-space model.

Combining the `ss` model `CL` with the Control Design Block `F` returns a Generalized state-space (`genss`) model.

```
CLF = F*CL
```

Recommended Model Type for Building Block Diagrams

You can represent block diagram components with any model type. However, some connection operations convert models to `ss` form before computing the combined model. Therefore, working with `tf` or `zpk` models causes the software to perform multiple conversions to and from `thess` representation. These conversions can result in high-order polynomials whose evaluation can be plagued by inaccuracies. Additionally, the `tf` or `zpk` representations are inefficient for manipulating multi-input multi-output (MIMO) systems and tend to inflate the model order. Therefore, `ss` or `frd` models yield the most accurate numerical results.

For example, load the models `Pd` and `Cd`, which are 9th-order and 2nd-order discrete-time transfer functions, respectively.

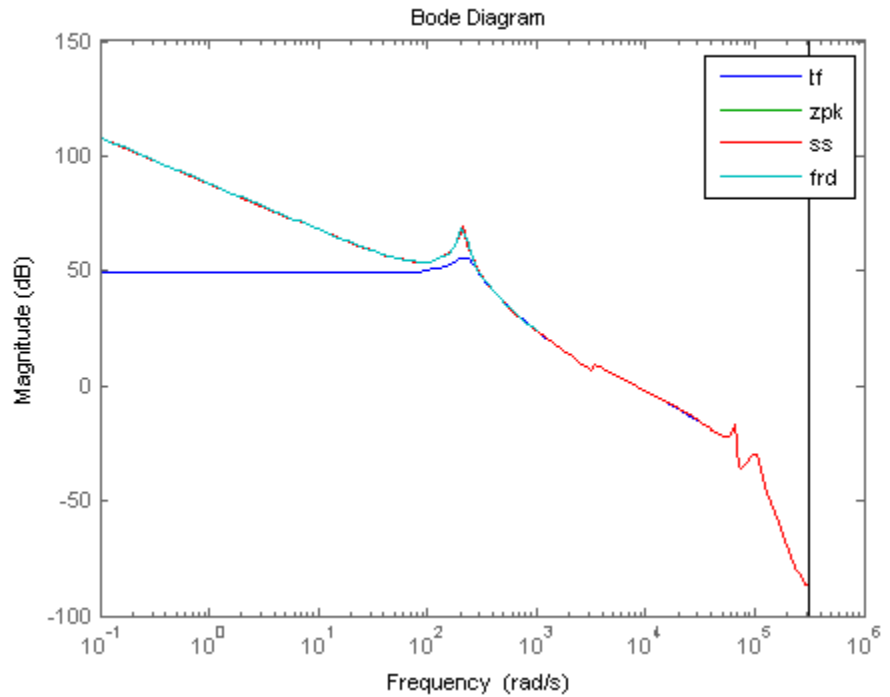
```
load numdemo Pd Cd
```

Even connecting these two models in series can lead to numerical inaccuracies. To see this, compute the open-loop transfer function $L = Pd * Cd$ using the `tf`, `zpk`, `ss`, and `frd` representations.

```
Ltf = Pd*Cd;  
Lzp = zpk(Pd)*Cd;  
Lss = ss(Pd)*Cd;  
  
w = logspace(-1,3,100);  
Lfrd = frd(Pd,w)*Cd;
```

Plot the magnitude of the frequency response to compare the four representations.

```
bodemag(Ltf,Lzp,Lss,Lfrd)  
legend('tf','zpk','ss','frd')
```



The `tf` representation has lost low-frequency dynamics that are preserved in the other representations.

Operations on Models

- “Overview” on page 5-2
- “Precedence and Property Inheritance” on page 5-3
- “Viewing LTI Systems as Matrices” on page 5-5
- “Data Retrieval” on page 5-6
- “Extracting and Modifying Subsystems” on page 5-8
- “Arithmetic Operations on LTI Models” on page 5-15
- “Model Interconnection Functions” on page 5-20
- “Converting Between Continuous- and Discrete-Time Representations” on page 5-24
- “Resampling of Discrete-Time Models” on page 5-37
- “References” on page 5-41

Overview

You can perform basic matrix operations such as addition, multiplication, or concatenation on models objects or model arrays. Such operations are "overloaded," which means that they use the same syntax as they do for matrices, but are adapted to apply to model objects. These overloaded operations and their interpretation in this context are discussed in this chapter. You can read about discretization methods in this chapter as well.

These operations can be applied to model objects of different types. As a result, before discussing operations on LTI models, we discuss model type precedence and how LTI model properties are inherited when models are combined using these operations. To read about functions for analyzing LTI models, see Chapter 6, "Model Analysis Tools".

Precedence and Property Inheritance

You can apply operations to LTI models of different types. Operations on systems of different types work as follows: the resulting type is determined by the precedence rules, and all operands are first converted to this type before performing the operation. Operations like addition and commands like `feedback` operate on more than one LTI model at a time. If these LTI models are represented as LTI objects of different types (for example, the first operand is TF and the second operand is SS), it is not obvious what type (for example, TF or SS) the resulting model should be. Such type conflicts are resolved by *precedence rules*. Specifically, TF, ZPK, SS, and FRD objects are ranked according to the precedence hierarchy:

$$\text{FRD} > \text{SS} > \text{ZPK} > \text{TF}$$

Thus ZPK takes precedence over TF, SS takes precedence over both TF and ZPK, and FRD takes precedence over all three. In other words, any operation involving two or more LTI models produces:

- An FRD object if at least one operand is an FRD object
- An SS object if no operand is an FRD object and at least one operand is an SS object
- A ZPK object if no operand is an FRD or SS object and at least one is an ZPK object
- A TF object only if all operands are TF objects

For example, if `sys1` is a transfer function and `sys2` is a state-space model, then the result of their addition

$$\text{sys} = \text{sys1} + \text{sys2}$$

is a state-space model, since state-space models have precedence over transfer function models.

To supersede the precedence rules and force the result of an operation to be a given type, for example, a transfer function (TF), you can either

- Convert all operands to TF *before* performing the operation

- Convert the result to TF *after* performing the operation

Suppose, in the above example, you want to compute the transfer function of `sys`. You can either use *a priori* conversion of the second operand

```
sys = sys1 + tf(sys2);
```

or *a posteriori* conversion of the result

```
sys = tf(sys1 + sys2)
```

Note These alternatives are not equivalent numerically; computations are carried out on transfer functions in the first case, and on state-space models in the second case.

Another issue is property inheritance, that is, how the operand property values are passed on to the result of the operation. While inheritance is partly operation-dependent, some general rules are summarized below:

- In operations combining discrete-time LTI models, all models must have identical or unspecified (`sys.Ts = -1`) sample times. Models resulting from such operations inherit the specified sample time, if there is one.
- Most operations ignore the `Notes` and `Userdata` properties.
- In general, when two LTI models `sys1` and `sys2` are combined using operations such as `+`, `*`, `[,]`, `[;]`, `append`, and `feedback`, the resulting model inherits its I/O names and I/O groups from `sys1` and `sys2`. However, conflicting I/O names or I/O groups are not inherited. For example, the `InputName` property for `sys1 + sys2` is left unspecified if `sys1` and `sys2` have different `InputName` property values.
- A model resulting from operations on TF or ZPK models inherits its `Variable` property value from the operands. Conflicts are resolved according the following rules:
 - For continuous-time models, 'p' has precedence over 's'.
 - For discrete-time models, 'z⁻¹' has precedence over 'q' and 'z', while 'q' has precedence over 'z'.

Viewing LTI Systems as Matrices

In the frequency domain, an LTI system is represented by the linear input/output map

$$y = Hu.$$

This map is characterized by its transfer matrix H , a function of either the Laplace or Z-transform variable. The transfer matrix H maps inputs to outputs, so there are as many columns as inputs and as many rows as outputs.

If you think of LTI systems in terms of (transfer) matrices, certain basic operations on LTI systems are naturally expressed with a matrix-like syntax. For example, the parallel connection of two LTI systems `sys1` and `sys2` can be expressed as

$$\text{sys} = \text{sys1} + \text{sys2}$$

because parallel connection amounts to adding the transfer matrices. Similarly, subsystems of a given LTI model `sys` can be extracted using matrix-like subscripting. For instance,

$$\text{sys}(3, 1:2)$$

provides the I/O relation between the first two inputs (column indices) and the third output (row index), which is consistent with

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} H(1,1) & H(2,1) \\ H(2,1) & H(2,2) \\ H(3,1) & H(3,2) \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

for $y = Hu$.

Data Retrieval

The functions `tf`, `zpk`, `ss`, and `frd` pack the model data and sample time in a single LTI object. Conversely, the following commands provide convenient data retrieval for any type of TF, SS, or ZPK model `sys`, or FRD model `sysfr`.

```
[num,den,Ts] = tfdata(sys)    % Ts = sample time
[z,p,k,Ts] = zpkdata(sys)
[a,b,c,d,Ts] = ssdata(sys)
[a,b,c,d,e,Ts] = dssdata(sys)
[response,frequency,Ts] = frdata(sysfr)
```

Note that:

- `sys` can be any type of LTI object, *except* an FRD model
- `sysfr`, the input argument to `frdata`, can only be an FRD model

You can use any variable names you want in the output argument list of any of these functions.

The output arguments `num` and `den` assigned to `tfdata`, and `z` and `p` assigned to `zpkdata`, are cell arrays, even in the SISO case. These cell arrays have as many rows as outputs, as many columns as inputs, and their ij th entry specifies the transfer function from the j th input to the i th output. For example,

```
H = [tf([1 -1],[1 2 10]) , tf(1,[1 0])]
```

creates the one-output/two-input transfer function

$$H(s) = \begin{bmatrix} \frac{s-1}{s^2+2s+10} & \frac{1}{s} \end{bmatrix}.$$

Typing

```
[num,den] = tfdata(H);
num{1,1}, den{1,1}
```

displays the coefficients of the numerator and denominator of the first input channel.

```
ans =
    0     1    -1
ans =
    1     2    10
```

Note that the same result is obtained using dot notation:

```
H.num{1,1}, H.den{1,1}
```

To obtain the numerator and denominator of SISO systems directly as row vectors, use the syntax

```
[num,den,Ts] = tfdata(sys,'v')
```

For example, typing

```
sys = tf([1 3],[1 2 5]);
[num,den] = tfdata(sys,'v')
```

produces

```
num =
    0     1     3
den =
    1     2     5
```

Similarly,

```
[z,p,k,Ts] = zpkdata(sys,'v')
```

returns the zeros, z , and the poles, p , as *vectors* for SISO systems.

Extracting and Modifying Subsystems

In this section...

“What is a Subsystem?” on page 5-8

“Basic Subsystem Concepts” on page 5-8

“Referencing FRD Models Through Frequencies” on page 5-11

“Referencing Channels by Name” on page 5-12

“Resizing LTI Systems” on page 5-13

What is a Subsystem?

Subsystems relate subsets of the inputs and outputs of a system. The transfer matrix of a subsystem is a submatrix of the system transfer matrix.

Basic Subsystem Concepts

For example, if `sys` is a system with two inputs, three outputs, and I/O relation

$$y = Hu$$

then $H(3, 1)$ gives the relation between the first input and third output:

$$y_3 = H(3, 1)u_1.$$

Accordingly, use matrix-like subindexing to extract this subsystem.

$$\text{SubSys} = \text{sys}(3,1)$$

The resulting subsystem `SubSys` is an LTI model of the same type as `sys`, with its sample time, time delay, I/O name, and I/O group property values inherited from `sys`.

For example, if `sys` has an input group named `controls` consisting of channels one, two, and three, then `SubSys` also has an input group named `controls` with the first channel of `SubSys` assigned to it.

If `sys` is a state-space model with matrices `a`, `b`, `c`, `d`, the subsystem `sys(3,1)` is a state-space model with data `a`, `b(:,1)`, `c(3,:)`, `d(3,1)`. Note the following rules when extracting subsystems:

- In the expression `sys(3,1)`, the first index selects the output channel while the second index selects the input channel.
- When extracting a subsystem from a given state-space model, the resulting state-space model may not be minimal. Use the command `smnreal` to eliminate unnecessary states in the subsystem.

You can use similar syntax to modify the LTI model `sys`. For example,

```
sys(3,1) = NewSubSys
```

redefines the I/O relation between the first input and third output, provided `NewSubSys` is a SISO LTI model.

Rules for Modifying LTI Model Subsystems

The following rules apply when modifying LTI model subsystems:

- `sys`, the LTI model that has had a portion reassigned, retains its original model type (TF, ZPK, SS, or FRD) regardless of the model type of `NewSubSys`.
- Subsystem assignment does not reassign any I/O names or I/O group names of `NewSubSys` that are already assigned to `NewSubSys`.
- Reassigning parts of a MIMO state-space model generally increases its order.
- If `NewSubSys` is an FRD model, then `sys` must also be an FRD model. Furthermore, their frequencies must match.

Other standard matrix subindexing extends to LTI objects as well. For example,

```
sys(3,1:2)
```

extracts the subsystem mapping the first two inputs to the third output.

```
sys(:,1)
```

selects the first input and all outputs, and

```
sys([1 3],:)
```

extracts a subsystem with the same inputs, but only the first and third outputs.

For example, consider the two-input/two-output transfer function

$$T(s) = \begin{bmatrix} \frac{1}{s+0.1} & 0 \\ \frac{s-1}{s^2+2s+2} & \frac{1}{s} \end{bmatrix}.$$

To extract the transfer function $T_{11}(s)$ from the first input to the first output, type

```
T(1,1)
```

```
Transfer function:
```

```
  1
-----
s + 0.1
```

Next reassign $T_{11}(s)$ to $1/(s + 0.5)$ and modify the second input channel of T by typing

```
T(1,1) = tf(1,[1 0.5]);
T(:,2) = [ 1 ; tf(0.4,[1 0]) ]
```

```
Transfer function from input 1 to output...
```

```
  1
-----
#1:  s + 0.5

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


Transfer function from input 2 to output...

```
#1: 1

      0.4
#2: ---
      s
```

Referencing FRD Models Through Frequencies

You can extract subsystems from FRD models, as you do with other LTI model types, by indexing into input and output (I/O) dimensions. You can also extract subsystems by indexing into the frequencies of an FRD model.

To index into the frequencies of an FRD model, use the string *'Frequency'* (or any abbreviation, such as, *'freq'*, as long as it does not conflict with existing I/O channel or group names) as a keyword. There are two ways you can specify FRD models using frequencies:

- Using integers to index into the frequency vector of the FRD model
- Using a Boolean (logical) expression to specify desired frequency points in an FRD model

For example, if `sys` is an FRD model with five frequencies, (e.g., `sys.Frequency=[1 1.1 1.2 1.3 1.4]`), then you can create a new FRD model `sys2` by indexing into the frequencies of `sys` as follows.

```
sys2 = sys('frequency', 2:3);
sys2.Frequency
```

```
ans =
    1.1000
    1.2000
```

displays the second and third entries in the frequency vector.

Similarly, you can use logical indexing into the frequencies.

```
sys2 = sys('frequency', sys.Frequency >1.0 & sys.Frequency <1.15);
sys2.freq
```

```
ans =  
    1.1000
```

You can also combine model extraction through frequencies with indexing into the I/O dimensions. For example, if `sys` is an FRD model with two inputs, two outputs, and frequency vector `[2.1 4.2 5.3]`, with `sys.Units` specified in rad/s, then

```
sys2 = sys(1,2,'freq',1)
```

specifies `sys2` as a SISO FRD model, with one frequency data point, 2.1 rad/s.

Referencing Channels by Name

You can also extract subsystems using I/O group or channel names. For example, if `sys` has an input group named `noise`, consisting of channels two, four, and five, then

```
sys(1,'noise')
```

is equivalent to

```
sys(1,[2 4 5])
```

Similarly, if `pressure` is the name assigned to an output channel of the LTI model `sys`, then

```
sys('pressure',1) = tf(1, [1 1])
```

reassigns the subsystem from the first input of `sys` to the output labeled `pressure`.

You can reference a set of channels by input or output name by using a cell array of strings for the names. For example, if `sys` has one output channel named `pressure` and one named `temperature`, then these two output channels can be referenced using

```
sys({'pressure','temperature'})
```

Resizing LTI Systems

Resizing a system consists of adding or deleting inputs and/or outputs. To delete the first two inputs, simply type

```
sys(:,1:2) = []
```

In deletions, at least one of the row/column indexes should be the colon (:) selector.

To perform input/output augmentation, you can proceed by concatenation or subassignment. Given a system `sys` with a single input, you can add a second input using

```
sys = [sys,h];
```

or, equivalently, using

```
sys(:,2) = h;
```

where `h` is any LTI model with one input, and the same number of outputs as `sys`. There is an important difference between these two options: while concatenation obeys the precedence rules, subsystem assignment does not alter the model type. So, if `sys` and `h` are TF and SS objects, respectively, the first statement produces a state-space model, and the second statement produces a transfer function.

For state-space models, both concatenation and subsystem assignment increase the model order because they assume that `sys` and `h` have independent states. If you intend to keep the same state matrix and only update the input-to-state or state-to-output relations, use `set` instead and modify the corresponding state-space data directly. For example,

```
sys = ss(a,b1,c,d1)
set(sys,'b',[b1 b2],'d',[d1 d2])
```

adds a second input to the state-space model `sys` by appending the B and D matrices. You should *simultaneously* modify both matrices with a single `set` command. Indeed, the statements

```
sys.b = [b1 b2]
```

and

```
set(sys, 'b', [b1 b2])
```

cause an error because they create invalid intermediate models in which the B and D matrices have inconsistent column dimensions.

Arithmetic Operations on LTI Models

| In this section... |
|---|
| “Supported Arithmetic Operations” on page 5-15 |
| “Addition and Subtraction” on page 5-15 |
| “Multiplication” on page 5-17 |
| “Inversion and Related Operations” on page 5-18 |
| “Transposition” on page 5-18 |
| “Pertransposition” on page 5-19 |

Supported Arithmetic Operations

You can apply almost all arithmetic operations to LTI models, including those shown below.

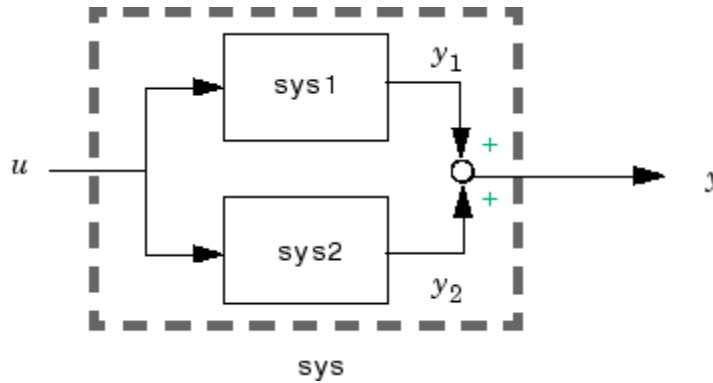
| Operation | Description |
|-----------|---------------------------------------|
| + | Addition |
| - | Subtraction |
| * | Multiplication |
| .* | Element-by-element multiplication |
| / | Right matrix divide |
| \ | Left matrix divide |
| inv | Matrix inversion |
| ' | Pertransposition |
| .' | Transposition |
| ^ | Powers of an LTI model (as in s^2) |

Addition and Subtraction

Adding LTI models is equivalent to connecting them in parallel. Specifically, the LTI model

$$\text{sys} = \text{sys1} + \text{sys2}$$

represents the parallel interconnection shown below.



If sys1 and sys2 are two state-space models with data A_1, B_1, C_1, D_1 and A_2, B_2, C_2, D_2 , the state-space data associated with $\text{sys1} + \text{sys2}$ is

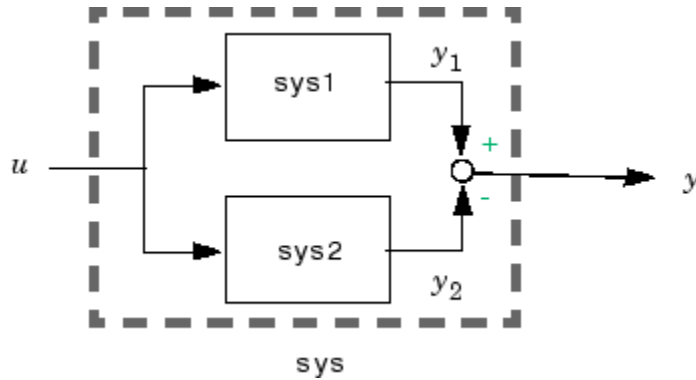
$$\begin{bmatrix} A_1 & 0 \\ 0 & A_2 \end{bmatrix}, \begin{bmatrix} B_1 \\ B_2 \end{bmatrix}, [C_1 \ C_2], D_1 + D_2.$$

Scalar addition is also supported and behaves as follows: if sys1 is MIMO and sys2 is SISO, $\text{sys1} + \text{sys2}$ produces a system with the same dimensions as sys1 whose ij th entry is $\text{sys1}(i, j) + \text{sys2}$.

Similarly, the subtraction of two LTI models

$$\text{sys} = \text{sys1} - \text{sys2}$$

is depicted by the following block diagram.

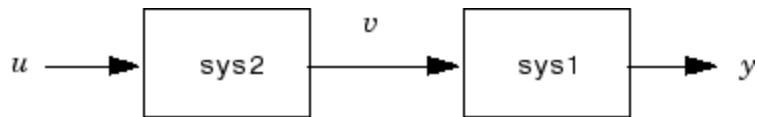


Multiplication

Multiplication of two LTI models connects them in series. Specifically,

$$\text{sys} = \text{sys1} * \text{sys2}$$

returns an LTI model `sys` for the series interconnection shown below.



Notice the reverse orders of `sys1` and `sys2` in the multiplication and block diagram. This is consistent with the way transfer matrices are combined in a series connection: if `sys1` and `sys2` have transfer matrices H_1 and H_2 , then

$$y = H_1 v = H_1(H_2 u) = (H_1 \times H_2)u.$$

For state-space models `sys1` and `sys2` with data A_1, B_1, C_1, D_1 and A_2, B_2, C_2, D_2 , the state-space data associated with `sys1*sys2` is

$$\begin{bmatrix} A_1 & B_1 C_2 \\ 0 & A_2 \end{bmatrix}, \begin{bmatrix} B_1 D_2 \\ B_2 \end{bmatrix}, [C_1 \quad D_1 C_2], D_1 D_2.$$

Finally, if `sys1` is MIMO and `sys2` is SISO, then `sys1*sys2` or `sys2*sys1` is interpreted as an entry-by-entry scalar multiplication and produces a system with the same dimensions as `sys1`, whose ij th entry is `sys1(i, j)*sys2`.

Inversion and Related Operations

Inversion of LTI models amounts to inverting the following input/output relationship.

$$y = Hu \rightarrow u = H^{-1}y.$$

This operation is defined only for square systems (that is, systems with as many inputs as outputs) and is performed using

```
inv(sys)
```

The resulting inverse model is of the same type as `sys`. Related operations include:

- Left division `sys1\sys2`, which is equivalent to `inv(sys1)*sys2`
- Right division `sys1/sys2`, which is equivalent to `sys1*inv(sys2)`

For a state-space model `sys` with data A, B, C, D , `inv(sys)` is defined only when D is a square invertible matrix, in which case its state-space data is

$$A - BD^{-1}C, \quad BD^{-1}, \quad -D - 1C, \quad D^{-1}.$$

Transposition

You can transpose an LTI model `sys` using

```
sys.'
```

This is a literal operation with the following effect:

- For TF models (with input arguments, `num` and `den`), the cell arrays `num` and `den` are transposed.
- For ZPK models (with input arguments, `z`, `p`, and `k`), the cell arrays, `z` and `p`, and the matrix `k` are transposed.

- For SS models (with model data A, B, C, D), transposition produces the state-space model A^T, C^T, B^T, D^T .
- For FRD models (with complex frequency response matrix `Response`), the matrix of frequency response data at each frequency is transposed.

Pertransposition

For a continuous-time system with transfer function $H(s)$, the *pertransposed* system has the transfer function

$$G(s) = [H(-s)]^T.$$

The discrete-time counterpart is

$$G(z) = [H(z^{-1})]^T.$$

Pertransposition of an LTI model `sys` is performed using

```
sys'
```

You can use pertransposition to obtain the Hermitian (conjugate) transpose of the frequency response of a given system. The frequency response of the pertranspose of $H(s)$, $G(s) = [H(-s)]^T$, is the Hermitian transpose of the frequency response of $H(s)$: $G(j\omega) = H(j\omega)^H$.

To obtain the Hermitian transpose of the frequency response of a system `sys` over a frequency range specified by the vector `w`, type

```
freqresp(sys', w);
```

Model Interconnection Functions

| In this section... |
|---|
| “Supported Interconnection Functions” on page 5-20 |
| “Concatenation of LTI Models” on page 5-21 |
| “Feedback and Other Interconnection Functions” on page 5-22 |

Supported Interconnection Functions

You can use the Control System Toolbox model interconnection functions to help you build models. With these functions, you can perform I/O concatenation (`[,]`, `[;]`, and `append`), general parallel and series connections (`parallel` and `series`), and feedback connections (`feedback` and `lft`). These functions are useful to model open- and closed-loop systems.

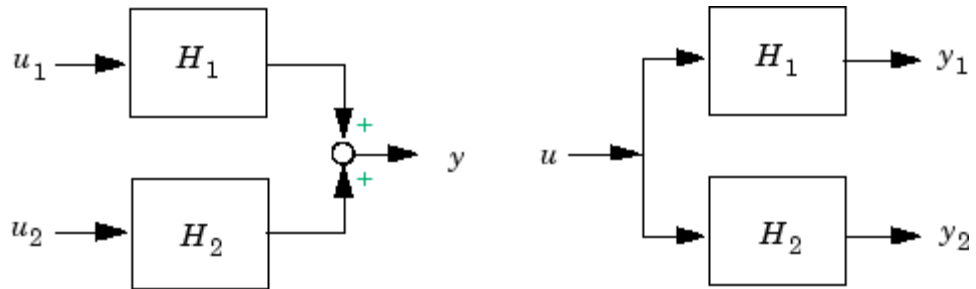
| Interconnection Operator | Description |
|--------------------------|--|
| <code>[,]</code> | Concatenates horizontally |
| <code>[;]</code> | Concatenates vertically |
| <code>append</code> | Appends models in a block diagonal configuration |
| <code>augstate</code> | Augments the output by appending states |
| <code>connect</code> | Forms an SS model from a block diagonal LTI object for an arbitrary interconnection matrix |
| <code>feedback</code> | Forms the feedback interconnection of two models |
| <code>lft</code> | Produces the LFT interconnection (Redheffer Star product) of two models |
| <code>parallel</code> | Forms the generalized parallel connection of two models |
| <code>series</code> | Forms the generalized series connection of two models |

Concatenation of LTI Models

LTI model concatenation is done in a manner similar to the way you concatenate matrices in the MATLAB technical computing environment, using

```
sys = [sys1 , sys2] % horizontal concatenation
sys = [sys1 ; sys2] % vertical concatenation
sys = append(sys1,sys2) % block diagonal appending
```

In I/O terms, horizontal and vertical concatenation have the following block-diagram interpretations (with H_1 and H_2 denoting the transfer matrices of sys1 and sys2).



$$y = \begin{bmatrix} H_1 & H_2 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} H_1 \\ H_2 \end{bmatrix} u$$

Horizontal Concatenation

Vertical Concatenation

You can use concatenation as an easy way to create MIMO transfer functions or zero-pole-gain models. For example,

```
H = [ tf(1,[1 0]) 1 ; 0 tf([1 -1],[1 1]) ]
```

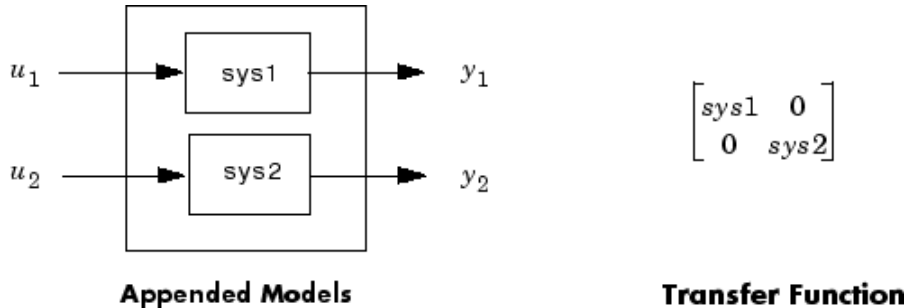
specifies

$$H(s) = \begin{bmatrix} \frac{1}{s} & 1 \\ 0 & \frac{s-1}{s+1} \end{bmatrix}.$$

Use

```
append(sys1, sys2)
```

to specify the block-decoupled LTI model interconnection.



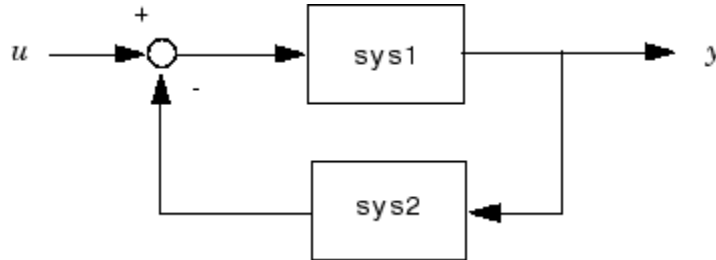
See `append` for more information on this function.

Feedback and Other Interconnection Functions

The following LTI model interconnection functions are useful for specifying closed- and open-loop model configurations:

- `feedback` puts two LTI models with compatible dimensions in a feedback configuration.
- `series` connects two LTI models in series.
- `parallel` connects two LTI models in parallel.
- `lft` performs the Redheffer star product on two LTI models.
- `connect` works with `append` to apply an arbitrary interconnection scheme to a set of LTI models.

For example, if sys1 has m inputs and p outputs, while sys2 has p inputs and m outputs, then the negative feedback configuration of these two LTI models



is realized with

```
feedback(sys1, sys2)
```

This specifies the LTI model with m inputs and p outputs whose I/O map is

$$(I + \text{sys1} \cdot \text{sys2})^{-1} \text{sys1}.$$

See the reference pages online for more information on `feedback`, `series`, `parallel`, `lft`, and `connect`.

Converting Between Continuous- and Discrete-Time Representations

In this section...

“Supported Conversion Functions and Methods” on page 5-24

“Zero-Order Hold Conversion Method” on page 5-25

“First-Order Hold Conversion Method” on page 5-27

“Impulse-Invariant Mapping” on page 5-28

“Tustin Approximation” on page 5-32

“Zero-Pole Matching Equivalents” on page 5-35

Supported Conversion Functions and Methods

The function `c2d` discretizes continuous-time TF, SS, or ZPK models. Conversely, `d2c` converts discrete-time TF, SS, or ZPK models to continuous time. Both `c2d` and `d2d` support several discretization and interpolation methods, as shown in the following table.

| Discretization Method | Use when: |
|---------------------------|--|
| Zero-order hold (ZOH) | You want an exact discretization in the time domain for staircase inputs. |
| First-order hold (FOH) | You want an exact discretization in the time domain for piecewise linear inputs. |
| Impulse-invariant mapping | <ul style="list-style-type: none"> You want an exact discretization in the time domain for impulse train inputs. You do not need exact phase matching in the frequency domain. |

| Discretization Method | Use when: |
|--------------------------------|---|
| Tustin approximation | <ul style="list-style-type: none"> • You want good matching in the frequency domain between the continuous- and discrete-time models. • Your model has important dynamics near the Nyquist frequency. |
| Zero-pole matching equivalents | You have a SISO model, and you want good matching in the frequency domain between the continuous- and discrete-time models. |

The default syntax performs a ZOH conversion:

```
sysd = c2d(sysc, Ts);    % Ts = sampling period in seconds
sysc = d2c(sysd);
```

You can specify another method as string input to d2c or c2d:

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

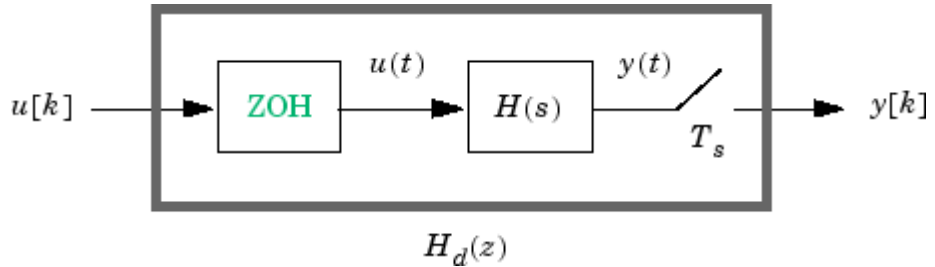
This syntax uses the default options for the specified discretization method. (See the c2d and d2c reference pages for more detail.) You can specify different options for discretizing systems with the 'tustin' or 'matched' methods using c2dOptions or d2cOptions to define the options. For example, to discretize a system using the Tustin method with a prewarp transformation at 4.2 rad/s and a sample time of 0.1 s:

```
opt = c2dOptions('Method', 'tustin', 'PrewarpFrequency', 4.2);
sysd = c2d(sysc, 0.1, opt);
```

See c2d and c2dOptions for more information.

Zero-Order Hold Conversion Method

The following block diagram illustrates the zero-order-hold discretization $H_d(z)$ of a continuous-time linear model $H(s)$



The ZOH block generates the continuous-time input signal $u(t)$ by holding each sample value $u(k)$ constant over one sample period:

$$u(t) = u[k], \quad kT_s \leq t \leq (k+1)T_s$$

The signal $u(t)$ is the input to the continuous system $H(s)$. The output $y[k]$ results from sampling $y(t)$ every T_s seconds.

Conversely, given a discrete system $H_d(z)$, `d2c` produces a continuous system $H(s)$. The ZOH discretization of $H(s)$ coincides with $H_D(z)$.

The ZOH discrete-to-continuous conversion has the following limitations:

- `d2c` cannot convert LTI models with poles at $z = 0$.
- For discrete-time LTI models having negative real poles, ZOH `d2c` conversion produces a continuous system with higher order. The model order increases because negative real poles in the z domain map to pairs of complex poles in the s domain.

The next example illustrates the behavior of `d2c` with real negative poles. Consider the following discrete-time zpk model, with one real negative pole at $z = -0.5$.

```
hd = zpk([], -0.5, 1, 0.1)
Zero/pole/gain:
  1
  -----
 (z+0.5)

Sampling time: 0.1
```


Use `d2c` to convert this model to continuous-time with the default ZOH method:

```
hc = d2c(hd)
```

The result is a second-order model:

```
Zero/pole/gain:
  4.621 (s+149.3)
-----
(s^2 + 13.86s + 1035)
```

Discretizing the model again returns the original discrete-time system, up to canceling the pole/zero pair at $z = -0.5$:

```
c2d(hc,0.1)

Zero/pole/gain:
 (z+0.5)
-----
 (z+0.5)^2

Sampling time: 0.1
```

ZOH Method for Systems with Time Delays

You can use the ZOH method to discretize SISO or MIMO continuous-time models with time delays. The ZOH method yields an exact discretization for systems with I/O delays only (no internal delays).

For more details about how the ZOH method handles systems with time delays, see “ZOH, FOH, and Impulse-Invariant Methods” in the `c2d` reference page .

First-Order Hold Conversion Method

First-order hold (FOH) differs from ZOH by the underlying hold mechanism. To turn the input samples $u[k]$ into a continuous input $u(t)$, FOH uses linear interpolation between samples:

$$u(t) = u[k] + \frac{t - kT_s}{T_s} (u[k+1] - u[k]), \quad kT_s \leq t \leq (k+1)T_s$$

This method is generally more accurate than ZOH for systems driven by smooth inputs. Because of causality constraints, you can use this option only for c2d conversions and not d2c conversions.

Note This FOH method differs from standard causal FOH and is more appropriately called *triangle approximation* (see [2], p. 228). It is also known as *ramp-invariant* approximation because it is distortion-free for ramp inputs.

FOH Method for Systems with Time Delays

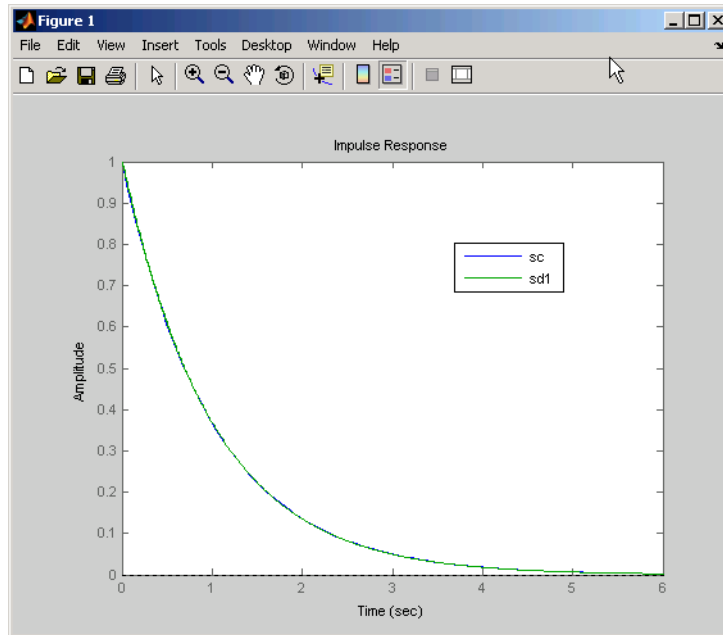
You can use the FOH method to discretize SISO or MIMO continuous-time models with time delays. The FOH method handles time delays in the same way as the ZOH method. See “ZOH Method for Systems with Time Delays” on page 5-27 and the c2d reference page for more details.

For more details about how the ZOH method handles systems with time delays, see “ZOH, FOH, and Impulse-Invariant Methods” in the c2d reference page .

Impulse-Invariant Mapping

The impulse-invariant mapping matches the discretized impulse response to that of the continuous time system. For example:

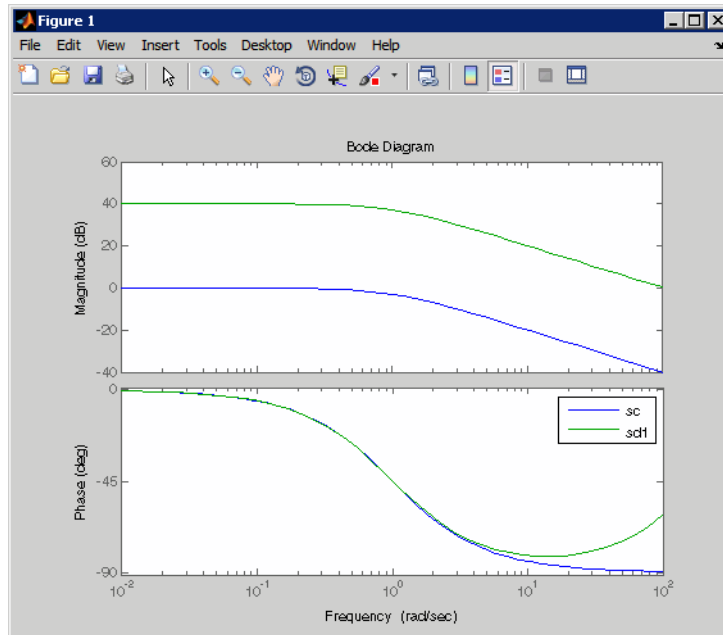
```
n = 1; d = [1 1]; % Simple 1st order continuous system
sc = ss(tf(n, d)); % state space representation
% Convert to discrete system via impulse invariant mapping
sd1 = c2d(sc, 0.01, 'impulse');
impulse(sc, sd1) % Plot both impulse responses
```



The impulse response plot shows that the impulse responses match. Impulse-invariant mapping is therefore ideal when you want the discretized system to match the time-domain impulse response of the continuous-time system.

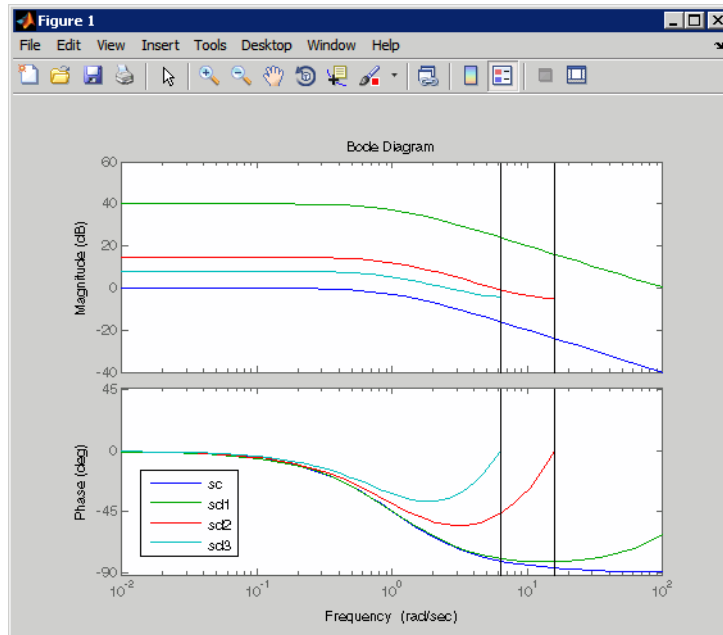
The frequency responses do not match, however. For example:

```
bode(sc, sd1)
```



Impulse-invariant mapping introduces a shift in the DC gain of the discretized system. In addition, it introduces a phase mismatch at higher frequencies. The phase mismatch results from alias effects, and this effect becomes more pronounced as the sampling time increases. For example:

```
sd2 = c2d(sc, 0.2, 'impulse');
sd3 = c2d(sc, 0.5, 'impulse');
bode(sc, sd1, sd2, sd3)
```



While the shift in the DC gain of this system decreases with decreasing sampling time, the aliasing effects become more pronounced. Because of aliasing, impulse-invariant mapping is not a good choice if you want to match the frequency response of the continuous system. In most cases, choose a bilinear transform (such as “Tustin Approximation” on page 5-32) for preserving the frequency-domain response of the transformed model.

In general, impulse-invariant discretization does not preserve the DC gain. For example, consider the continuous-time transfer function:

$$G_c(s) = \frac{1}{s + a}.$$

The DC gain $G_c(0) = 1/a$. The impulse-invariant discretization of $G_c(s)$ is:

$$G_d(z) = \frac{z}{z - e^{-aT_s}}$$

where T_s is the sampling time. The DC gain of the discretized system is:

$$G_d(1) = \frac{1}{1 - e^{-aT_s}}$$

For further discussion of impulse invariance scaling issues and aliasing, see [3].

Impulse-Invariant Mapping for Systems with Time Delays

You can use impulse-invariant mapping to discretize SISO or MIMO continuous-time models with time delay, except that the method does not support `ss` models with internal delays. For supported models, impulse-invariant mapping yields an exact discretization. See the `c2d` reference page for more information.

Tustin Approximation

The Tustin or bilinear approximation uses the approximation

$$z = e^{sT_s} \approx \frac{1 + sT_s/2}{1 - sT_s/2}$$

to relate s -domain and z -domain transfer functions. In `c2d` conversions, the discretization $H_d(z)$ of a continuous transfer function $H(s)$ is:

$$H_d(z) = H(s'), \quad s' = \frac{2}{T_s} \frac{z-1}{z+1}$$

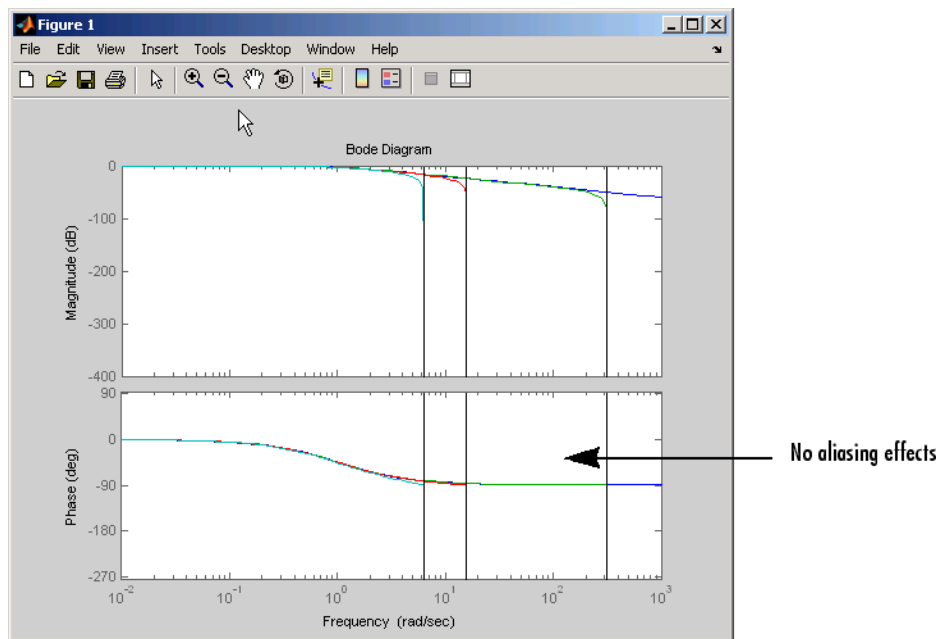
Similarly, the `d2c` conversion relies on the inverse correspondence

$$H(s) = H_d(z'), \quad z' = \frac{1 + sT_s/2}{1 - sT_s/2}$$

Use the Tustin discretization method if you need good frequency domain matching between your continuous-time system and the corresponding discretized system. (See “Impulse-Invariant Mapping” on page 5-28.)

For example, the following demonstration shows that in contrast to 'impulse', the 'tustin' method is free of aliasing effects that depend upon sampling time:

```
n = 1; d = [1 1]; % Simple 1st order continuous system
sc = ss(tf(n, d)); % state space representation
% Convert to discrete system with three different sampling times
sd4 = c2d(sc, 0.1, 'tustin');
sd5 = c2d(sc, 0.2, 'tustin');
sd6 = c2d(sc, 0.5, 'tustin');
bode(sc, sd4, sd5, sd6)
```



Tustin Approximation with Frequency Prewarping

The Tustin approximation with frequency prewarping uses this transformation of variables:

$$H_d(z) = H(s'), \quad s' = \frac{\omega}{\tan(\omega T_s / 2)} \frac{z - 1}{z + 1}$$

This change of variable ensures the matching of the continuous- and discrete-time frequency responses at the prewarp frequency ω , because of the following correspondence:

$$H(j\omega) = H_d(e^{j\omega T_s})$$

To use the Tustin approximation with frequency prewarping, use `c2dOptions` to specify the prewarp frequency. For example:

```
n = 1; d = [1 1]; % Simple 1st order continuous system
sc = ss(tf(n, d)); % state space representation

% create a c2dOptions object specifying the Tustin method
% with prewarp frequency 4.2 rad/s
opt = c2dOptions('Method', 'tustin', 'PrewarpFrequency', 4.2);
sd = c2d(sc, 0.01, opt);
```

Specifying a `PrewarpFrequency` of 0 is equivalent to using the Tustin method with no prewarp. See the `c2d` and `c2dOptions` reference pages for more information about specifying discretization options.

Tustin Approximation for Systems with Time Delays

You can use the Tustin approximation to discretize SISO or MIMO continuous-time models with time delays. The Tustin approximation yields an approximate discretization for a system with time delay τ . The integer portion of the delay, kT_s , maps to a delay of k sampling periods in the discretized model. This approach ignores the residual fractional delay, $\tau - kT_s$, by default. You can instead approximate the fractional delay by a Thiran filter using the `c2dOptions` command.

For example, consider the following transfer function:

$$H(s) = e^{-0.25s} \frac{10}{s^2 + 3s + 10}.$$

To discretize this system using a Thiran filter to approximate a residual fractional delay, enter:


```
h = tf(10,[1 3 10],'iodelay',0.25); % create transfer function
% define the discretization options
opts = c2dOptions('Method', 'tustin', 'FractDelayApproxOrder', 3);
hdtust = c2d(h, 0.1, opts);
```

These commands discretize `h` with a sample time of 0.1 s, approximating the residual fractional time delay as a Thiran filter of order up to 3.

For state-space systems, `c2d` models the Thiran filter as additional states in the discretized model.

For more details about using the Tustin approximation to discretize systems with time delays, see “Tustin Approximation and Zero-Pole Matching Methods” on the `c2d` reference page. For a discussion of Thiran filters, see the `thiran` reference page and [3].

Zero-Pole Matching Equivalents

The method of conversion by computing zero-pole matching equivalents applies only to SISO systems. The continuous and discretized systems have matching DC gains. Their poles and zeros are related by the transformation:

$$z_i = e^{s_i T_s}$$

where:

- z_i is the i th pole or zero of the discrete-time system.
- s_i is the i th pole or zero of the continuous-time system.
- T_s is the sampling time.

See [2] for more details.

Use the zero-pole matching method by specifying the matched method with `c2d`, `c2dOptions`, `d2c`, or `d2cOptions`. See the reference pages for those commands for more information.

Zero-Pole Matching for Systems with Time Delays

You can use zero-pole matching to discretize SISO continuous-time models with time delay, except that the method does not support `ss` models with internal delays. The zero-pole matching method handles time delays in the same way as the Tustin approximation. See “Tustin Approximation and Zero-Pole Matching Methods” on the `c2d` reference page.

Resampling of Discrete-Time Models

In this section...

“Available Commands for Resampling Discrete-Time Models” on page 5-37
 “Example of Resampling a Discrete-Time Model” on page 5-37

Available Commands for Resampling Discrete-Time Models

You can resample a discrete-time TF, SS, or ZPK model using the commands described in the following table.

| To... | Use the command... |
|---|--------------------|
| <ul style="list-style-type: none"> Downsample a system. Upsample a system without any restriction on the new sampling time. | d2d |
| Upsample a system with the highest accuracy when: <ul style="list-style-type: none"> The new sample time is integer-value-times faster than the sample time of the original model. Your new model can have more states than the original model. | upsample |

Example of Resampling a Discrete-Time Model

This example shows how to upsample a system using both the d2d and upsample commands and compares the results of both to the original system.

1 Create the original system, h1, by typing:

```
h1 = tf([1 0.4],[1 -0.7],0.3)
```

This command returns the following result:

Transfer function:

$z + 0.4$

$z - 0.7$

Sampling time: 0.3

The sample time is 0.3 seconds.

- 2** Create a new system with a sampling time of 0.1 seconds using the `d2d` command. Type:

```
h2 = d2d(h1,0.1)
```

This command returns the following result:

Transfer function:

$z - 0.4769$

$z - 0.8879$

Sampling time: 0.1

The sample time is 0.1 seconds.

- 3** Create another new system with a sampling time of 0.1 seconds using the `upsample` command by typing:

```
h3 = upsample(h1,3)
```

This command returns the following result:

Transfer function:

$z^3 + 0.4$

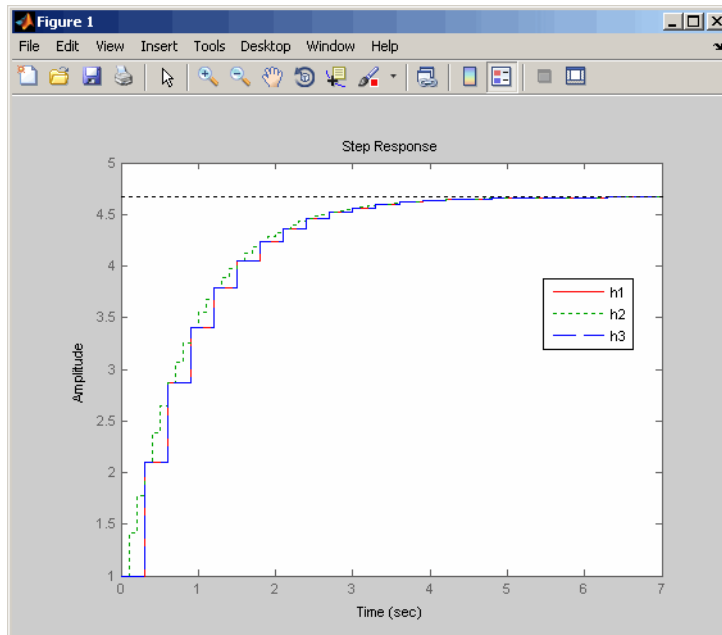
$z^3 - 0.7$

Sampling time: 0.1

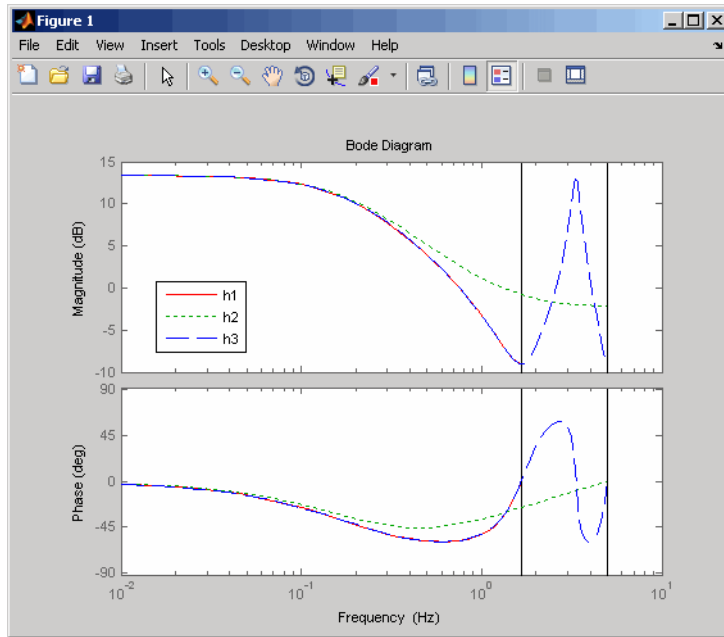
The sample time is 0.1 seconds and `h3` has three times as many poles and zeros as `h1`.

- 4 Compare the results of the new upsampled systems h2 and h3 to the original system h1. Use step response and bode plots by typing:

```
step(h1, '-r', h2, ':g', h3, '--b')
figure
bode(h1, '-r', h2, ':g', h3, '--b')
```



The step response plot shows that the upsampled system h3, created using the `upsample` command, provides a better match than h2 to the original system h1. The h3 system matches h1 at multiples of the original sampling time.



The bode plot shows that the upsampled system h_3 , created using the `upsample` command, provides an exact match of the original system h_1 up to the Nyquist frequency π / T_{s_0} , where T_{s_0} is the sampling time of the original system.

References

- [1] Åström, K.J. and B. Wittenmark, *Computer-Controlled Systems: Theory and Design*, Prentice-Hall, 1990, pp. 48-52.
- [2] Franklin, G.F., Powell, D.J., and Workman, M.L., *Digital Control of Dynamic Systems* (3rd Edition), Prentice Hall, 1997.
- [3] Smith, J.O. III, “Impulse Invariant Method”, *Physical Audio Signal Processing*, August 2007.
http://www.dsprelated.com/dspbooks/pasp/Impulse_Invariant_Method.html.
- [4] T. Laakso, V. Valimaki, “Splitting the Unit Delay”, *IEEE Signal Processing Magazine*, Vol. 13, No. 1, p.30-60, 1996.

Model Analysis Tools

- “General Model Characteristics” on page 6-2
- “Model Dynamics” on page 6-4
- “State-Space Realizations” on page 6-7
- “Analyzing Systems With Time Delays” on page 6-8
- “Sensitivity Analysis” on page 6-16
- “Discretization” on page 6-19

General Model Characteristics

General model characteristics include the model type, I/O dimensions, and continuous or discrete nature. Related commands are listed in the table below. These commands operate on continuous- or discrete-time LTI models or arrays of LTI models of any type.

| General Model Characteristics Commands | Description |
|--|---|
| <code>class</code> | Display model type ('tf', 'zpk', 'ss', or 'frd'). |
| <code>hasdelay</code> | Test true if LTI model has any type of delay. |
| <code>isa</code> | Test true if LTI model is of specified class. |
| <code>isct</code> | Test true for continuous-time models. |
| <code>isdt</code> | Test true for discrete-time models. |
| <code>isempty</code> | Test true for empty LTI models. |
| <code>isproper</code> | Test true for proper LTI models. |
| <code>issiso</code> | Test true for SISO models. |
| <code>ndims</code> | Display the number of model/array dimensions. |
| <code>reshape</code> | Change the shape of an LTI array. |
| <code>size</code> | Output/input/array dimensions. Used with special syntax, <code>size</code> also returns the number of state dimensions for state-space models, and the number of frequencies in an FRD model. |

This example illustrates the use of some of these commands. See the related reference pages for more details.

```
H = tf({1 [1 -1]},{[1 0.1] [1 2 10]})
```

```
Transfer function from input 1 to output:
```

```
      1
-----
```

```
s + 0.1

Transfer function from input 2 to output:
      s - 1
-----
s^2 + 2 s + 10

class(H)

ans =
tf

size(H)
Transfer function with 2 input(s) and 1 output(s).

[ny,nu] = size(H) % Note: ny = number of outputs

ny =
    1

nu =
    2

isct(H) % Is this system continuous?

ans =
    1

isdt(H) % Is this system discrete?

ans =
    0
```

Model Dynamics

You can use functions to determine the system poles, zeros, DC gain, norms, etc. You can apply these functions to single LTI models or LTI arrays. The following table gives an overview of these commands.

| Model Dynamics | |
|----------------|--|
| covar | Covariance of response to white noise. |
| damp | Natural frequency and damping of system poles. |
| dcgain | Low-frequency (DC) gain. |
| dsort | Sort discrete-time poles by magnitude. |
| esort | Sort continuous-time poles by real part. |
| norm | Norms of LTI systems (H_2 and L_∞). |
| pole, eig | System poles. |
| pzmap | Pole/zero map. |
| zero | System transmission zeros. |

With the exception of the L_∞ norm, these commands are not supported for FRD models.

Here is an example of model analysis using some of these commands.

```
h = tf([4 8.4 30.8 60],[1 4.12 17.4 30.8 60])
```

Transfer function:

$$\frac{4 s^3 + 8.4 s^2 + 30.8 s + 60}{s^4 + 4.12 s^3 + 17.4 s^2 + 30.8 s + 60}$$

```
pole(h)
```

```
ans =
    -1.7971 + 2.2137i
    -1.7971 - 2.2137i
```

```

    -0.2629 + 2.7039i
    -0.2629 - 2.7039i
zero(h)
ans =
    -0.0500 + 2.7382i
    -0.0500 - 2.7382i
    -2.0000
dcgain(h)

ans =
     1

[ninf,fpeak] = norm(h,inf) % peak gain of freq. response

ninf =
    1.3402 % peak gain
fpeak =
    1.8537 % frequency where gain peaks

```

These functions also operate on LTI arrays and return arrays. For example, the poles of a three dimensional LTI array `sysarray` are obtained as follows.

```

sysarray = tf(rss(2,1,1,3))
Model sysarray(:,:,1,1)
=====
Transfer function:
    -0.6201 s - 1.905
    -----
    s^2 + 5.672 s + 7.405

Model sysarray(:,:,2,1)
=====
Transfer function:
    0.4282 s^2 + 0.3706 s + 0.04264
    -----
    s^2 + 1.056 s + 0.1719

Model sysarray(:,:,3,1)
=====
Transfer function:

```

$$\frac{0.621 s + 0.7567}{s^2 + 2.942 s + 2.113}$$

3x1 array of continuous-time transfer functions.

pole(sysarray)

ans(:,:,1) =

-3.6337

-2.0379

ans(:,:,2) =

-0.8549

-0.2011

ans(:,:,3) =

-1.6968

-1.2452

State-Space Realizations

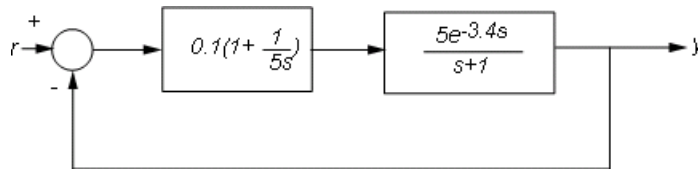
You can use the following functions to analyze, perform state coordinate transformations on, and derive canonical state-space realizations for single state-space LTI models or LTI arrays of state-space models.

| State-Space Realizations | |
|---------------------------------|---|
| canon | Canonical state-space realizations. |
| ctrb | Controllability matrix. |
| ctrbf | Controllability staircase form. |
| gram | Controllability and observability gramians. |
| obsv | Observability matrix. |
| obsvf | Observability staircase form. |
| ss2ss | State coordinate transformation. |

Analyzing Systems With Time Delays

You can use the usual analysis commands (`step`, `bode`, `margin`, ...) to analyze systems with time delays. The software makes no approximations when performing such analysis.

For example, consider the following control loop, where the plant is modeled as first-order plus dead time:



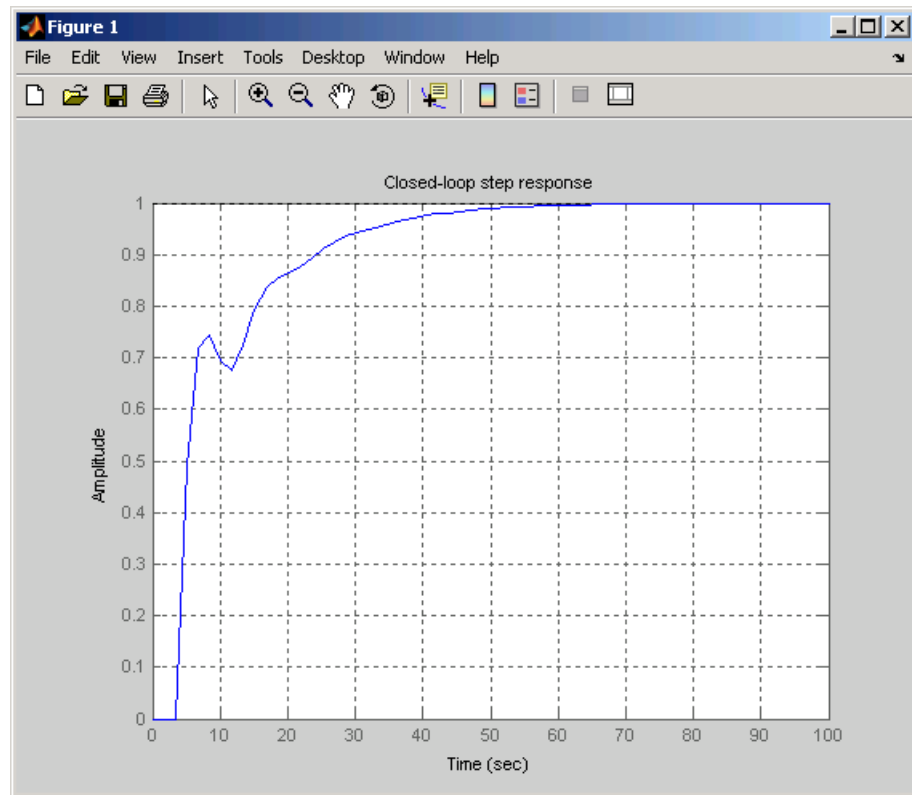
You can model the closed-loop system from r to y with the following commands:

```
s = tf('s');
P = ss(5*exp(-3.4*s)/(s+1));
C = 0.1 * (1 + 1/(5*s));
T = feedback(P*C,1)
```

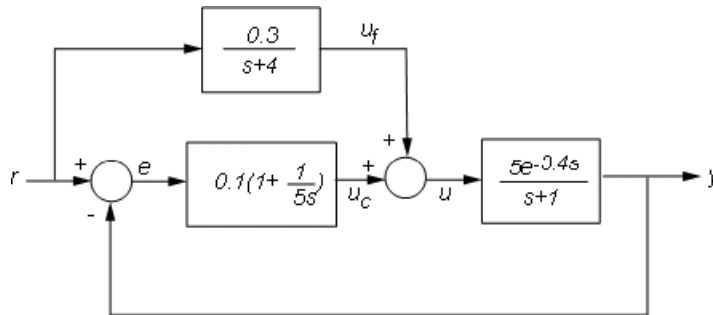
T is a state-space model with an internal delay. For more information about models with internal delays, see “Closing Feedback Loops with Time Delays” on page 2-26.

Plot the step response of T :

```
step(T)
grid, title('Closed-loop step response')
```

For more complicated interconnections, you can name the input and output signals of each block and use `connect` to automatically take care of the wiring. Suppose, for example, that you want to add feedforward to the control loop of the previous model.

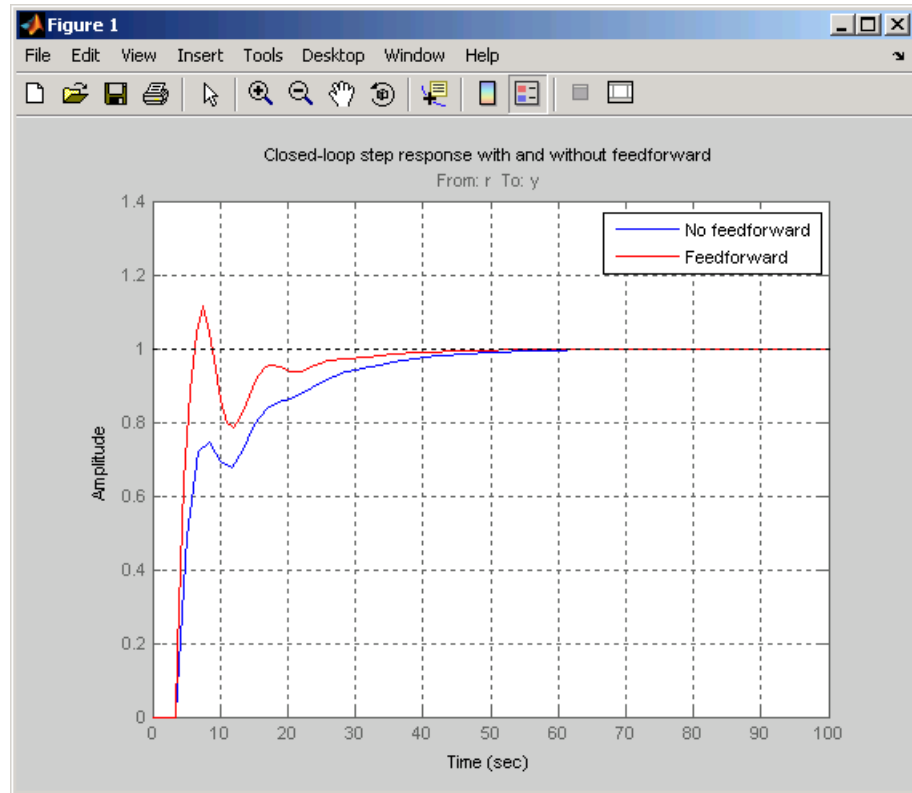


You can derive the corresponding closed-loop model Tff by

```
F = 0.3/(s+4);
P.InputName = 'u'; P.OutputName = 'y';
C.InputName = 'e'; C.OutputName = 'uc';
F.InputName = 'r'; F.OutputName = 'uf';
Sum1 = sumblk('e','r','y','+-'); % e = r-y
Sum2 = sumblk('u','uf','uc','++'); % u = uf+uc
Tff = connect(P,C,F,Sum1,Sum2,'r','y');
```

and compare its response with the feedback only design.

```
step(T,'b',Tff,'r')
legend('No feedforward','Feedforward')
grid
title('Closed-loop step response with and without feedforward')
```



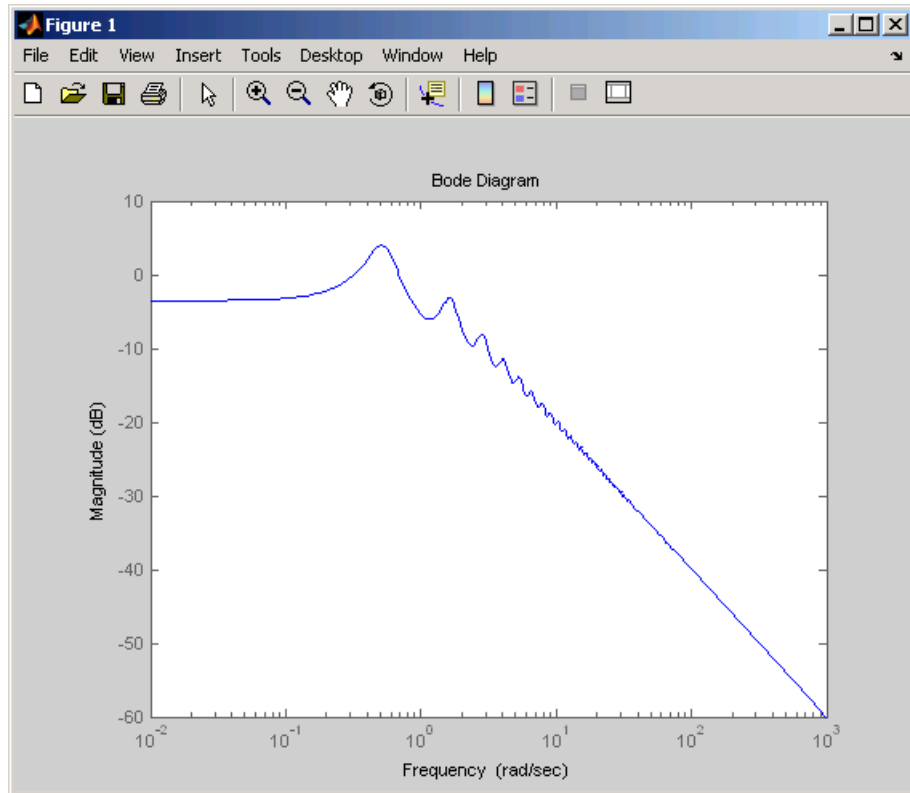
The state-space representation keeps track of the internal delays in both models.

Considerations to Keep in Mind when Analyzing Systems with Internal Time Delays

The time and frequency responses of delay systems can look odd and suspicious to those only familiar with delay-free LTI analysis. Time responses can behave chaotically, Bode plots can exhibit gain oscillations, etc. These are not software or numerical quirks but real features of such systems. Below are a few illustrations of these phenomena.

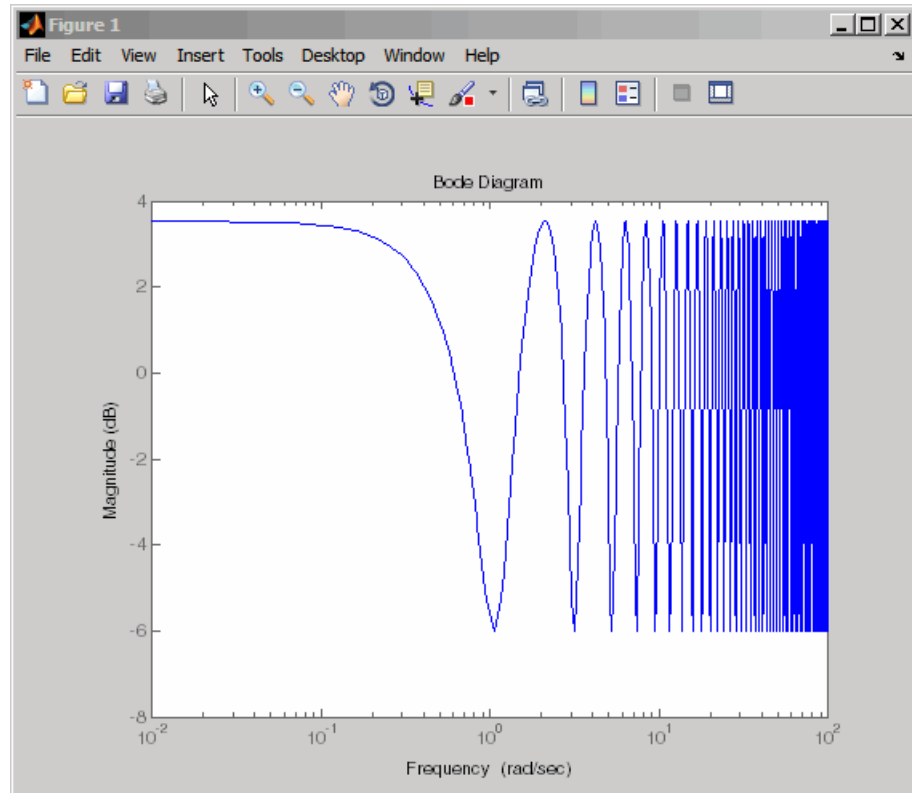
Gain ripple:

```
s=tf('s');  
G = exp(-5*s)/(s+1);  
T = feedback(ss(G),.5);  
bodemag(T)
```

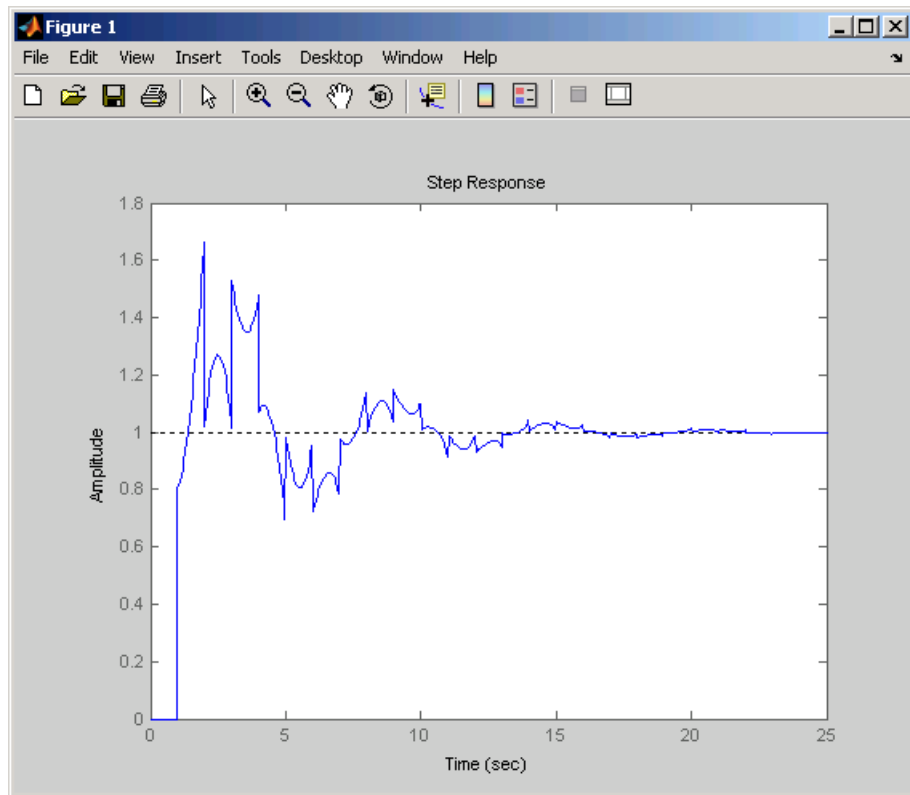


Gain oscillations:

```
G = ss(1) + 0.5 * exp(-3*s);  
bodemag(G)
```

**Jagged step response:**

```
G = exp(-s) * (0.8*s^2+s+2)/(s^2+s);  
T = feedback(ss(G),1);  
step(T)
```



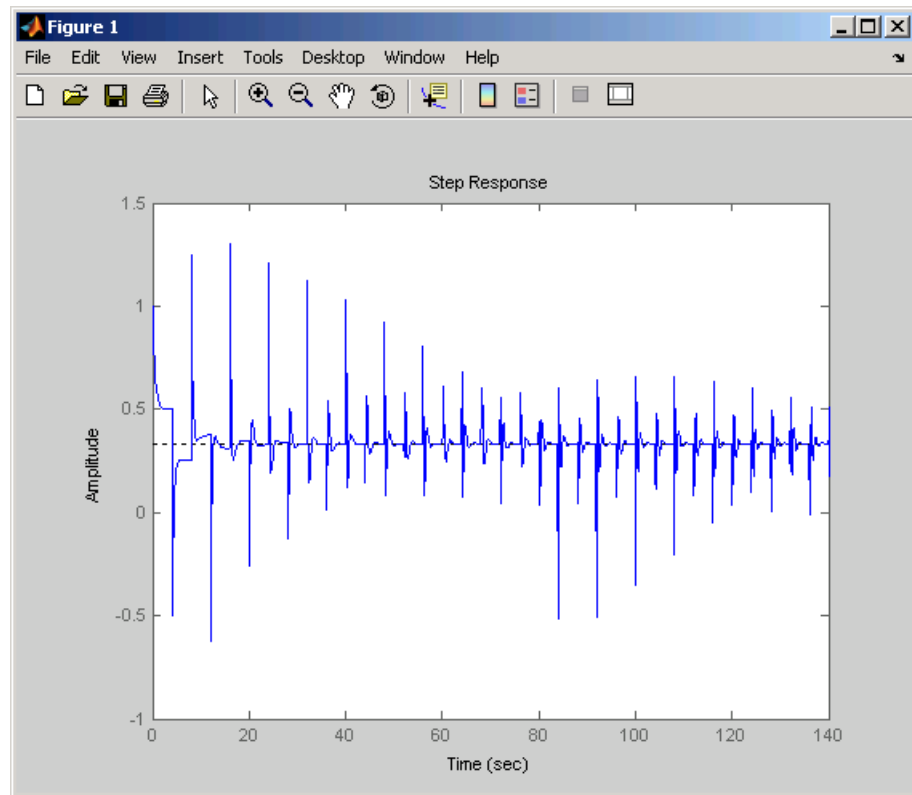
Note the rearrivals (echoes) of the initial step function.

Chaotic response:

```
G = ss(1/(s+1)) + exp(-4*s);
```

```
T = feedback(1,G);
```

```
step(T)
```



You can use Control System Toolbox tools to model and analyze these and other strange-appearing artifacts of internal delays.

Sensitivity Analysis

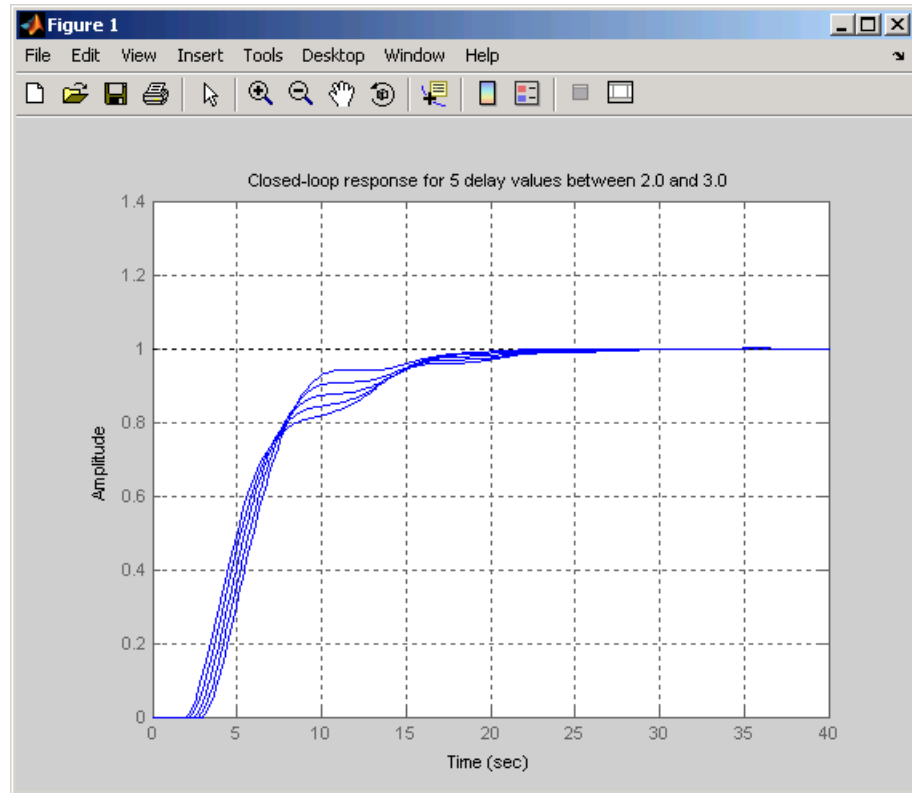
Delays are rarely known accurately, so it is often important to understand how sensitive a control system is to the delay value. Such sensitivity analysis is easily performed using LTI arrays and the `InternalDelay` property. For example, consider this notched PI control system developed in “PI Control Loop with Dead Time” from the Analyzing Control Systems with Delays demo.

```
% Create a 3rd-order plant with a PI controller and notch filter.
s = tf('s');
P = exp(-2.6*s)*(s+3)/(s^2+0.3*s+1);
C = 0.06 * (1 + 1/s);
T = feedback(ss(P*C),1)
notch = tf([1 0.2 1],[1 .8 1]);
C = 0.05 * (1 + 1/s);
Tnotch = feedback(ss(P*C*notch),1);
```

Create five models with delay values ranging from 2.0 to 3.0:

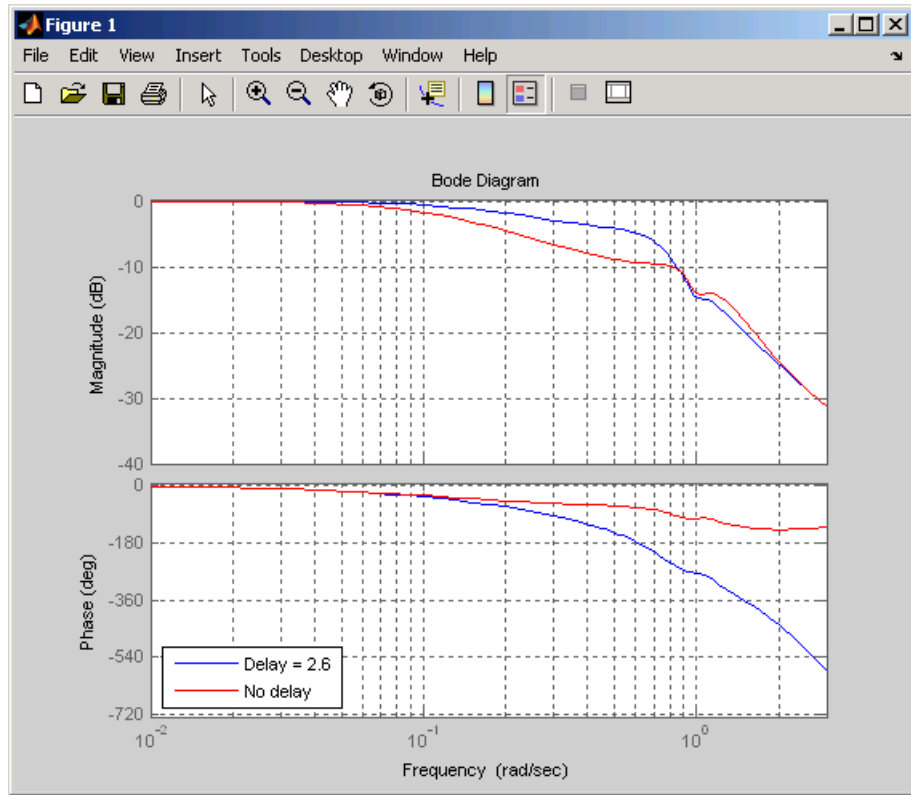
```
tau = linspace(2,3,5);           % 5 delay values
Tsens = repsys(Tnotch,[1 1 5]);  % 5 copies of Tnotch
                                   % for j=1:5
Tsens(:, :, j).InternalDelay = tau(j); % jth delay value
                                   % -> jth model end

% Use step to create an envelope plot.
step(Tsens)
grid
title('Closed-loop response for 5 delay values between 2.0 and 3.0')
```

This plot shows that uncertainty on the delay value has little effect on closed-loop characteristics. Note that while you can change the values of internal delays, you cannot change how many there are because this is part of the model structure. To eliminate some internal delays, set their value to 0 or use `pade` with order zero:

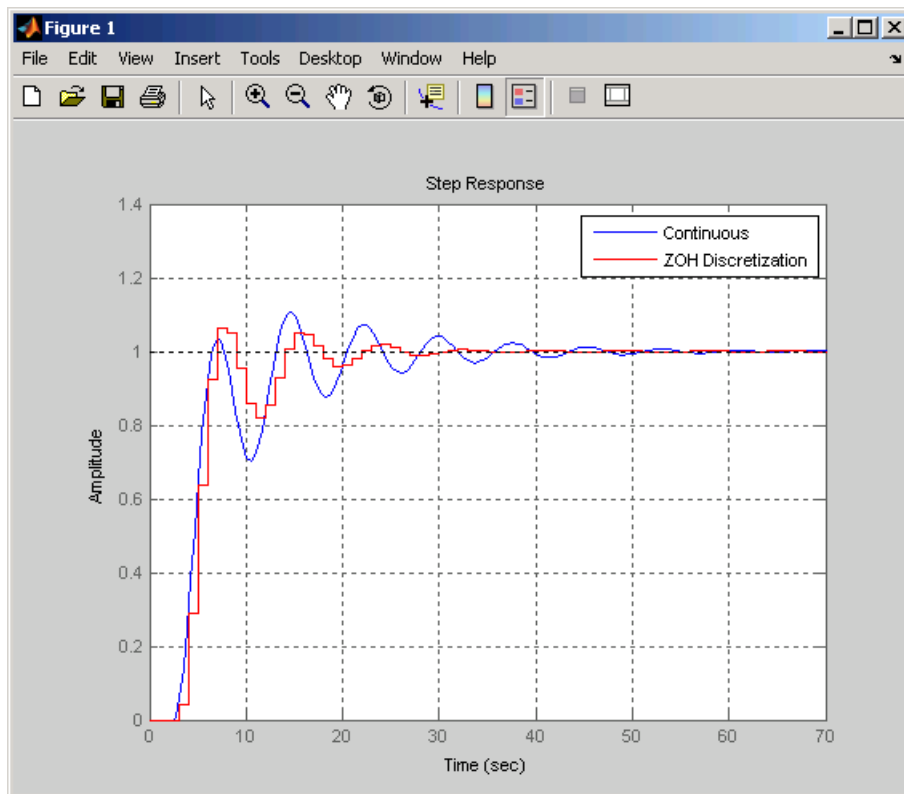
```
Tnotch0 = Tnotch;
Tnotch0.InternalDelay = 0;
bode(Tnotch, 'b', Tnotch0, 'r', {1e-2, 3})
grid, legend('Delay = 2.6', 'No delay', 'Location', 'SouthWest')
```



Discretization

You can use `c2d` to discretize continuous-time delay systems. Available methods include zero-order hold (ZOH), first-order hold (FOH), and Tustin. For models with internal delays, the ZOH discretization is not always exact, i.e., the continuous and discretized step responses may not match:

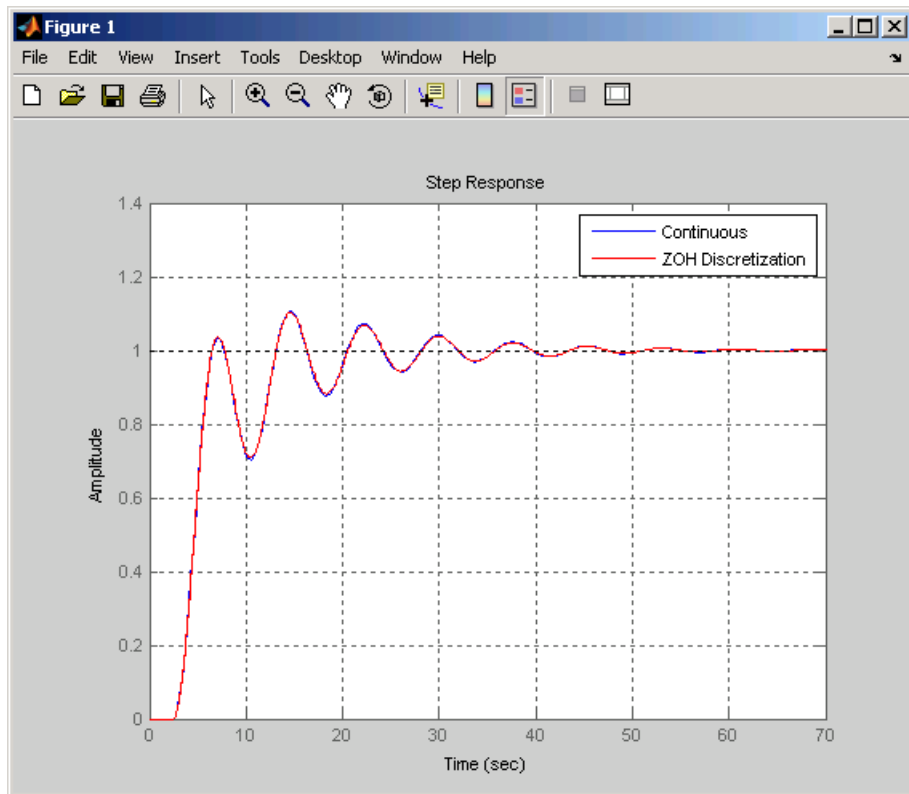
```
s = tf('s');  
P = exp(-2.6*s)*(s+3)/(s^2+0.3*s+1);  
C = 0.06 * (1 + 1/s);  
T = feedback(ss(P*C),1);  
Td = c2d(T,1); step(T,'b',Td,'r')  
grid, legend('Continuous','ZOH Discretization')
```



To correct such discretization gaps, reduce the sampling period until the continuous and discrete responses match closely:

```
Td = c2d(T,0.05); step(T,'b',Td,'r')
grid, legend('Continuous','ZOH Discretization')
```

Warning: Discretization is only approximate due to internal delays. Use faster sampling rate if discretization error is large.



Note that internal delays remain internal in the discretized model and do not inflate the model order:

```
order(Td)
```

ans =

3

For more information about discretizing systems with time delays, see “Converting Between Continuous- and Discrete-Time Representations” on page 5-24 and the c2d reference page.

Customization

- Chapter 7, “Preliminaries”
- Chapter 8, “Setting Toolbox Preferences”
- Chapter 9, “Setting Tool Preferences”
- Chapter 10, “Customizing Response Plot Properties”

Preliminaries

- “Terminology” on page 7-2
- “Property and Preferences Hierarchy” on page 7-3
- “Ways to Customize Plots” on page 7-5

Terminology

You can use the Control System Toolbox editors to set properties and preferences in the SISO Design Tool, the LTI Viewer, and in any response plots that you create from the MATLAB prompt.

Properties refer to settings that are specific to an individual response plot. This includes the following:

- Axes labels, and limits
- Data units and scales
- Plot styles, such as grids, fonts, and axes foreground colors
- Plot characteristics, such as rise time, peak response, and gain and phase margins

Preferences refers to properties that persist either

- Within a single session for a specific instance of an LTI Viewer or a SISO Design Tool
- Across Control System Toolbox sessions

The former are called *tool preferences*, the latter *toolbox preferences*.

Property and Preferences Hierarchy

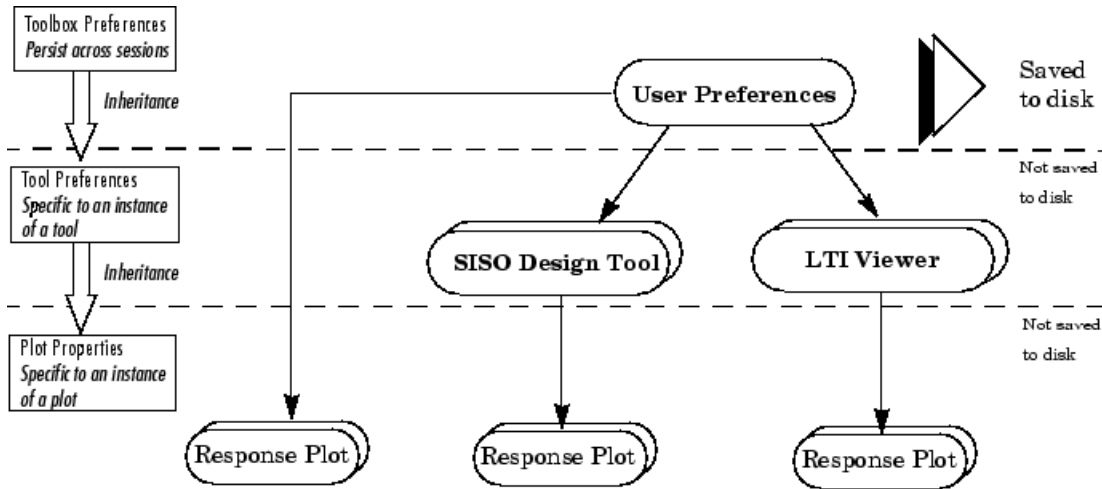
You can use three graphical user interfaces (GUIs) to control over the visualization of time and frequency plots generated by the toolbox:

- Toolbox Preferences
- Tool Preferences
- Plot Properties

Preferences refer to global options that you can save from session to session or to any LTI Viewer or SISO Design Tool that you open during a single session. *Properties* are options that apply only to the current window. This section gives an overview of the three GUIs. For more information, see the following topics:

- Chapter 8, “Setting Toolbox Preferences”
- Chapter 9, “Setting Tool Preferences”
- Chapter 10, “Customizing Response Plot Properties”

Although you can set plot properties in any response plot, you can use the Toolbox Preferences Editor to set properties for any response plot that you generate. This figure shows the inheritance hierarchy from toolbox preference to plot properties.



Preference and Property Inheritance Hierarchy

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

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

Ways to Customize Plots

You can customize your plots by changing plot properties. For example, you can change the plot units. The following table describes ways that you can customize plots.

| To change plot properties of | For more information, see |
|---|---|
| A single plot, directly from the plot | <ul style="list-style-type: none"> • “Customizing Response Plots Using the Response Plots Property Editor” on page 10-3 and “Customizing Response Plots Using Plot Tools” on page 10-19 for response plots • Chapter 9, “Setting Tool Preferences” for LTI Viewer plots and Graphical Tuning Window plots |
| A single plot or many plots, programmatically from the command line | “Customizing Response Plots from the Command Line” on page 10-23 |
| All Control System Toolbox plots (changes apply globally to all plot types and persist from session to session) | “Toolbox Preferences Editor” on page 8-2 |

Setting Toolbox Preferences

- “Toolbox Preferences Editor” on page 8-2
- “Units Pane” on page 8-4
- “Style Pane” on page 8-7
- “Options Pane” on page 8-8
- “SISO Tool Pane” on page 8-9

Toolbox Preferences Editor

| In this section... |
|--|
| “Overview of the Toolbox Preferences Editor” on page 8-2 |
| “Opening the Toolbox Preferences Editor” on page 8-2 |

Overview of the Toolbox Preferences Editor

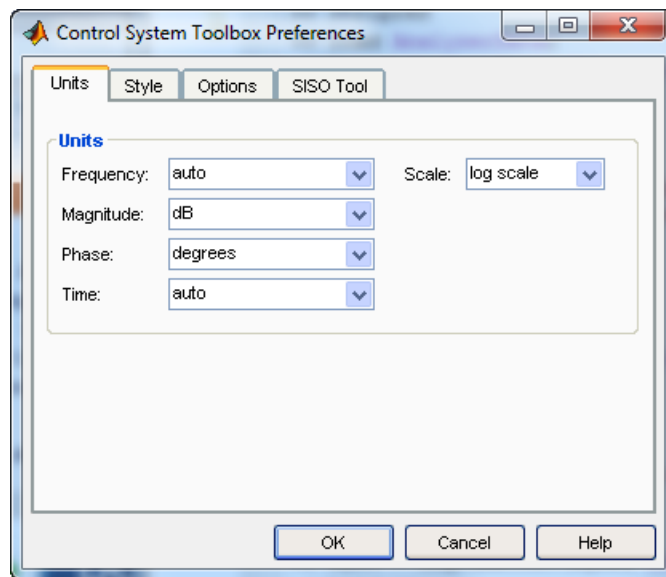
The Toolbox Preferences editor allows you to set plot preferences that will persist from session to session. This is the highest level shown in “Property and Preferences Hierarchy” on page 7-3.

Opening the Toolbox Preferences Editor

To open the Toolbox Preferences editor, select **Toolbox Preferences** from the **File** menu of the LTI Viewer or the SISO Design Tool. Alternatively, you can type

```
ctrlpref
```

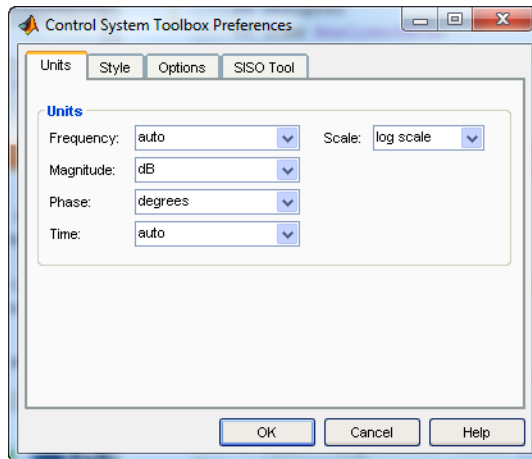
at the MATLAB prompt.



Control System Toolbox™ Preferences Editor

- “Units Pane” on page 8-4
- “Style Pane” on page 8-7
- “Options Pane” on page 8-8
- “SISO Tool Pane” on page 8-9

Units Pane



Use the **Units** pane to set preferences for the following:

- **Frequency**

The default auto option uses $\text{rad}/\text{TimeUnit}$ as the frequency units relative to the system time units, where `TimeUnit` is the system time units specified in the `TimeUnit` property of the system on frequency-domain plots. For multiple systems with different time units, the units of the first system is used.

For the frequency axis, you can select logarithmic or linear scales.

Other Frequency Units Options

- 'Hz'
- 'rad/s'
- 'rpm'
- 'kHz'
- 'MHz'
- 'GHz'
- 'rad/nanosecond'

- 'rad/microsecond'
- 'rad/millisecond'
- 'rad/minute'
- 'rad/hour'
- 'rad/day'
- 'rad/week'
- 'rad/month'
- 'rad/year'
- 'cycles/nanosecond'
- 'cycles/microsecond'
- 'cycles/millisecond'
- 'cycles/hour'
- 'cycles/day'
- 'cycles/week'
- 'cycles/month'
- 'cycles/year'
- **Magnitude** — Decibels (dB) or absolute value (abs)
- **Phase** — Degrees or radians
- **Time**

The default auto option uses the time units specified in the `TimeUnit` property of the system on the time- and frequency-domain plots. For multiple systems with different time units, the units of the first system is used.

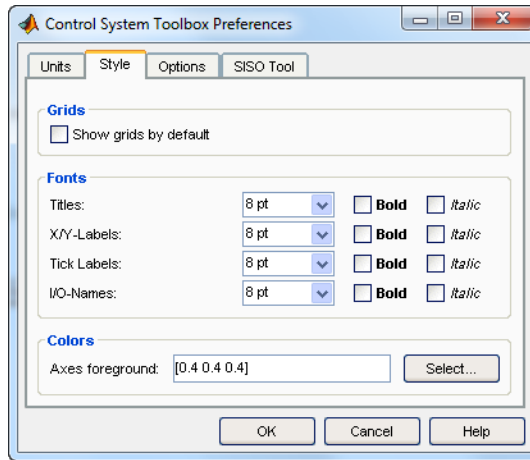
Other Time Units Options

- 'nanoseconds'
- 'microseconds'
- 'milliseconds'

- 'seconds'
- 'minutes'
- 'hours'
- 'days'
- 'weeks'
- 'months'
- 'years'

Style Pane

Use the **Style** pane to toggle grid visibility and set font preferences and axes foreground colors for all plots you create. This figure shows the Style pane.



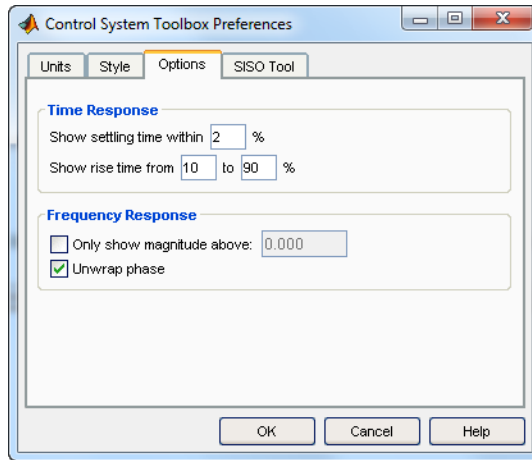
You have the following choices:

- **Grid** — Activate grids by default in new plots.
- **Fonts** — Set the font size, weight (bold), and angle (italic).
- **Colors** — Specify the color vector to use for the axes foreground, which includes the X-Y axes, grid lines, and tick labels. Use a three-element vector to represent red, green, and blue (RGB) values. Vector element values can range from 0 to 1.

If you do not want to specify RGB values numerically, click the **Select** button to open the **Select Colors** dialog box. See “Select colors” on page 9-13 for more information.

Options Pane

The Options pane has selections for time responses and frequency responses. This figure shows the Options pane with default settings.

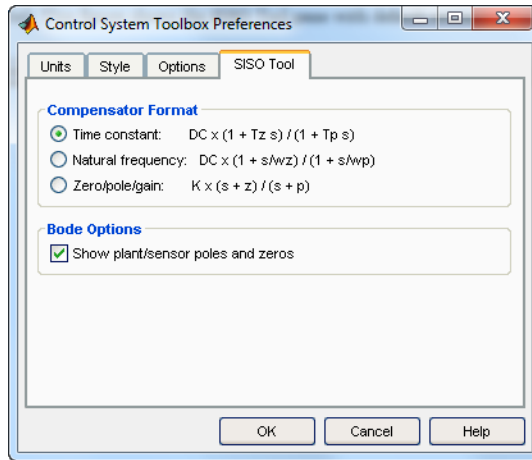


The following are the available options for the Options pane:

- **Time Response:**
 - Show settling time within $xx\%$ — You can set the threshold of the settling time calculation to any percentage from 0 to 100%. The default is 2%.
 - Specify rise time from $xx\%$ to $yy\%$ — The standard definition of rise time is the time it takes the signal to go from 10% to 90% of the final value. You can choose any percentages you like (from 0% to 100%), provided that the first value is smaller than the second.
- **Frequency Response:**
 - Only show magnitude above xx —Specify a lower limit for magnitude values in response plots so that you can focus on a region of interest.
 - Unwrap phase—By default, the phase is unwrapped. Wrap the phrase by clearing this box. If the phase is wrapped, all phase values are shifted such that their equivalent value displays in the range $[-180^\circ, 180^\circ)$.

SISO Tool Pane

The SISO Tool pane has settings for the SISO Design Tool. This figure shows the SISO Tool pane with default settings.



You can make the following selections:

- **Compensator Format** — Select the time constant, natural frequency, or zero/pole/gain format. The time constant format is a factorization of the compensator transfer function of the form

$$DC \times \frac{(1 + Tz_1 s)}{(1 + Tp_1 s)} \dots$$

where DC is compensator DC gain, Tz_1, Tz_2, \dots , are the zero time constants, and Tp_1, Tp_2, \dots , are the pole time constants.

The natural frequency format is

$$DC \times \frac{(1 + s/\omega_{z_1})}{(1 + s/\omega_{p_1})} \dots$$

where DC is compensator DC gain, ω_{z1} , and ω_{z2} , ... and ω_{p1} , ω_{p2} , ..., are the natural frequencies of the zeros and poles, respectively.

The zero/pole/gain format is

$$K \times \frac{(s + z_1)}{(s + p_1)}$$

where K is the overall compensator gain, and z_1, z_2, \dots and p_1, p_2, \dots , are the zero and pole locations, respectively.

- **Bode Options** — By default, the SISO Design Tool shows the plant and sensor poles and zeros as blue x's and o's, respectively. Clear this box to eliminate the plant's poles and zeros from the Bode plot. Note that the compensator poles and zeros (in red) will still appear.

Setting Tool Preferences

- “Introduction” on page 9-2
- “LTI Viewer Preferences Editor” on page 9-3
- “Graphical Tuning Window Preferences Editor” on page 9-9

Introduction

Both the LTI Viewer and the Graphical Tuning Window have Tool Preferences Editors. These editors comprise the middle layer of “Property and Preferences Hierarchy” on page 7-3.

Both editors allow you to set default characteristics for specific instances of LTI Viewers and Graphical Tuning windows. If you open a new instance of either, each defaults to the characteristics specified in the Toolbox Preferences editor.

LTI Viewer Preferences Editor

In this section...

“Opening the LTI Viewer Preference Editor” on page 9-3

“Units Pane” on page 9-4

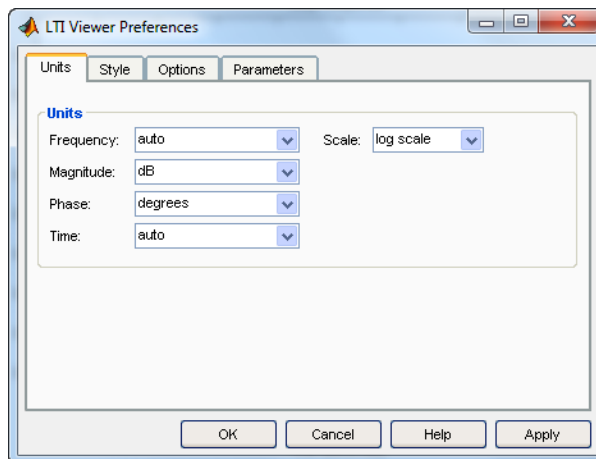
“Style Pane” on page 9-6

“Options Pane” on page 9-7

“Parameters Pane” on page 9-8

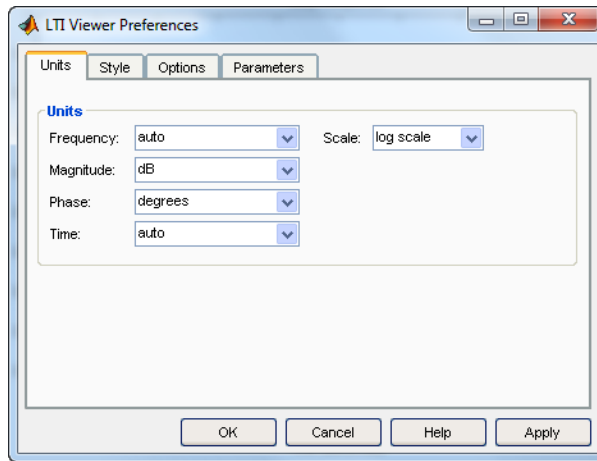
Opening the LTI Viewer Preference Editor

Select **Viewer Preferences** under the **Edit** menu of the LTI Viewer to open the **LTI Viewer Preferences** editor, which is a tool for customizing various LTI Viewer properties, including units, fonts, and various other viewer characteristics. This figure shows the editor open to its first pane.



- “Units Pane” on page 9-4
- “Style Pane” on page 9-6
- “Options Pane” on page 9-7
- “Parameters Pane” on page 9-8

Units Pane



You can select the following on the **Units** pane:

- **Frequency**

The default auto option uses $\text{rad}/\text{TimeUnit}$ as the frequency units relative to the system time units, where `TimeUnit` is the system time units specified in the `TimeUnit` property of the system on frequency-domain plots. For multiple systems with different time units, the units of the first system is used.

For the frequency axis, you can select logarithmic or linear scales.

Other Frequency Units Options

- 'Hz '
- 'rad/s '
- 'rpm '
- 'kHz '
- 'MHz '
- 'GHz '
- 'rad/nanosecond '

- 'rad/microsecond'
- 'rad/millisecond'
- 'rad/minute'
- 'rad/hour'
- 'rad/day'
- 'rad/week'
- 'rad/month'
- 'rad/year'
- 'cycles/nanosecond'
- 'cycles/microsecond'
- 'cycles/millisecond'
- 'cycles/hour'
- 'cycles/day'
- 'cycles/week'
- 'cycles/month'
- 'cycles/year'
- **Magnitude** — Decibels (dB) or absolute value (abs)
- **Phase** — Degrees or radians
- **Time**

The default auto option uses the time units specified in the `TimeUnit` property of the system on the time- and frequency-domain plots. For multiple systems with different time units, the units of the first system is used.

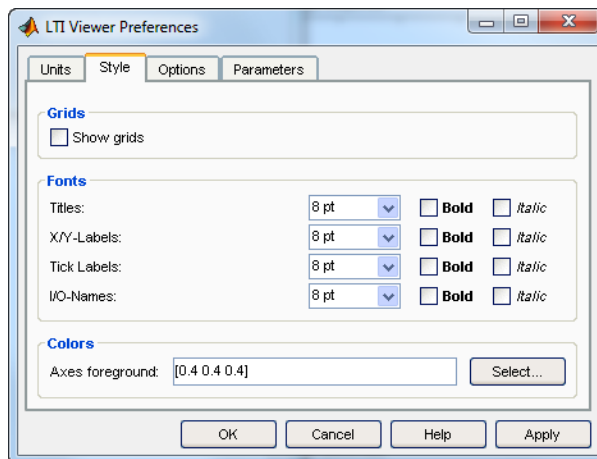
Other Time Units Options

- 'nanoseconds'
- 'microseconds'
- 'milliseconds'

- 'seconds'
- 'minutes'
- 'hours'
- 'days'
- 'weeks'
- 'months'
- 'years'

Style Pane

Use the **Style** pane to toggle grid visibility and set font preferences and axes foreground colors for all plots in the LTI Viewer. This figure shows the Style pane.



You have the following choices:

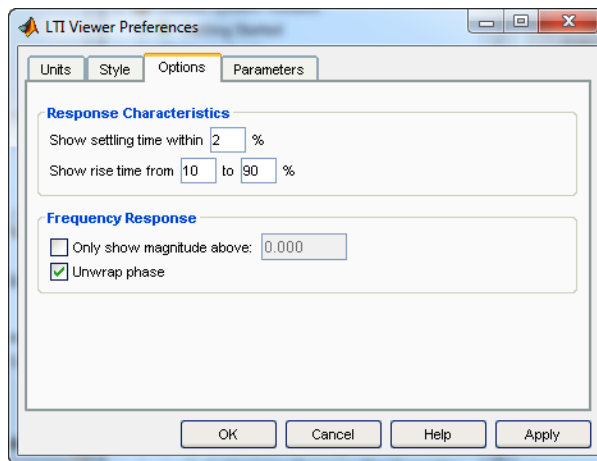
- **Grid** — Activate grids for all plots in the LTI Viewer
- **Fonts** — Set the font size, weight (bold), and angle (italic)
- **Colors** — Specify the color vector to use for the axes foreground, which includes the X-Y axes, grid lines, and tick labels. Use a three-element

vector to represent red, green, and blue (RGB) values. Vector element values can range from 0 to 1.

- If you do not want to specify the RGB values numerically, press the **Select** button to open the **Select Colors** window. See “Select colors” on page 9-13 for more information.

Options Pane

The Options pane has selections for time responses and frequency responses.



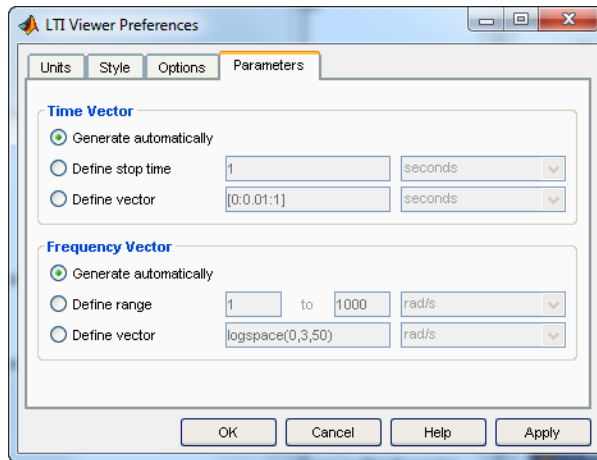
The following choices are available:

- **Time Response:**
 - Show settling time within $xx\%$ — You can set the threshold of the settling time calculation to any percentage from 0 to 100%. The default is 2%.
 - Specify rise time from $xx\%$ to $yy\%$ — The standard definition of rise time is the time it takes the signal to go from 10% to 90% of the final value. You can choose any percentages you like (from 0% to 100%), provided that the first value is smaller than the second.
- **Frequency Response:**
 - Only show magnitude above xx —Specify a lower limit for magnitude values in response plots so that you can focus on a region of interest.

- Unwrap phase—By default, the phase is unwrapped. Wrap the phase by clearing this box. If the phase is wrapped, all phase values are shifted such that their equivalent value displays in the range $[-180^\circ, 180^\circ)$.

Parameters Pane

Use the **Parameters** pane, shown below, to specify input vectors for time and frequency simulation.



The defaults are to generate time and frequency vectors for your plots automatically. You can, however, override the defaults as follows:

- **Time Vector:**
 - Define stop time — Specify the final time value for your simulation
 - Define vector — Specify the time vector manually using equal-sized time steps
- **Frequency Vector:**
 - Define range — Specify the bandwidth of your response. Whether it's in rad/sec or Hz depends on the selection you made in the Units pane.
 - Define vector — Specify the vector for your frequency values. Any real, positive, strictly monotonically increasing vector is valid.

Graphical Tuning Window Preferences Editor

In this section...

“Opening the Graphical Tuning Window Preferences Editor” on page 9-9

“Units Pane” on page 9-10

“Time Delays Pane” on page 9-11

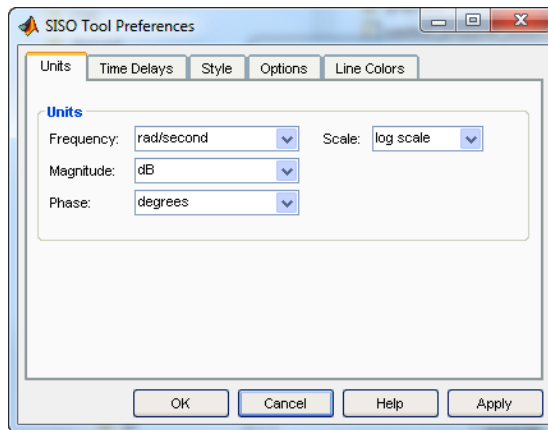
“Style Pane” on page 9-12

“Options Pane” on page 9-15

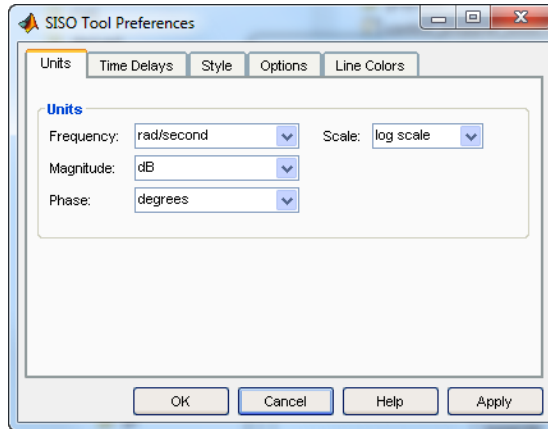
“Line Colors Pane” on page 9-16

Opening the Graphical Tuning Window Preferences Editor

To open the **SISO Tool Preferences** editor, select **SISO Tool Preferences** from the **Edit** menu of the Graphical Tuning window. This window opens.



Units Pane



The **Units** pane has settings for the following units:

- **Frequency**

The default units are rad/second.

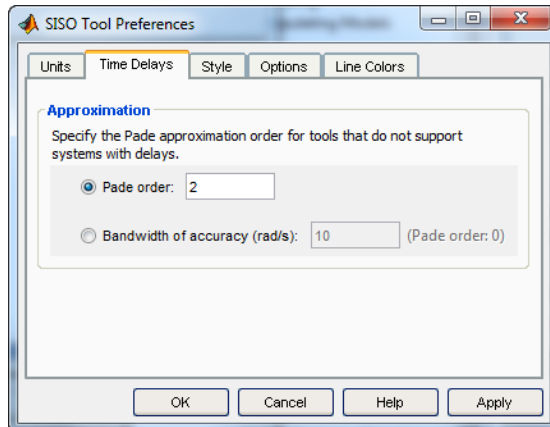
Other Frequency Units Options

- ' Hz '
- ' rad / s '
- ' rpm '
- ' kHz '
- ' MHz '
- ' GHz '
- ' rad/nanosecond '
- ' rad/microsecond '
- ' rad/millisecond '
- ' rad/minute '
- ' rad/hour '
- ' rad/day '

- 'rad/week'
 - 'rad/month'
 - 'rad/year'
 - 'cycles/nanosecond'
 - 'cycles/microsecond'
 - 'cycles/millisecond'
 - 'cycles/hour'
 - 'cycles/day'
 - 'cycles/week'
 - 'cycles/month'
 - 'cycles/year'
- **Magnitude** — Decibels (dB) or absolute value (abs)
 - **Phase** — Degrees or radians

For frequency and magnitude axes, you can select logarithmic or linear scales.

Time Delays Pane



In the Time Delays pane, specify the order for Padé approximations of delays in your system as either:

- Actual Padé order
- Bandwidth of accuracy (rad/s) — The highest frequency at which the approximated response matches the actual system. The software computes and displays the corresponding Padé order.

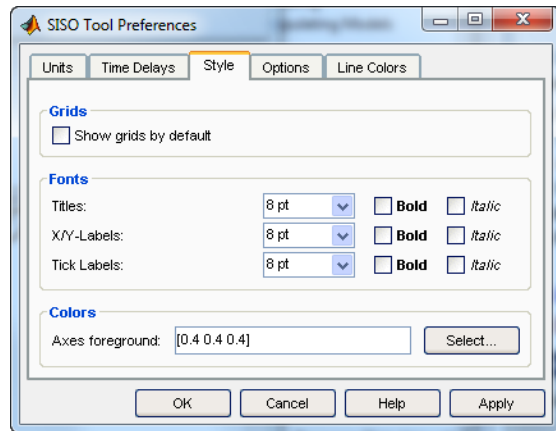
The following compensator design tools do not support systems with exact time delays. If your system has exact continuous-time delays, these tools automatically compute a Padé approximation of the delays. In this case, you receive a notification.

- Root locus
- Pole-zero
- PID automated tuning
- IMC automated tuning
- LQG automated tuning
- Loop shaping automated tuning

Tip To determine if a certain Padé order gives a good approximation of your system, use the `pade` command to approximate your system. Then, compare a plot of the two systems using the `bode` command.

Style Pane

Use the Style pane to toggle grid visibility and set font preferences and axes foreground colors for all plots in the Graphical Tuning Window. This figure shows the Style pane.



Grids Panel

Select the box to activate grids for all plots in the GRAPHICAL Tuning Window

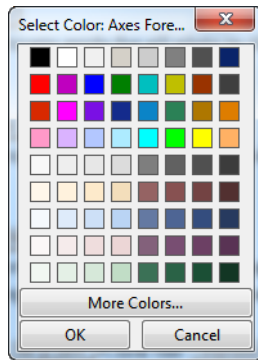
Fonts Panel

Set the font size, weight (bold), and angle (italic) by using the menus and check boxes.

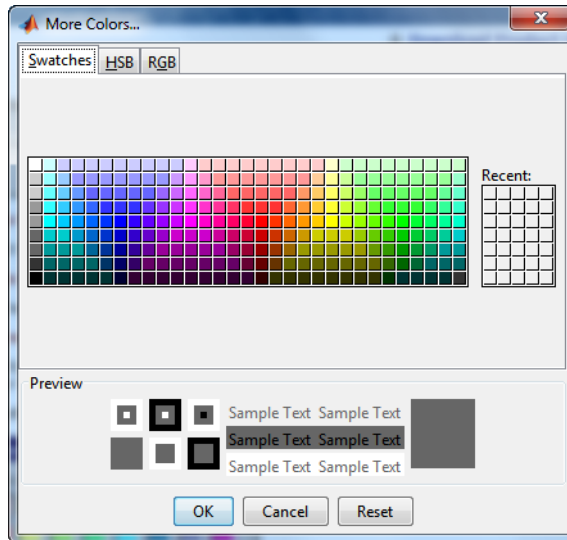
Colors Panel

Specify the color vector to use for the axes foreground, which includes the X-Y axes, grid lines, and tick labels. Use a three-element vector to represent red, green, and blue (RGB) values. Vector element values can range from 0 to 1.

Select colors. Click the **Select** button to open the Select Color window for the axes foreground.



You can use this window to choose axes foreground colors without having to set RGB (red-green-blue) values numerically. To make your selections, click on the colored rectangles and press OK. If you want a broader range of colors, click the **More Colors** button. This opens the More Colors window, as shown in the following figure.

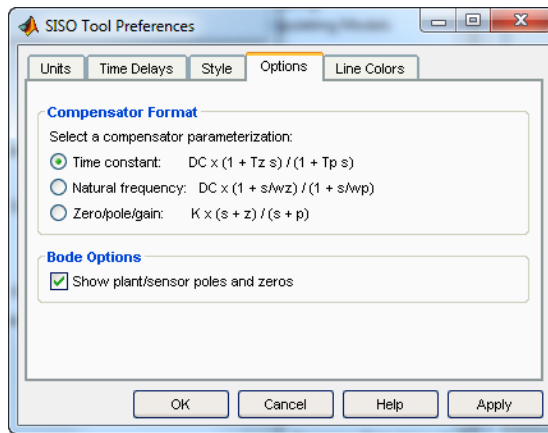


Click on the array of color swatches to select a color. Alternatively, click the HSB tab to use the Hue Saturation Brightness (HSB) color picker. Or, click the RGB tab to enter numeric RGB values.

When you select a color, the **Preview** section of the window changes to preview your selection. Click **OK** to accept the choice. Click **Reset** to reset the selection.

Options Pane

The Options pane, shown below, has selections for compensator format and Bode diagrams.



You can make the following selections:

- **Compensator Format** — Select the time constant, natural frequency, or zero/pole/gain format. The time constant format is a factorization of the compensator transfer function of the form

$$DC \times \frac{(1 + Tz_1 s)}{(1 + Tp_1 s)} \dots$$

where DC is compensator DC gain, Tz_1, Tz_2, \dots , are the zero time constants, and Tp_1, Tp_2, \dots , are the pole time constants.

The natural frequency format is

$$DC \times \frac{(1 + s/\omega_{z1}) \dots}{(1 + s/\omega_{p1}) \dots}$$

where DC is compensator DC gain, ω_{z1} , and ω_{z2} , ... and ω_{p1} , ω_{p2} , ..., are the natural frequencies of the zeros and poles, respectively.

The zero/pole/gain format is

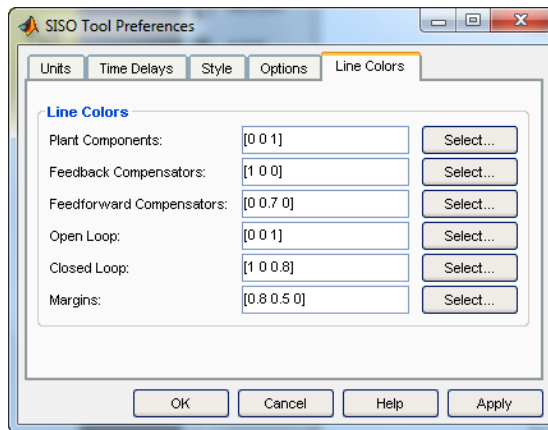
$$K \times \frac{(s + z_1)}{(s + p_1)}$$

where K is the overall compensator gain, and z_1 , z_2 , ... and p_1 , p_2 , ..., are the zero and pole locations, respectively.

- **Bode Options** — By default, the GRAPHICAL Tuning Window shows the plant and sensor poles and zeros as blue x's and o's, respectively. Clear this check box to eliminate the plant's poles and zeros from the Bode plot. Note that the compensator poles and zeros (in red) will still appear.

Line Colors Pane

The Line Colors pane, shown below, has selections for specify the colors of the lines in the response plots of the Graphical Tuning Window.



To change the colors of plot lines associated with parts of your model, specify a three-element vector to represent red, green, and blue (RGB) values. Vector element values can range from 0 to 1.

If you do not want to specify the RGB values numerically, click the **Select** button to open the Select Color window. See “Select colors” on page 9-13 for more information.

Customizing Response Plot Properties

- “Introduction” on page 10-2
- “Customizing Response Plots Using the Response Plots Property Editor” on page 10-3
- “Customizing Response Plots Using Plot Tools” on page 10-19
- “Customizing Response Plots from the Command Line” on page 10-23
- “Customizing Plots Inside the SISO Design Tool” on page 10-48

Introduction

The lowest level of the “Property and Preferences Hierarchy” on page 7-3 is setting response plot properties. This means that any property you set for a given plot will only affect that plot.

Customizing Response Plots Using the Response Plots Property Editor

In this section...

“Opening the Property Editor” on page 10-3

“Overview of Response Plots Property Editor” on page 10-4

“Labels Pane” on page 10-6

“Limits Pane” on page 10-6

“Units Pane” on page 10-7

“Style Pane” on page 10-16

“Options Pane” on page 10-17

“Editing Subplots Using the Property Editor” on page 10-18

Opening the Property Editor

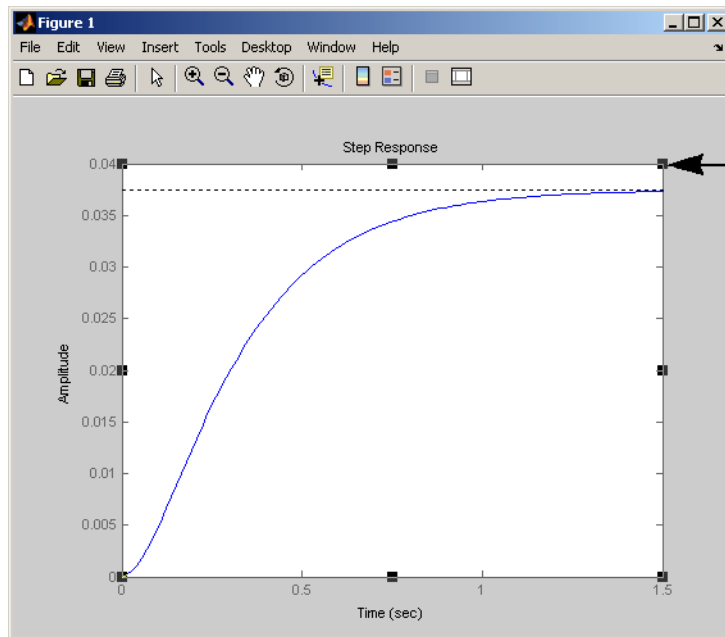
After you create a response plot, there are two ways to open the Property Editor:

- Double-click in the plot region
- Select **Properties** from the right-click menu

Before looking at the Property Editor, open a step response plot using these commands.

```
load ltiexamples
step(sys_dc)
```

This creates a step plot. Select **Properties** from the right-click menu. Note that when you open the **Property Editor**, a set of black squares appear around the step response, as this figure shows:

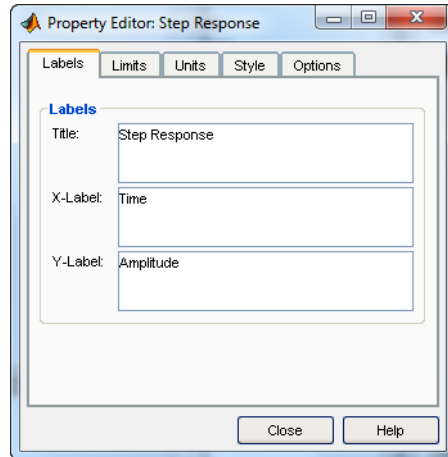


The black squares surrounding the plot indicate that the Property Editor is active for this plot.

SISO System Step Response

Overview of Response Plots Property Editor

This figure shows the Property Editor dialog box for a step response.



The Property Editor for Step Response

In general, you can change the following properties of response plots. Note that only the **Labels** and **Limits** panes are available when using the **Property Editor** with Simulink® Design Optimization™ software.

- Titles and X- and Y-labels in the **Labels** pane.
- Numerical ranges of the X and Y axes in the **Limits** pane.
- Units where applicable (e.g., rad/s to Hertz) in the **Units** pane.

If you cannot customize units, as is the case with step responses, the Property Editor will display that no units are available for the selected plot.

- Styles in the **Styles** pane.

You can show a grid, adjust font properties, such as font size, bold and italics, and change the axes foreground color

- Change options where applicable in the **Options** pane.

These include peak response, settling time, phase and gain margins, etc. Plot options change with each plot response type. The Property Editor displays only the options that make sense for the selected response plot.

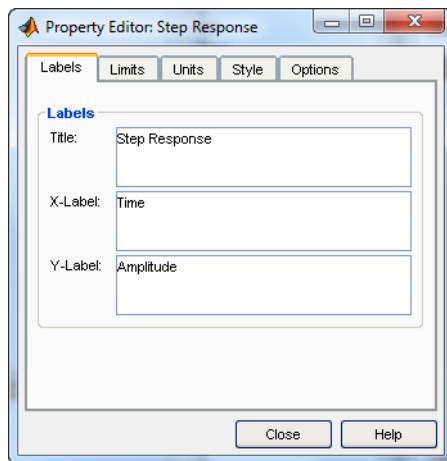
For example, phase and gain margins are not available for step responses.

As you make changes in the Property Editor, they display immediately in the response plot. Conversely, if you make changes in a plot using right-click

menus, the Property Editor for that plot automatically updates. The Property Editor and its associated plot are dynamically linked.

Labels Pane

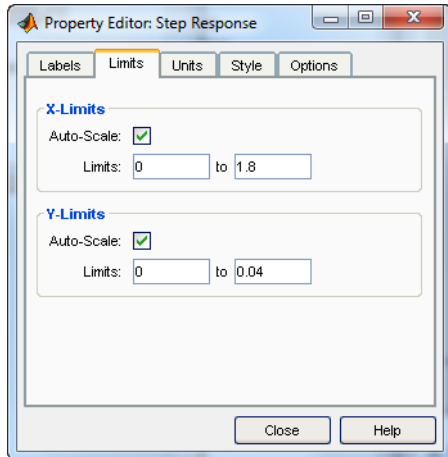
To specify new text for plot titles and axis labels, type the new string in the field next to the label you want to change. Note that the label changes immediately as you type, so you can see how the new text looks as you are typing.



Limits Pane

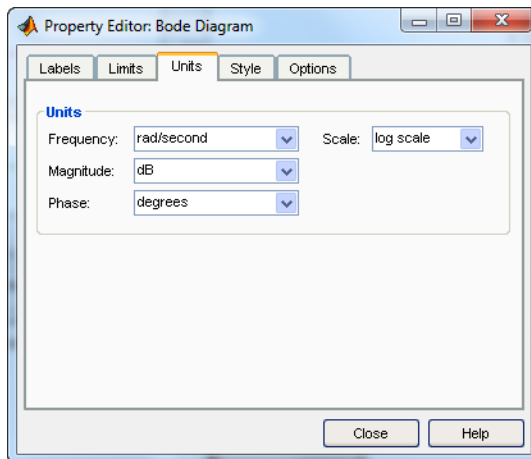
Default values for the axes limits make sure that the maximum and minimum x and y values are displayed. If you want to override the default settings, change the values in the Limits fields. The **Auto-Scale** box automatically clears if you click a different field. The new limits appear immediately in the response plot.

To reestablish the default values, select the **Auto-Scale** box again.



Units Pane

You can use the **Units** pane to change units in your response plot. The contents of this pane depend on the response plot associated with the editor.



The following table lists the options available for the response objects. Use the menus to toggle between units.

Optional Unit Conversions for Response Plots

| Response Plot | Unit Conversions |
|----------------------------|---|
| Bode and Bode Magnitude | <ul style="list-style-type: none"> • Frequency <p>By default, shows rad/TimeUnit where TimeUnit is the system time units specified in the TimeUnit property of the input system.</p> <p>Frequency Units Options</p> <ul style="list-style-type: none"> - 'Hz' - 'rad/s' - 'rpm' - 'kHz' - 'MHz' - 'GHz' - 'rad/nanosecond' - 'rad/microsecond' - 'rad/millisecond' - 'rad/minute' - 'rad/hour' - 'rad/day' - 'rad/week' - 'rad/month' - 'rad/year' - 'cycles/nanosecond' - 'cycles/microsecond' - 'cycles/millisecond' - 'cycles/hour' - 'cycles/day' |

Optional Unit Conversions for Response Plots (Continued)

| Response Plot | Unit Conversions |
|---------------|--|
| | <ul style="list-style-type: none"> - 'cycles/week' - 'cycles/month' - 'cycles/year' • Frequency scale is logarithmic or linear. • Magnitude in decibels (dB) or the absolute value • Phase in degrees or radians |
| Impulse | <ul style="list-style-type: none"> • Time. <p>By default, shows the system time units specified in the TimeUnit property of the input system.</p> <p>Time Units Options</p> <ul style="list-style-type: none"> - 'nanoseconds' - 'microseconds' - 'milliseconds' - 'seconds' - 'minutes' - 'hours' - 'days' - 'weeks' - 'months' - 'years' |

Optional Unit Conversions for Response Plots (Continued)

| Response Plot | Unit Conversions |
|---------------|---|
| Nichols Chart | <ul style="list-style-type: none"> • Frequency <p>By default, shows rad/TimeUnit where TimeUnit is the system time units specified in the TimeUnit property of the input system.</p> <p>Frequency Units Options</p> <ul style="list-style-type: none"> - 'Hz' - 'rad/s' - 'rpm' - 'kHz' - 'MHz' - 'GHz' - 'rad/nanosecond' - 'rad/microsecond' - 'rad/millisecond' - 'rad/minute' - 'rad/hour' - 'rad/day' - 'rad/week' - 'rad/month' - 'rad/year' - 'cycles/nanosecond' - 'cycles/microsecond' - 'cycles/millisecond' - 'cycles/hour' - 'cycles/day' |

Optional Unit Conversions for Response Plots (Continued)

| Response Plot | Unit Conversions |
|-----------------|---|
| | <ul style="list-style-type: none"> - 'cycles/week' - 'cycles/month' - 'cycles/year' • Phase in degrees or radians |
| Nyquist Diagram | <ul style="list-style-type: none"> • Frequency <p>By default, shows rad/TimeUnit where TimeUnit is the system time units specified in the TimeUnit property of the input system.</p> <p>Frequency Units Options</p> <ul style="list-style-type: none"> - 'Hz' - 'rad/s' - 'rpm' - 'kHz' - 'MHz' - 'GHz' - 'rad/nanosecond' - 'rad/microsecond' - 'rad/millisecond' - 'rad/minute' - 'rad/hour' - 'rad/day' - 'rad/week' - 'rad/month' - 'rad/year' |

Optional Unit Conversions for Response Plots (Continued)

| Response Plot | Unit Conversions |
|---------------|---|
| | <ul style="list-style-type: none"> - 'cycles/nanosecond' - 'cycles/microsecond' - 'cycles/millisecond' - 'cycles/hour' - 'cycles/day' - 'cycles/week' - 'cycles/month' - 'cycles/year' |
| Pole/Zero Map | <ul style="list-style-type: none"> • Time. By default, shows the system time units specified in the TimeUnit property of the input system. Time Units Options - 'nanoseconds' - 'microseconds' - 'milliseconds' - 'seconds' - 'minutes' - 'hours' - 'days' - 'weeks' - 'months' - 'years' • Frequency |

Optional Unit Conversions for Response Plots (Continued)

| Response Plot | Unit Conversions |
|---------------|---|
| | <p>By default, shows rad/TimeUnit where TimeUnit is the system time units specified in the TimeUnit property of the input system.</p> <p>Frequency Units Options</p> <ul style="list-style-type: none"> - 'Hz' - 'rad/s' - 'rpm' - 'kHz' - 'MHz' - 'GHz' - 'rad/nanosecond' - 'rad/microsecond' - 'rad/millisecond' - 'rad/minute' - 'rad/hour' - 'rad/day' - 'rad/week' - 'rad/month' - 'rad/year' - 'cycles/nanosecond' - 'cycles/microsecond' - 'cycles/millisecond' - 'cycles/hour' - 'cycles/day' |

Optional Unit Conversions for Response Plots (Continued)

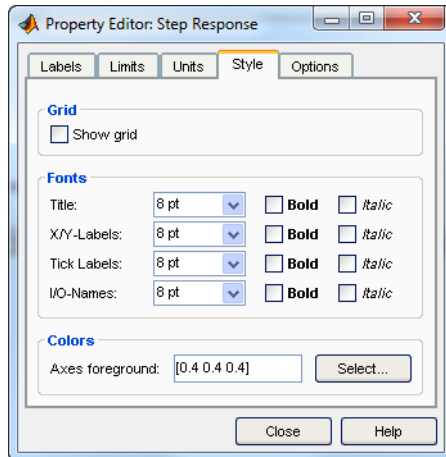
| Response Plot | Unit Conversions |
|-----------------|--|
| | <ul style="list-style-type: none"> - 'cycles/week' - 'cycles/month' - 'cycles/year' |
| Singular Values | <ul style="list-style-type: none"> • Frequency <p>By default, shows rad/TimeUnit where TimeUnit is the system time units specified in the TimeUnit property of the input system.</p> <p>Frequency Units Options</p> <ul style="list-style-type: none"> - 'Hz' - 'rad/s' - 'rpm' - 'kHz' - 'MHz' - 'GHz' - 'rad/nanosecond' - 'rad/microsecond' - 'rad/millisecond' - 'rad/minute' - 'rad/hour' - 'rad/day' - 'rad/week' - 'rad/month' - 'rad/year' - 'cycles/nanosecond' |

Optional Unit Conversions for Response Plots (Continued)

| Response Plot | Unit Conversions |
|---------------|--|
| | <ul style="list-style-type: none"> - 'cycles/microsecond' - 'cycles/millisecond' - 'cycles/hour' - 'cycles/day' - 'cycles/week' - 'cycles/month' - 'cycles/year' • Frequency scale is logarithmic or linear. • Magnitude in decibels or the absolute value using logarithmic or linear scale |
| Step | <ul style="list-style-type: none"> • Time. <p>By default, shows the system time units specified in the TimeUnit property of the input system.</p> <p>Time Units Options</p> <ul style="list-style-type: none"> - 'nanoseconds' - 'microseconds' - 'milliseconds' - 'seconds' - 'minutes' - 'hours' - 'days' - 'weeks' - 'months' - 'years' |

Style Pane

Use the Style pane to toggle grid visibility and set font preferences and axes foreground colors for response plots.



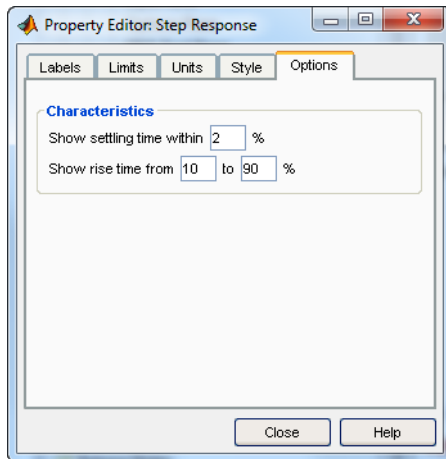
You have the following choices:

- **Grid** — Activate grids by default in new plots.
- **Fonts** — Set the font size, weight (bold), and angle (italic) for fonts used in response plot titles, X/Y-labels, tick labels, and I/O-names.
- **Colors** — Specify the color vector to use for the axes foreground, which includes the X-Y axes, grid lines, and tick labels. Use a three-element vector to represent red, green, and blue (RGB) values. Vector element values can range from 0 to 1.

If you do not want to specify RGB values numerically, click the **Select** button to open the **Select Color** dialog box. See “Select colors” on page 9-13 for more information.

Options Pane

The **Options** pane allows you to customize response characteristics for plots. Each response plot has its own set of characteristics and optional settings; the table below lists them. Use the check boxes to activate the feature and the fields to specify rise or settling time percentages.



Response Characteristic Options for Response Plots

| Plot | Customizable Feature |
|---------------------------------|--|
| Bode Diagram and Bode Magnitude | Select lower magnitude limit Adjust phase offsets to keep phase close to a particular value, within a range of $\pm 180^\circ$, at a given frequency. Unwrap phase (default is unwrapped) |
| Impulse | Show settling time within $xx\%$ (specify the percentage) |

Response Characteristic Options for Response Plots (Continued)

| Plot | Customizable Feature |
|-----------------|--|
| Nichols Chart | Select lower magnitude limit Adjust phase offsets to keep phase close to a particular value, within a range of $\pm 180^\circ$, at a given frequency. Unwrap phase (default is unwrapped) |
| Nyquist Diagram | None |
| Pole/Zero Map | None |
| Sigma | None |
| Step | Show settling time within $xx\%$ (specify the percentage) Show rise time from xx to $yy\%$ (specify the percentages) |

Editing Subplots Using the Property Editor

If you create more than one plot in a single figure window, you can edit each plot individually. For example, the following code creates a figure with two plots, a step and an impulse response with two randomly selected systems:

```
subplot(2,1,1)
step(rss(2,1))
subplot(2,1,2)
impulse(rss(1,1))
```

After the figure window appears, double-click in the upper (step response) plot to activate the **Property Editor**. You will see a set of small black squares appear around the step response, indicating that it is the active plot for the editor. To switch to the lower (impulse response) plot, just click once in the impulse response plot region. The set of black squares switches to the impulse response, and the **Property Editor** updates as well.

Customizing Response Plots Using Plot Tools

In this section...

“Properties You Can Customize Using Plot Tools” on page 10-19

“Opening and Working with Plot Tools” on page 10-20

“Example of Changing Line Color Using Plot Tools” on page 10-20

Properties You Can Customize Using Plot Tools

The following table shows the plot properties you can customize using plot tools.

| For... | You can customize the following properties: |
|-----------|--|
| Responses | <ul style="list-style-type: none"> • System name • Line color • Line style • Line width • Marker type <p>For SISO systems, these changes apply to a single plot line or an array of plot lines representing the system on one axis. For MIMO systems, these changes apply to all of the plotted lines representing the system on multiple axis.</p> |
| Plot axes | <ul style="list-style-type: none"> • Title • X-label • Y-label |
| Figures | <ul style="list-style-type: none"> • Figure name • Colormap • Figure color |

Note To make other changes to response plots, see “Customizing Response Plots Using the Response Plots Property Editor” on page 10-3 and “Customizing Response Plots from the Command Line” on page 10-23.

Opening and Working with Plot Tools

See the following documentation for information about how to open and work with Plot Tools and the Plot Tools Property Editor:

- “Working in Plot Edit Mode” in the MATLAB documentation.
- “The Property Editor” in the MATLAB documentation.

Example of Changing Line Color Using Plot Tools

To change the line color of a MIMO system plot:

- 1** Create a step response plot of a MIMO system by typing

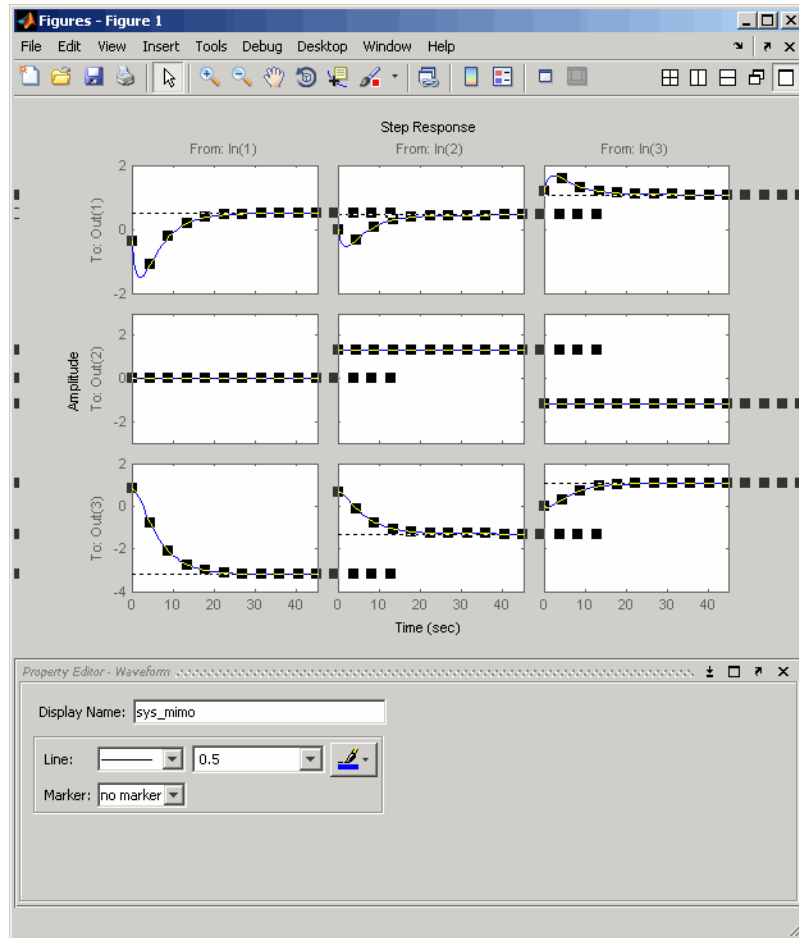
```
sys_mimo=rss(3,3,3);  
stepplot(sys_mimo)
```

- 2** In the figure window, select **View > Property Editor**.

This action opens the Plot Tools Property Editor.

- 3** Click the plot line in any of the nine axis.

This action selects the response for the `sys_mimo` system and updates the Plot Tools Property Editor to show the available editable properties for the response.



Note The Plot Tools Property Editor applies changes to the response of the MIMO system. Any change you make applies to all of the plotted lines in the figure.

Tip You can also change the properties of the response using the right-click menu while in plot edit mode.

4 In the **Property Editor – Waveform** pane, select the color red.

This action changes the color of the response that represents the MIMO system to red.

Customizing Response Plots from the Command Line

In this section...

- “Overview of Customizing Plots from the Command Line” on page 10-23
- “Obtaining Plot Handles” on page 10-26
- “Obtaining Plot Options Handles” on page 10-27
- “Examples of Customizing Plots from the Command Line” on page 10-30
- “Properties and Values Reference” on page 10-33
- “Property Organization Reference” on page 10-47

Overview of Customizing Plots from the Command Line

- “When to Customize Plots from the Command Line” on page 10-23
- “How to Customize Plots from the Command Line” on page 10-23
- “Example of Changing Bode Plot Units from the Command Line” on page 10-25

When to Customize Plots from the Command Line

You can customize any response plot from the command line. The command line is the most efficient way to customize a large number of plots. For example, if you have a batch job that produces many plots, you can change the x -axis units automatically for all the plot with just a few lines of code.

How to Customize Plots from the Command Line

You can use the Control System Toolbox application program interface (API) to customize plotting options for response plots from the command line.

Note This section assumes some very basic familiarity with Handle Graphics® and object-oriented concepts, namely, classes, objects, and Handle Graphics handles. See “MATLAB Classes” and Handle Graphics Objects in the MATLAB online documentation for more information.

To customize plots from the command line:

- 1 Obtain the *plot handle*, which is an identifier for the plot, using the API’s plotting syntax.

For example,

```
h=stepplot(sys)
```

returns the plot handle *h* for the step plot.

For more information on obtaining plot handles, see “Obtaining Plot Handles” on page 10-26.

- 2 Obtain the *plot options handle*, which is an identifier for all settable plot options. To get a plot options handle for a given plot, type

```
p=getoptions(h);
```

p is the plot options handle for plot handle *h*.

For more information on obtaining plot options handles, see “Obtaining Plot Options Handles” on page 10-27.

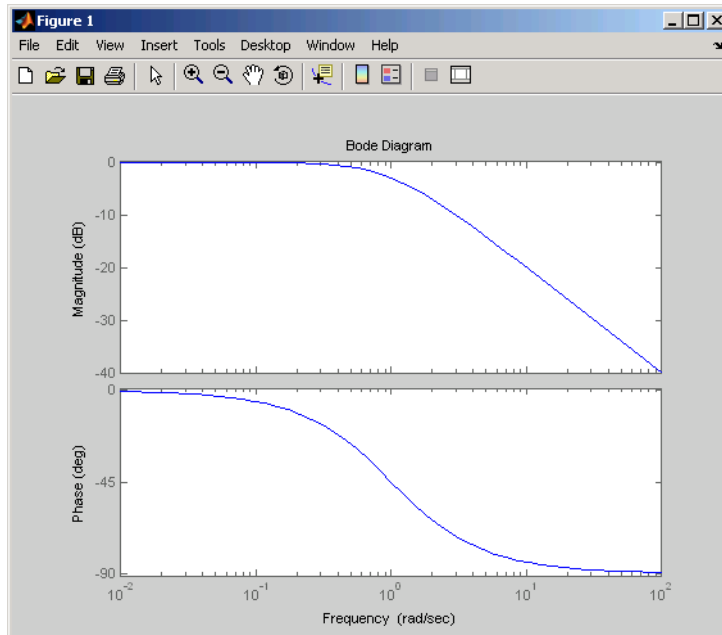
- 3 Use `setoptions`, along with the plot handle and the plot options handle, to access and modify many plot options.

Note You can also use `setoptions` to customize plots using property/value pairs instead of the plot options handle. Using property/value pairs shortens the procedure to one line of code.

Example of Changing Bode Plot Units from the Command Line

You can change of the following plot from rad/s to Hz.

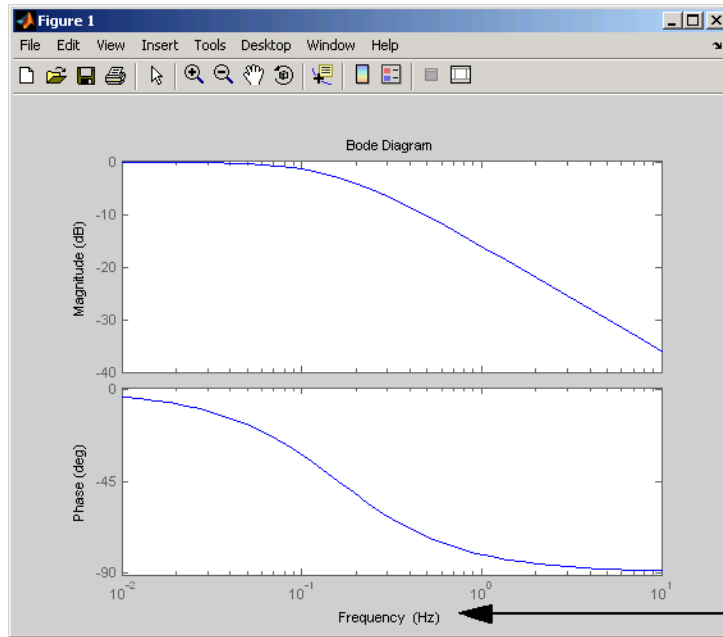
```
s = tf('s');  
sys= 1/(s+1);  
h= bodeplot(sys);
```



To change the units to Hz, type the following:

```
p = getoptions(h);  
p.FreqUnits = 'Hz';  
setoptions(h,p)
```

The units for the x -axis are now in Hz.



Frequency units are now in Hz.

Note You can also change the plot units to Hz using property/value pairs by typing

```
setoptions(h, 'FreqUnits', 'Hz');
```

For more examples of customizing plots from the command line, see “Examples of Customizing Plots from the Command Line” on page 10-30.

Obtaining Plot Handles

To programmatically interact with response plot, you need the *plot handle*. This handle is an identifier to the response plot object. Because the Control System Toolbox plotting commands, `bode`, `rlocus`, etc., all use the plot handle internally, this API provides a set of commands that explicitly return the handle to your response plot. These functions all end with “plot,” which makes them easy to identify. This table lists the functions.

Functions That Return the Plot Handle

| Function | Plot |
|-------------|---|
| bodeplot | Bode magnitude and phase |
| hsvplot | Hankel singular values |
| impzplot | Impulse response |
| initialplot | Initial condition |
| iozplot | Pole/zero maps for input/output pairs |
| lsimplot | Time response to arbitrary inputs |
| nicholsplot | Nichols chart |
| nyquistplot | Nyquist |
| pzplot | Pole/zero |
| rlocusplot | Root locus |
| sigmaplot | Singular values of the frequency response |
| stepplot | Step response |

To get a plot handle for any response plot, use the functions from the table. For example,

```
h = bodeplot(sys)
```

returns plot handle `h` (it also renders the Bode plot). Once you have this handle, you can modify the plot properties using the `setoptions` and `getoptions` methods of the plot object, in this case, a Bode plot handle.

Obtaining Plot Options Handles

- “Overview of Plot Options Handles” on page 10-28
- “Retrieving a Handle” on page 10-28
- “Creating a Handle” on page 10-28
- “Which Properties Can You Modify?” on page 10-29

Overview of Plot Options Handles

Once you have the plot handle, you need the *plot options handle*, which is an identifier for all the settable plot properties for a given response plot. There are two ways to create a plot options handle:

- Retrieving a Handle — Use `getoptions` to get the handle.
- Creating a Handle — Use `<responseplot>options` to instantiate a handle. See Functions for Creating Plot Options Handles on page 10-29 for a complete list.

Retrieving a Handle

The `getoptions` function retrieves a plot options handle from a plot handle.

```
p=getoptions(h) % Returns plot options handle p for plot handle h.
```

If you specify a property name as an input argument, `getoptions` returns the property value associated with the property name.

```
property_value=getoptions(h,PropertyName) % Returns a property  
% value.
```

Creating a Handle

You can create a default plot options handle by using functions in the form of

```
<responseplot>options
```

For example,

```
p=bodeoptions;
```

instantiates a handle for Bode plots. See “Properties and Values Reference” on page 10-33 for a list of default values.

If you want to set the default values to the Control System Toolbox default values, pass `cstprefs` to the function. For example,

```
p = bodeoptions('cstprefs');
```

set the Bode plot property/value pairs to the Control System Toolbox default values.

This table lists the functions that create a plot options handle.

Functions for Creating Plot Options Handles

| Function | Type of Plot Options Handle Created |
|----------------|-------------------------------------|
| bodeoptions | Bode phase and magnitude |
| hsvoptions | Hankel singular values |
| nicholsoptions | Nichols plot |
| nyquistoptions | Nyquist plot |
| pzoptions | Pole/zero plot |
| sigmaoptions | Sigma (singular values) plot |
| timeoptions | Time response (impulse, step, etc.) |

Which Properties Can You Modify?

Use

```
help <responseplot>options
```

to see a list of available property value pairs that you can modify. For example,

```
help bodeoptions
```

You can modify any of these parameters using `setoptions`. The next topic provides examples of modifying various response plots.

See “Properties and Values Reference” on page 10-33 for a complete list of property/value pairs for response plots.

Examples of Customizing Plots from the Command Line

- “Manipulating Plot Options Handles” on page 10-30
- “Changing Plot Units” on page 10-30
- “Create Plots Using Existing Plot Options Handle” on page 10-31
- “Creating a Default Plot Options Handle” on page 10-32
- “Using Dot Notation Like a Structure” on page 10-32
- “Setting Property Pairs in setoptions” on page 10-33

Manipulating Plot Options Handles

There are two fundamental ways to manipulate plot option handles:

- Dot notation — Treat the handle like a MATLAB structure.
- Property value pairs — Specify property/value pairs explicitly as input arguments to `setoptions`.

For some examples, both dot notation and property/value pairs approaches are shown. For all examples, use

```
sys=tf(1,[1 1])
```

for the system.

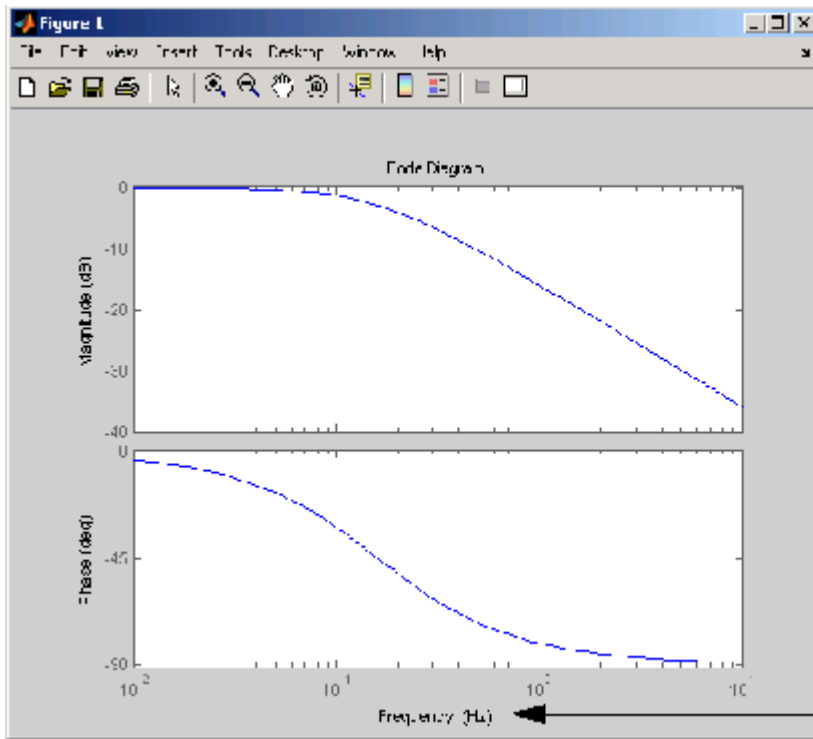
Changing Plot Units

Change the frequency units of a Bode plot from rad/s to Hz.

```
h = bodeplot(sys);  
p = getoptions(h);  
p.FreqUnits = 'Hz'  
setoptions(h,p)
```

or, for the last three lines, substitute

```
setoptions(h, 'FreqUnits', 'Hz')
```

Frequency units are
now in Hz.

Create Plots Using Existing Plot Options Handle

You can use an existing plot options handle to customize a second plot:

```
h1 = bodeplot(sys);
p1 = getoptions(h1);
h2 = bodeplot(sys,p1);
```

OR

```
h1 = bodeplot(sys);
h2 = bodeplot(sys2);
setoptions(h2,getoptions(h1))
```

Creating a Default Plot Options Handle

Instantiate a plot options handle with this code.

```
p = bodeoptions;
```

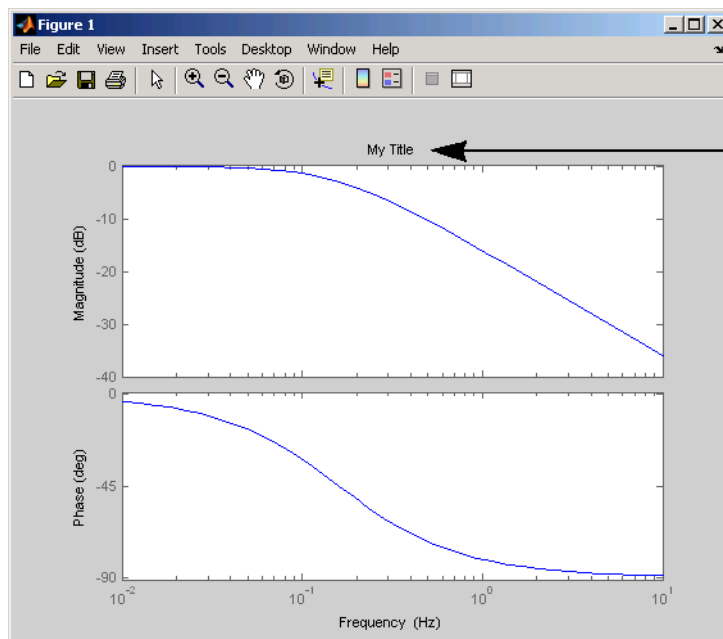
Change the frequency units and apply the changes to sys.

```
p.FreqUnits = 'Hz';  
h = bodeplot(sys,p);
```

Using Dot Notation Like a Structure

You can always use dot notation to assign values to properties.

```
h1 = bodeplot(sys)  
p1 = getoptions(h1)  
p1.FreqUnits = 'Hz'  
p1.Title.String = 'My Title';  
setoptions(h1,p1)
```



Setting Property Pairs in setoptions

Instead of using dot notation, specify frequency units as property/value pairs in setoptions.

```
h1 = bodeplot(sys)
setoptions(h1, 'FreqUnits', 'Hz')
```

Verify that the units have changed from rad/s to Hz.

```
getoptions(h1, 'FreqUnits') % Returns frequency units for h1.

ans =

Hz
```

Properties and Values Reference

- “Property/Value Pairs Common to All Response Plots” on page 10-33
- “Bode Plots” on page 10-38
- “Hankel Singular Values” on page 10-40
- “Nichols Plots” on page 10-40
- “Nyquist Charts” on page 10-42
- “Pole/Zero Maps” on page 10-43
- “Sigma Plots” on page 10-45
- “Time Response Plots” on page 10-46

Property/Value Pairs Common to All Response Plots

The following tables discuss property/value pairs common to all response plots.

Title

| Property | Default Value | Description |
|----------------|---------------|-------------|
| Title.String | none | String |
| Title.FontSize | 8 | Double |

Title (Continued)

| Property | Default Value | Description |
|------------------|----------------------|--------------------------------|
| Title.FontWeight | normal | [light normal demi bode] |
| Title.FontAngle | normal | [normal italic oblique] |
| Title.Color | [0 0 0] | 1-by-3 RGB vector |

X Label

| Property | Default Value | Description |
|-------------------|----------------------|--------------------------------|
| XLabel.String | none | String |
| XLabel.FontSize | 8 | Double |
| XLabel.FontWeight | normal | [light normal demi bode] |
| XLabel.FontAngle | normal | [normal italic oblique] |
| XLabel.Color | [0 0 0] | 1-by-3 RGB vector |

Y Label

| Property | Default Value | Description |
|-------------------|----------------------|--------------------------------|
| YLabel.String | none | String |
| YLabel.FontSize | 8 | Double |
| YLabel.FontWeight | normal | [light normal demi bode] |

Y Label (Continued)

| Property | Default Value | Description |
|------------------|---------------|-----------------------------|
| YLabel.FontAngle | normal | [normal italic oblique] |
| YLabel.Color | [0 0 0] | 1-by-3 RGB vector |

Tick Label

| Property | Default Value | Description |
|----------------------|---------------|--------------------------------|
| TickLabel.FontSize | 8 | Double |
| TickLabel.FontWeight | normal | [light normal demo bode] |
| TickLabel.FontAngle | normal | [normal italic oblique] |
| Ticklabel.Color | [0 0 0] | 1-by-3 RGB vector |

Grid and Axis Limits

| Property | Default Value | Description |
|----------|---------------|---|
| grid | off | [on off] |
| Xlim | {[]} | A cell array of 1-by-2 doubles that specifies the <i>x</i> -axis limits when XLimMode is set to manual. When XLim is scalar, scalar expansion is applied; otherwise the length of the cell array must equal the number of columns (i.e., number of system inputs) for the plot. The 1-by-2 doubles must be a strictly increasing pair [xmin, xmax]. |

Grid and Axis Limits (Continued)

| Property | Default Value | Description |
|-----------------|----------------------|---|
| XLimMode | {auto} | A cell array of strings [auto manual] that specifies the <i>x</i> -axis limits mode. When XLimMode is set to manual the limits are set to the values specified in XLim. When XLimMode is scalar, scalar expansion is applied; otherwise the length of the cell array must equal the number of columns (i.e., number of system inputs) for the plot. |
| YLim | {[]} | A cell array of 1-by-2 doubles specifies the <i>y</i> -axis limits when YLimMode is set to manual. When YLim is scalar, scalar expansion is applied; otherwise the length of the cell array must equal the number of rows (i.e., number of system outputs) for the plot. The 1-by-2 doubles must be a strictly increasing pair [ymin, ymax]. |
| YLimMode | {auto} | A cell array of strings [auto manual] that specifies the <i>y</i> -axis limits mode. When YLimMode is set to manual the limits are set to the values specified in YLim. When YLimMode is scalar, scalar expansion is applied; otherwise the length of the cell array must equal the number of rows (i.e., number of system outputs) for the plot. |

I/O Grouping

| Property | Default Value | Description |
|-----------------|----------------------|--|
| IOWGrouping | none | [none inputs outputs all] Specifies input/output groupings for responses. |

Input Labels

| Property | Default Value | Description |
|------------------------|---------------|--------------------------------|
| InputLabels.FontSize | 8 | Double |
| InputLabels.FontWeight | normal | [light normal demi bode] |
| InputLabels.FontAngle | normal | [normal italic oblique] |
| InputLabels.Color | [0 0 0] | 1-by-3 RGB vector |

Output Labels

| Property | Default Value | Description |
|-------------------------|---------------|--------------------------------|
| OutputLabel.FontSize | 8 | Double |
| OutputLabels.FontWeight | normal | [light normal demi bode] |
| OutputLabels.FontAngle | normal | [normal italic oblique] |
| OutputLabels.Color | [0 0 0] | 1-by-3 RGB vector |

Input/Output Visible

| Property | Default Value | Description |
|---------------|---------------|---|
| InputVisible | {on} | [on off] A cell array that specifies the visibility of each input channel. If the value is a scalar, scalar expansion is applied. |
| OutputVisible | {on} | [on off] A cell array that specifies the visibility of each output channel. If the value is a scalar, scalar expansion is applied. |

Bode Plots

| Property | Default Value | Description |
|-----------|---------------|---|
| FreqUnits | rad/s | <p>Available Options</p> <ul style="list-style-type: none"> • 'Hz' • 'rad/s' • 'rpm' • 'kHz' • 'MHz' • 'GHz' • 'rad/nanosecond' • 'rad/microsecond' • 'rad/millisecond' • 'rad/minute' • 'rad/hour' |

| Property | Default Value | Description |
|-----------------|---------------|--|
| | | <ul style="list-style-type: none"> • 'rad/day' • 'rad/week' • 'rad/month' • 'rad/year' • 'cycles/nanosecond' • 'cycles/microsecond' • 'cycles/millisecond' • 'cycles/hour' • 'cycles/day' • 'cycles/week' • 'cycles/month' • 'cycles/year' |
| FreqScale | log | [linear log] |
| MagUnits | dB | [db abs] |
| MagScale | linear | [linear log] |
| PhaseUnits | deg | [rad deg] |
| PhaseWrapping | off | [on off] |
| MagVisible | on | [on off] |
| PhaseVisible | on | [on off] |
| MagLowerLimMode | auto | [auto manual] Enables a manual lower magnitude limit specification by MagLowerLim. |
| MagLowerLim | 0 | Double Specifies the lower magnitude limit when MagLowerLimMode is set to manual. |

| Property | Default Value | Description |
|--------------------|---------------|---|
| PhaseMatching | off | [on off] Enables adjusting phase effects for phase response. |
| PhaseMatchingFreq | 0 | Double |
| PhaseMatchingValue | 0 | Double |

Hankel Singular Values

| Property | Default Value | Description |
|----------|---------------|---|
| Yscale | linear | [linear log] |
| AbsTol | 0 | Double See hsvd and stabsep for details. |
| RelTol | 1*e-08 | Double See hsvd and stabsep for details. |
| Offset | 1*e-08 | Double See hsvd and stabsep for details. |

Nichols Plots

| Property | Default Value | Description |
|-----------|---------------|---|
| FreqUnits | rad/s | <p>Available Options</p> <ul style="list-style-type: none"> • 'Hz' • 'rad/s' • 'rpm' • 'kHz' • 'MHz' • 'GHz' • 'rad/nanosecond' |

| Property | Default Value | Description |
|--------------------|---------------|--|
| | | <ul style="list-style-type: none"> • 'rad/microsecond' • 'rad/millisecond' • 'rad/minute' • 'rad/hour' • 'rad/day' • 'rad/week' • 'rad/month' • 'rad/year' • 'cycles/nanosecond' • 'cycles/microsecond' • 'cycles/millisecond' • 'cycles/hour' • 'cycles/day' • 'cycles/week' • 'cycles/month' • 'cycles/year' |
| MagUnits | dB | [dB abs] |
| PhaseUnits | deg | [rad deg] |
| MagLowerLimMode | auto | [auto manual] |
| MagLowerLim | 0 | double |
| PhaseMatching | off | [on off] |
| PhaseMatchingFreq | 0 | Double |
| PhaseMatchingValue | 0 | Double |

Nyquist Charts

| Property | Default Value | Description |
|-----------|---------------|--|
| FreqUnits | rad/s | <p>Available Options</p> <ul style="list-style-type: none"> • 'Hz' • 'rad/s' • 'rpm' • 'kHz' • 'MHz' • 'GHz' • 'rad/nanosecond' • 'rad/microsecond' • 'rad/millisecond' • 'rad/minute' • 'rad/hour' • 'rad/day' • 'rad/week' • 'rad/month' • 'rad/year' • 'cycles/nanosecond' • 'cycles/microsecond' • 'cycles/millisecond' • 'cycles/hour' • 'cycles/day' • 'cycles/week' • 'cycles/month' |

| Property | Default Value | Description |
|-----------------|---------------|---|
| | | <ul style="list-style-type: none"> 'cycles/year' |
| MagUnits | dB | [dB abs] |
| PhaseUnits | deg | [rad deg] |
| ShowFullContour | on | [on off] |

Pole/Zero Maps

| Property | Default Value | Description |
|-----------|---------------|---|
| FreqUnits | rad/s | <p>Available Options</p> <ul style="list-style-type: none"> 'Hz' 'rad/s' 'rpm' 'kHz' 'MHz' 'GHz' 'rad/nanosecond' 'rad/microsecond' 'rad/millisecond' 'rad/minute' 'rad/hour' 'rad/day' 'rad/week' 'rad/month' 'rad/year' |

| Property | Default Value | Description |
|-----------|---------------|--|
| | | <ul style="list-style-type: none"> • 'cycles/nanosecond' • 'cycles/microsecond' • 'cycles/millisecond' • 'cycles/hour' • 'cycles/day' • 'cycles/week' • 'cycles/month' • 'cycles/year' |
| TimeUnits | seconds | <p>Available Options</p> <ul style="list-style-type: none"> • 'nanoseconds' • 'microseconds' • 'milliseconds' • 'seconds' • 'minutes' • 'hours' • 'days' • 'weeks' • 'months' • 'years' |

Sigma Plots

| Property | Default Value | Description |
|-----------|---------------|--|
| FreqUnits | rad/s | <p>Available Options</p> <ul style="list-style-type: none"> • 'Hz' • 'rad/s' • 'rpm' • 'kHz' • 'MHz' • 'GHz' • 'rad/nanosecond' • 'rad/microsecond' • 'rad/millisecond' • 'rad/minute' • 'rad/hour' • 'rad/day' • 'rad/week' • 'rad/month' • 'rad/year' • 'cycles/nanosecond' • 'cycles/microsecond' • 'cycles/millisecond' • 'cycles/hour' • 'cycles/day' • 'cycles/week' • 'cycles/month' |

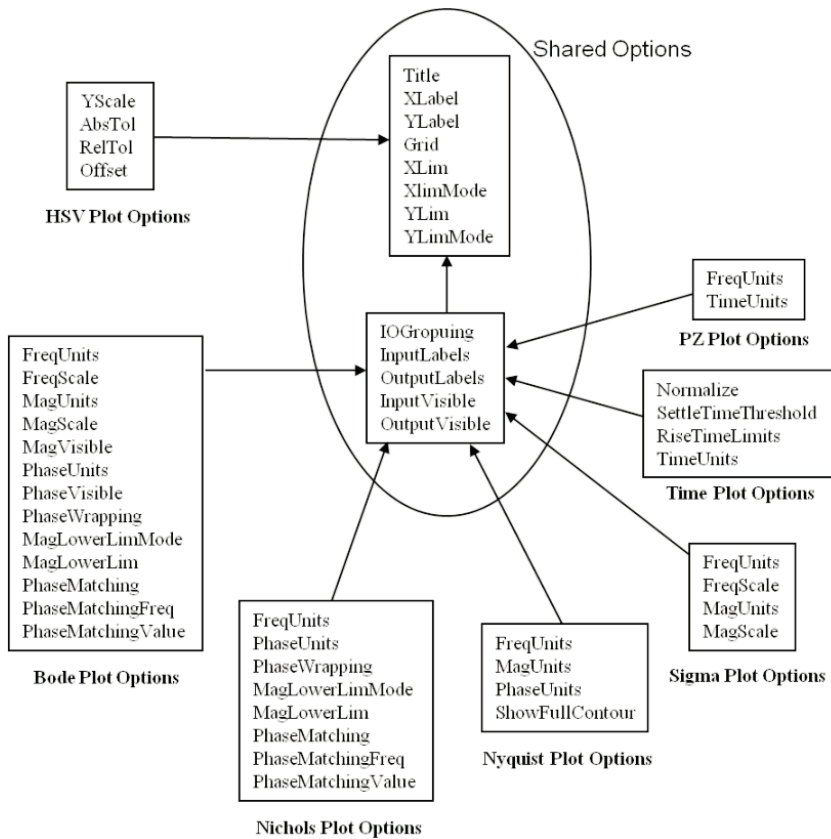
| Property | Default Value | Description |
|-----------|---------------|---|
| | | <ul style="list-style-type: none"> 'cycles/year' |
| FreqScale | log | [linear log] |
| MagUnits | dB | [dB abs] |
| MagScale | linear | [linear log] |

Time Response Plots

| Property | Default Value | Description |
|---------------------|---------------|---|
| Normalize | off | [on off] Normalize the y-scale of all responses in the plot. |
| SettleTimeThreshold | 0.02 | Double Specifies the settling time threshold. 0.02 = 2%. |
| RiseTimeLimits | [0.1, 0.9] | 1-by-2 double Specifies the limits used to define the rise time. [0.1, 0.9] is 10% to 90%. |
| TimeUnits | seconds | <p>Available Options</p> <ul style="list-style-type: none"> 'nanoseconds' 'microseconds' 'milliseconds' 'seconds' 'minutes' 'hours' 'days' 'weeks' |

| Property | Default Value | Description |
|----------|---------------|---|
| | | <ul style="list-style-type: none"> 'months' 'years' |

Property Organization Reference



Customizing Plots Inside the SISO Design Tool

| In this section... |
|--|
| “Overview of Customizing SISO Design Tool Plots” on page 10-48 |
| “Root Locus Property Editor” on page 10-48 |
| “Open-Loop Bode Property Editor” on page 10-52 |
| “Open-Loop Nichols Property Editor” on page 10-55 |
| “Prefilter Bode Property Editor” on page 10-57 |

Overview of Customizing SISO Design Tool Plots

Customizing plots inside the SISO Design Tool is similar to how you customize any response plot. The following property editors are specific to the SISO Design Tool:

- Root Locus Property Editor
- Open-Loop Bode Property Editor
- Open-Loop Nichols Property Editor
- Prefilter Bode Property Editor

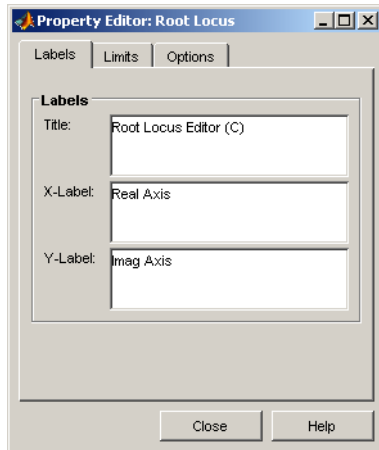
You can use each of these property editors to create the customized plots within the SISO Design tool.

Root Locus Property Editor

There are three ways to open the Property Editor for root locus plots:

- Double-click in the root locus away from the curve
- Select **Properties** from the right-click menu
- Select **Root Locus** and then **Properties** from Edit in the menu bar

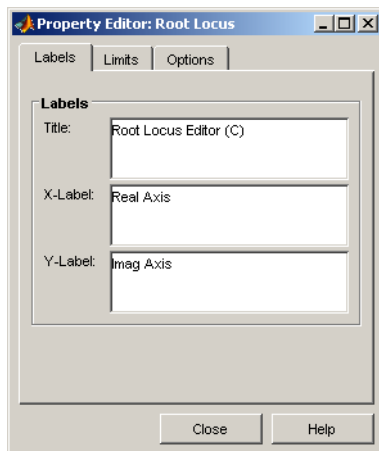
This figure shows the **Property Editor: Root Locus** window.



- “Labels Pane” on page 10-49
- “Limits Pane” on page 10-50
- “Options Pane” on page 10-51

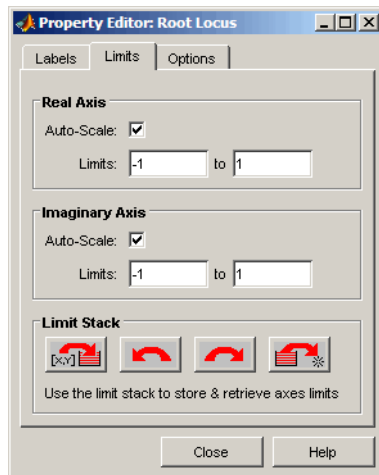
Labels Pane

You can use the **Label** pane to specify plot titles and axis labels. To specify a new label, type the string in the appropriate field. The root locus plot automatically updates.



Limits Pane

The SISO Design Tool specifies default values for the real and imaginary axes ranges to make sure that all the poles and zeros in your model appear in the root locus plot. Use the Limits pane, shown below, to override the default settings.



To change the limits, specify the new limits in the real and imaginary axes **Limits** fields. The **Auto-Scale** check box automatically clears once you click in a different field. Your root locus diagram updates immediately. If you want to reapply the default limits, select the **Auto-Scale** check boxes again.

The Limit Stack pane provides support for storing and retrieving custom limit specifications. There are four buttons available:



— Add the current limits to the stack



— Retrieve the previous stack entry



— Retrieve the next stack entry

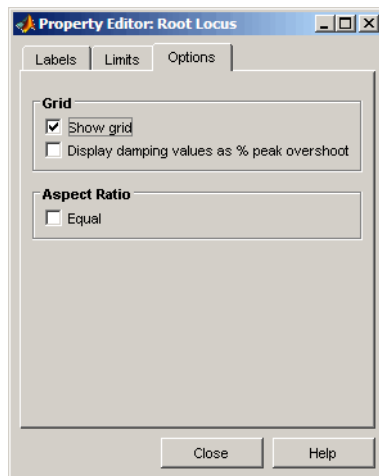


— Remove the current limits from the stack

Using these buttons, you can store and retrieve any number of saved custom axes limits.

Options Pane

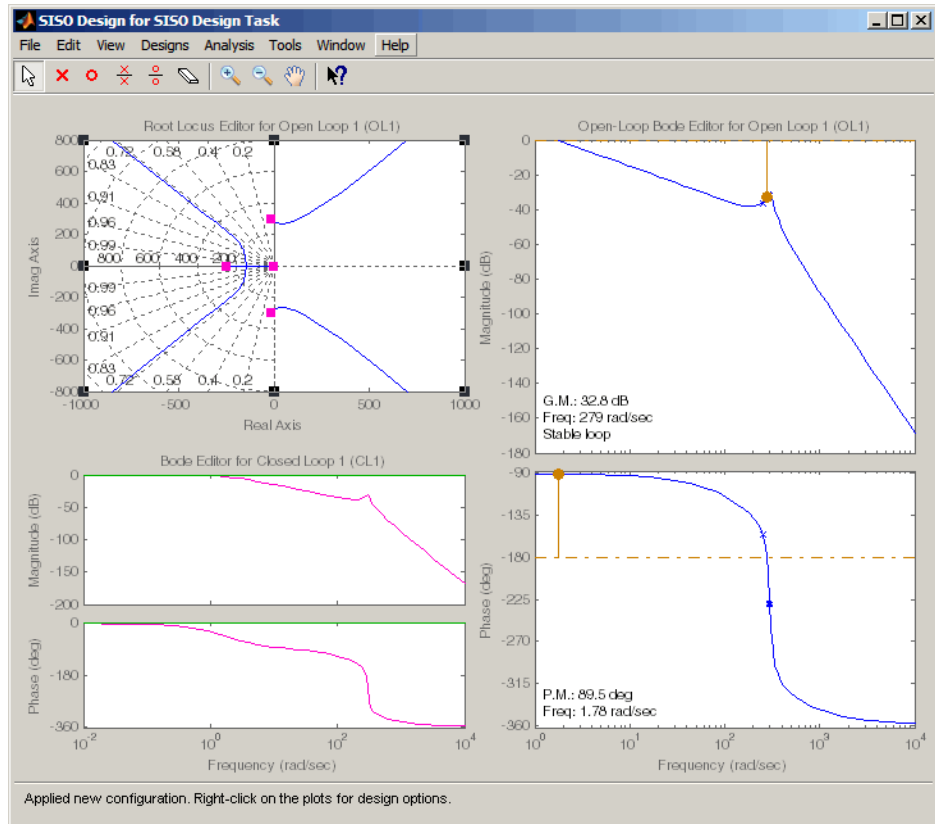
The Options pane contains settings for adding a grid and changing the plot's aspect ratio.



Select **Show grid** to display a grid on the root locus. If you have damping ratio constraints on your root locus, selecting **Display damping ratios as % peak overshoot** displays the damping ratio values along the grid lines. This figure shows both options activated for an imported model, Gservo. If you want to verify these settings, type

```
load ltiexamples
```

at the MATLAB prompt and import Gservo from the workspace into your SISO Design Tool.



The numbers displayed on the root locus gridlines are the damping ratios as a percentage of the overshoot values.

If you select the **Equal** check box in the **Aspect Ratio** pane, the x and y -axes are set to equal limit values.

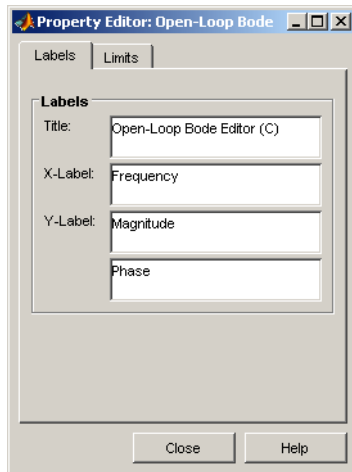
Open-Loop Bode Property Editor

As is the case with the root locus Property Editor, there are three ways to open the Bode diagram property editor:

- Double-click in the Bode magnitude or phase plot away from the curve.

- Select **Properties** from the right-click menu.
- Select **Open-Loop Bode** and then **Properties** from **Edit** in the menu bar.

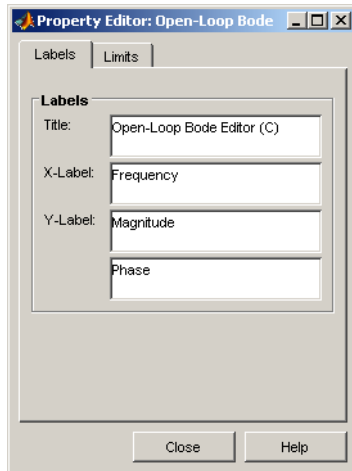
This figure shows the **Property Editor: Open-Loop Bode** editor.



- “Labels Pane” on page 10-53
- “Limits Pane” on page 10-54

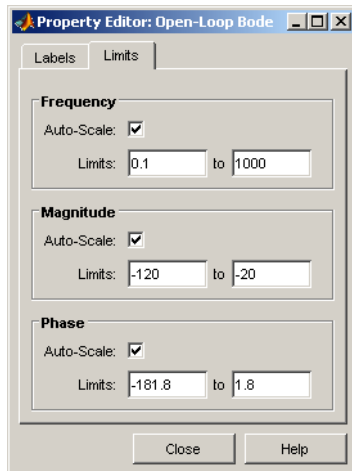
Labels Pane

You can use the **Label** pane to specify plot titles and axis labels. To specify a new label, type the string in the appropriate field. The Bode diagram automatically updates.



Limits Pane

You can use the Limits pane to override the default limits for the frequency, magnitude, and phase scales for your plots.



To change the limits, specify the new values in the **Limits** fields for frequency, magnitude, and phase. The **Auto-Scale** check box automatically deactivates once you click in a different field. The Bode diagram updates immediately.

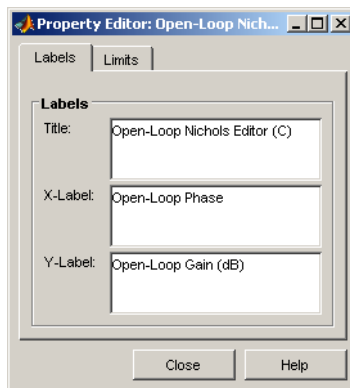
To restore the default settings, select the **Auto-Scale** boxes again.

Open-Loop Nichols Property Editor

As is the case with the root locus Property Editor, there are three ways to open the Nichols plot property editor:

- Double-click in the Nichols plot away from the curve.
- Select **Properties** from the right-click menu.
- Select **Open-Loop Nichols** and then **Properties** from **Edit** in the menu bar.

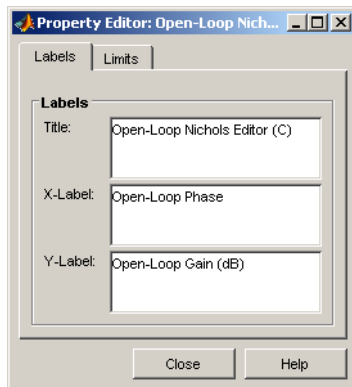
This figure shows the **Property Editor: Open-Loop Nichols** editor.



- “Labels Pane” on page 10-55
- “Limits Pane” on page 10-56

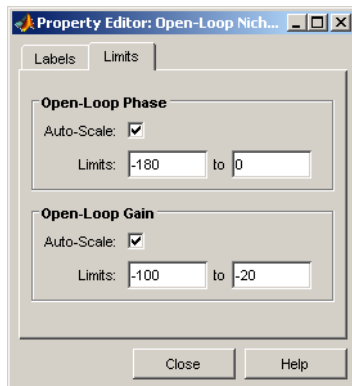
Labels Pane

You can use the Label pane to specify plot titles and axis labels. To specify a new label, type the string in the appropriate field. The Nichols plot automatically updates.



Limits Pane

You can use the Limits pane to override the default limits for the frequency, magnitude, and phase scales for your plots.



To change the limits, specify the new values in the **Limits** fields for open-loop phase and/or gain. The **Auto-Scale** check box automatically deactivates once you click in a different field. The Nichols plot updates immediately.

To restore the default settings, select the **Auto-Scale** boxes again.

Prefilter Bode Property Editor

The **Prefilter Bode Property** editor is identical to the Open-Loop Bode diagram property editor. There are three ways to open the prefilter editor:

- Double-click in the prefilter Bode magnitude or phase plot away from the curve.
- Select **Properties** from the right-click menu.
- Select **Prefilter Bode** and then **Properties** from **Edit** in the menu bar.

See “Open-Loop Bode Property Editor” on page 10-52 for a description of the features of this editor.

Design Case Studies

- “Yaw Damper for a 747 Jet Transport” on page 11-2
- “Hard-Disk Read/Write Head Controller” on page 11-19
- “LQG Regulation: Rolling Mill Example” on page 11-30
- “Kalman Filtering” on page 11-49

Yaw Damper for a 747 Jet Transport

In this section...

“Overview of this Case Study” on page 11-2
 “Creating the Jet Model” on page 11-2
 “Computing Open-Loop Eigenvalues” on page 11-4
 “Open-Loop Analysis” on page 11-5
 “Root Locus Design” on page 11-8
 “Washout Filter Design” on page 11-13

Overview of this Case Study

This case study demonstrates the tools for classical control design by stepping through the design of a yaw damper for a 747[®] jet transport aircraft.

Creating the Jet Model

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

```
A=[ - .0558  - .9968  .0802  .0415;
      .598   - .115  - .0318  0;
     -3.05   .388  - .4650  0;
      0  0.0805  1  0];
```

```
B=[ .00729  0;
     -0.475  0.00775;
      0.153  0.143;
      0      0];
```

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

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

```
sys = ss(A,B,C,D);
```

The following commands 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' 'bank angle'};

sys = ss(A,B,C,D,'statename',states,...
         'inputname',inputs,...
         'outputname',outputs);
```

You can display the LTI model `sys` by typing `sys`. This command produces the following result.

```
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 0.00729      0
yaw  -0.475  0.00775
roll 0.153   0.143
phi   0      0

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

d =
      rudder  aileron
yaw      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 radians as well.

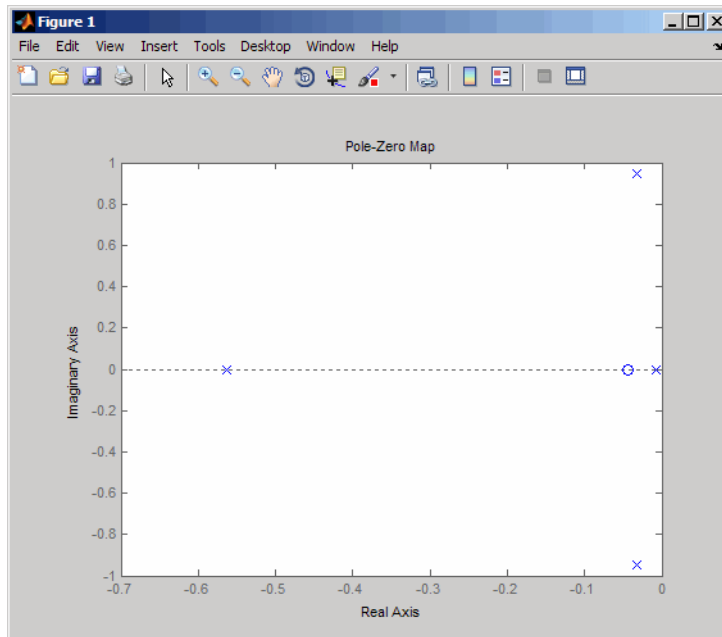
Computing Open-Loop Eigenvalues

Compute the open-loop eigenvalues and plot them in the s-plane.

```
damp(sys)
```

| Eigenvalue | Damping | Freq. (rad/s) |
|-------------------------|-----------|---------------|
| -7.28e-003 | 1.00e+000 | 7.28e-003 |
| -5.63e-001 | 1.00e+000 | 5.63e-001 |
| -3.29e-002 + 9.47e-001i | 3.48e-002 | 9.47e-001 |
| -3.29e-002 - 9.47e-001i | 3.48e-002 | 9.47e-001 |

```
pzmap(sys)
```



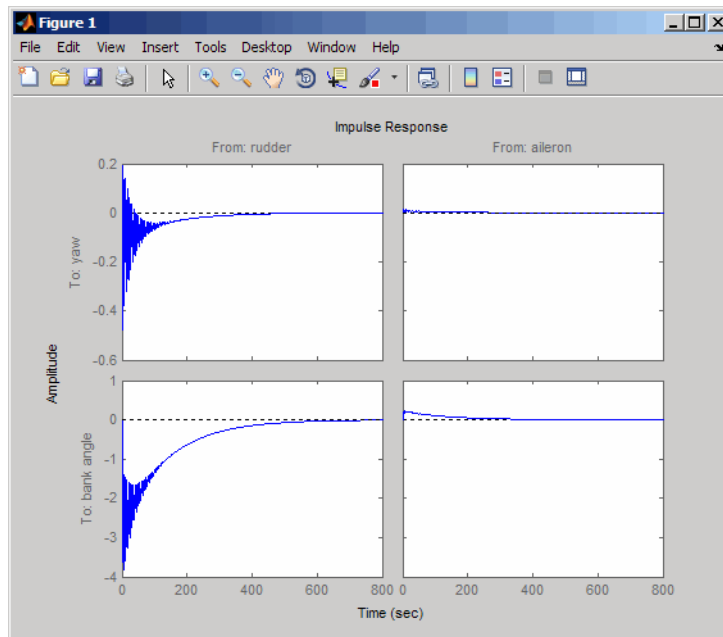
This model has one pair of lightly damped poles. They correspond to what is called the "Dutch roll mode."

Suppose you want to design a compensator that increases the damping of these poles, so that the resulting complex poles have a damping ratio $\zeta > 0.35$ with natural frequency $\omega_n < 1$ rad/sec. You can do this using the Control System Toolbox analysis tools.

Open-Loop Analysis

First, perform some open-loop analysis to determine possible control strategies. Start with the time response (you could use `step` or `impulse` here).

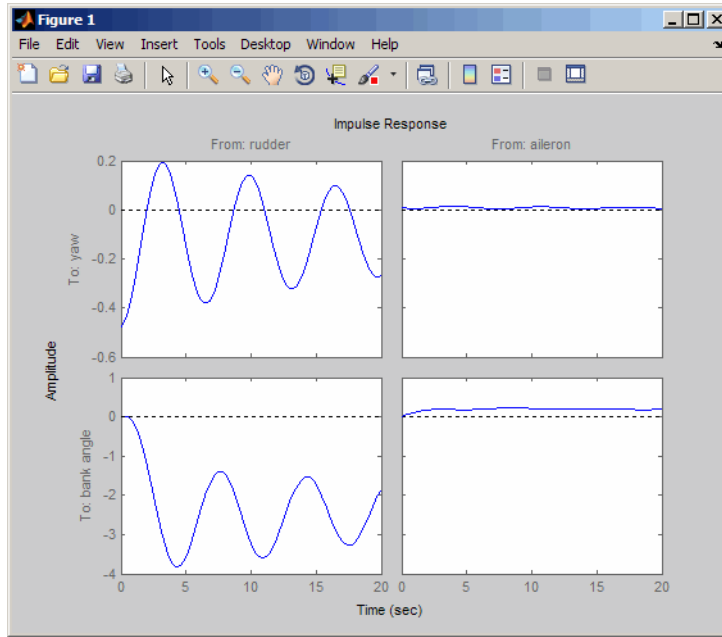
```
impulse(sys)
```



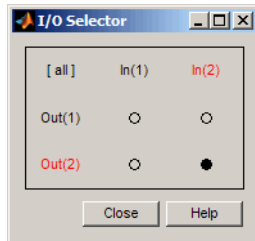
The impulse response confirms that the system is lightly damped. But the time frame is much too long because the passengers and the pilot are more concerned about the behavior during the first few seconds rather than the

first few minutes. Next look at the response over a smaller time frame of 20 seconds.

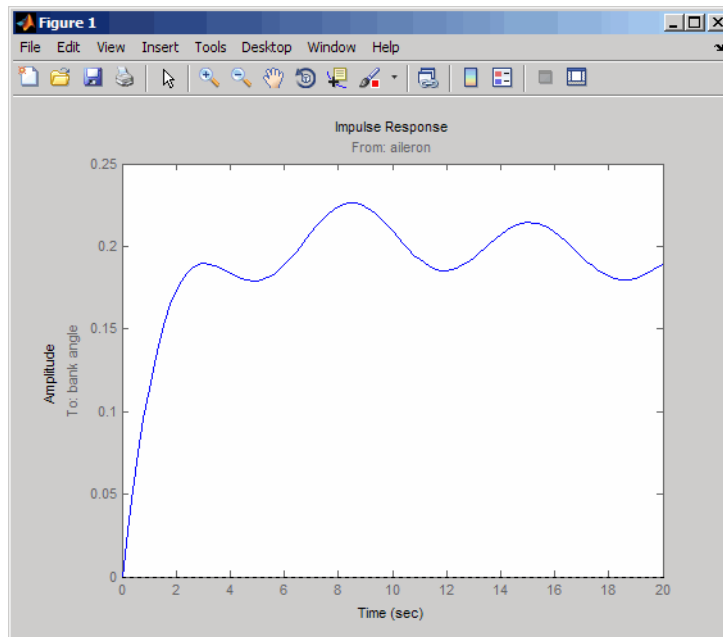
```
impulse(sys,20)
```



Look at the plot from aileron (input 2) to bank angle (output 2). To show only this plot, right-click and choose **I/O Selector**, then click on the (2,2) entry. The I/O Selector should look like this.



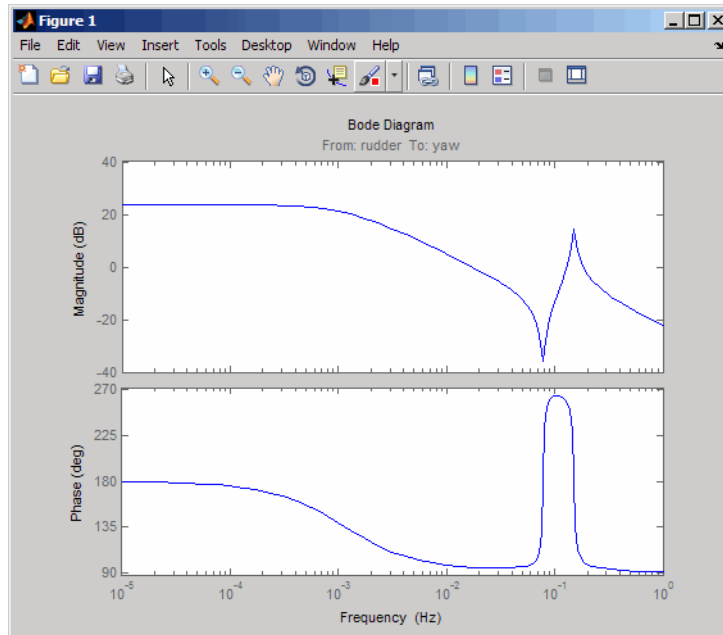
The new figure is shown below.



The aircraft is oscillating around a nonzero bank angle. Thus, the aircraft is turning in response to an aileron impulse. This behavior will prove important later in this case study.

Typically, yaw dampers are designed using the yaw rate as sensed output and the rudder as control input. Look at the corresponding frequency response.

```
sys11=sys('yaw','rudder') % Select I/O pair.  
bode(sys11)
```

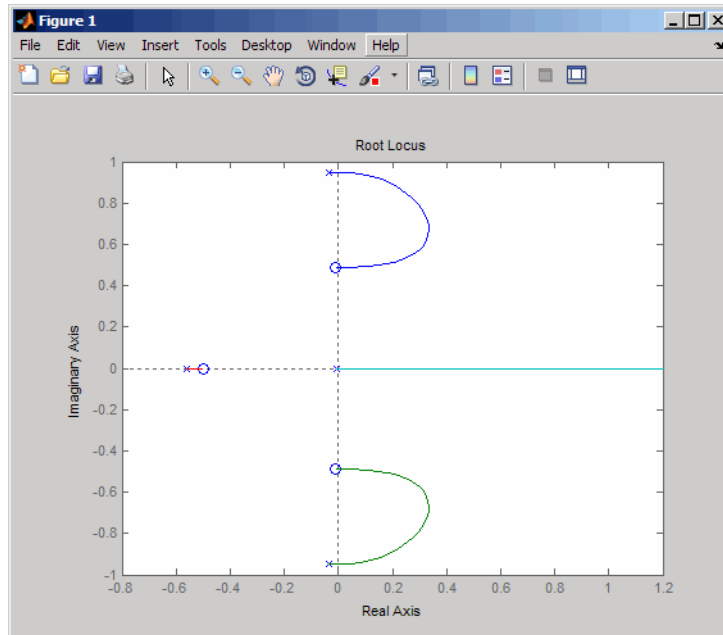


From this Bode diagram, you can see that the rudder has significant effect around the lightly damped Dutch roll mode (that is, near $\omega = 1$ rad/sec).

Root Locus Design

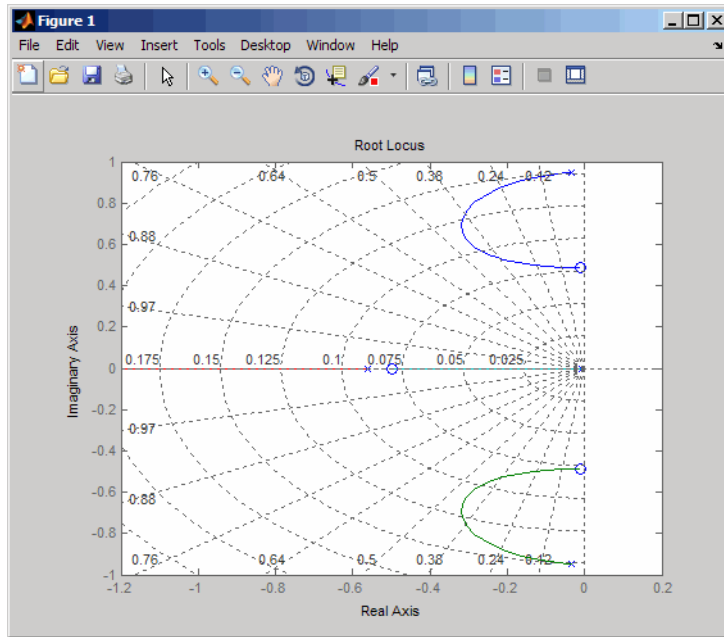
A reasonable design objective is to provide a damping ratio $\zeta > 0.35$ with a natural frequency $\omega_n < 1.0$ rad/sec. Since the simplest compensator is a static gain, first try to determine appropriate gain values using the root locus technique.

```
% Plot the root locus for the rudder to yaw channel
rlocus(sys11)
```

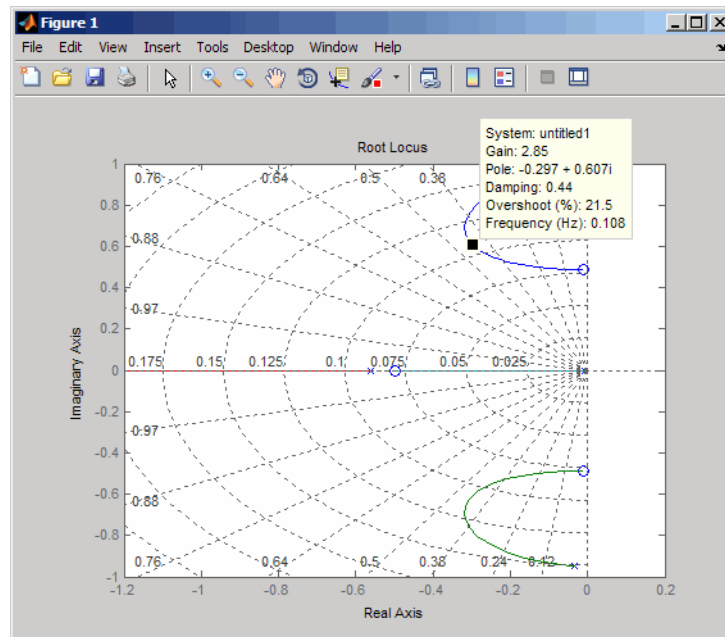


This is the root locus for negative feedback and shows that the system goes unstable almost immediately. If, instead, you use positive feedback, you may be able to keep the system stable.

```
rlocus(-sys11)  
sgrid
```



This looks better. By using simple feedback, you can achieve a damping ratio of $\zeta > 0.45$. Click on the blue curve and move the data marker to track the gain and damping values. To achieve a 0.45 damping ratio, the gain should be about 2.85. This figure shows the data marker with similar values.

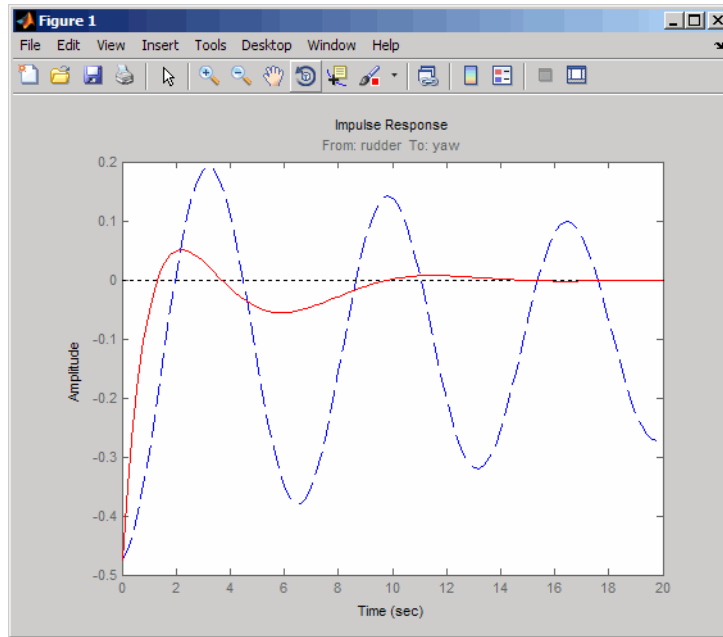


Next, close the SISO feedback loop.

```
K = 2.85;
c111 = feedback(sys11,-K); % Note: feedback assumes negative
                          % feedback by default
```

Plot the closed-loop impulse response for a duration of 20 seconds, and compare it to the open-loop impulse response.

```
impulse(sys11,'b--',c111,'r',20)
```



The closed-loop response settles quickly and does not oscillate much, particularly when compared to the open-loop response.

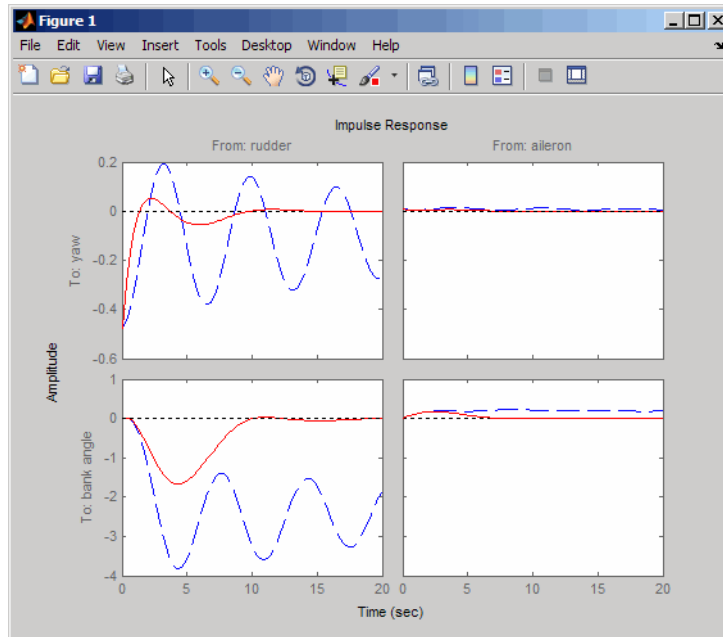
Now close the loop on the full MIMO model and see how the response from the aileron looks. The feedback loop involves input 1 and output 1 of the plant (use feedback with index vectors selecting this input/output pair). At the MATLAB prompt, type

```
cloop = feedback(sys, -K, 1, 1);
damp(cloop) % closed-loop poles
```

| Eigenvalue | Damping | Freq. (rad/s) |
|-------------------------|-----------|---------------|
| -3.42e-001 | 1.00e+000 | 3.42e-001 |
| -2.97e-001 + 6.06e-001i | 4.40e-001 | 6.75e-001 |
| -2.97e-001 - 6.06e-001i | 4.40e-001 | 6.75e-001 |
| -1.05e+000 | 1.00e+000 | 1.05e+000 |

Plot the MIMO impulse response.


```
impulse(sys, 'b - - ', cloop, 'r ', 20)
```



The yaw rate response is now well damped, but look at the plot from aileron (input 2) to bank angle (output 2). When you move the aileron, the system no longer continues to bank like a normal aircraft. You have over-stabilized the spiral mode. The spiral mode is typically a very slow mode and allows the aircraft to bank and turn without constant aileron input. Pilots are used to this behavior and will not like your design if it does not allow them to fly normally. This design has moved the spiral mode so that it has a faster frequency.

Washout Filter Design

What you need to do is make sure the spiral mode does not move further into the left-half plane when you close the loop. One way flight control designers have addressed this problem is to use a washout filter $kH(s)$ where

$$H(s) = \frac{s}{s + \alpha}$$

The washout filter places a zero at the origin, which constrains the spiral mode pole to remain near the origin. We choose $\alpha = 0.2$ for a time constant of five seconds and use the root locus technique to select the filter gain H . First specify the fixed part $s/(s + \alpha)$ of the washout by

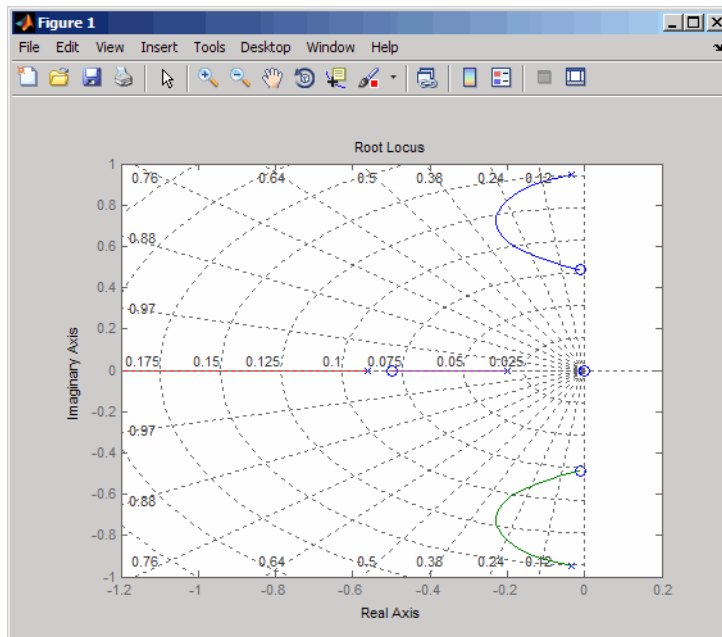
$$H = \text{zpk}(0, -0.2, 1);$$

Connect the washout in series with the design model `sys11` (relation between input 1 and output 1) to obtain the open-loop model

$$\text{olloop} = H * \text{sys11};$$

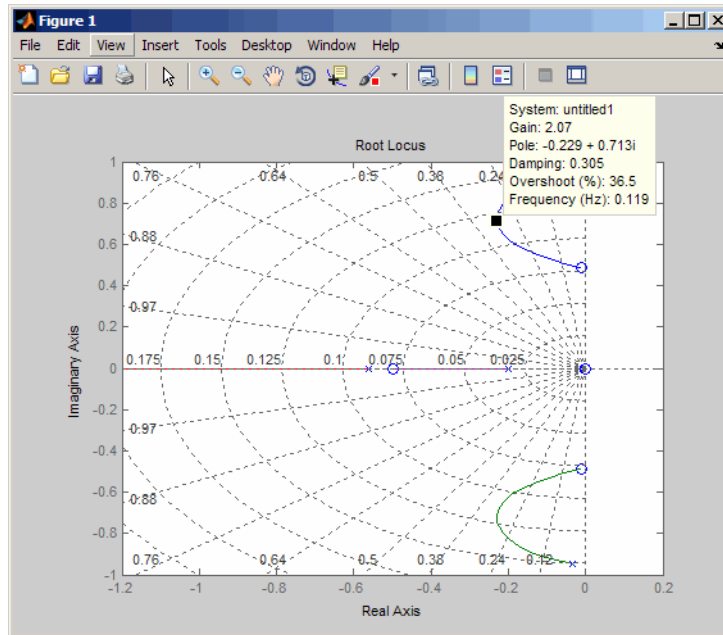
and draw another root locus for this open-loop model.

```
rlocus(-olloop)
sgrid
```



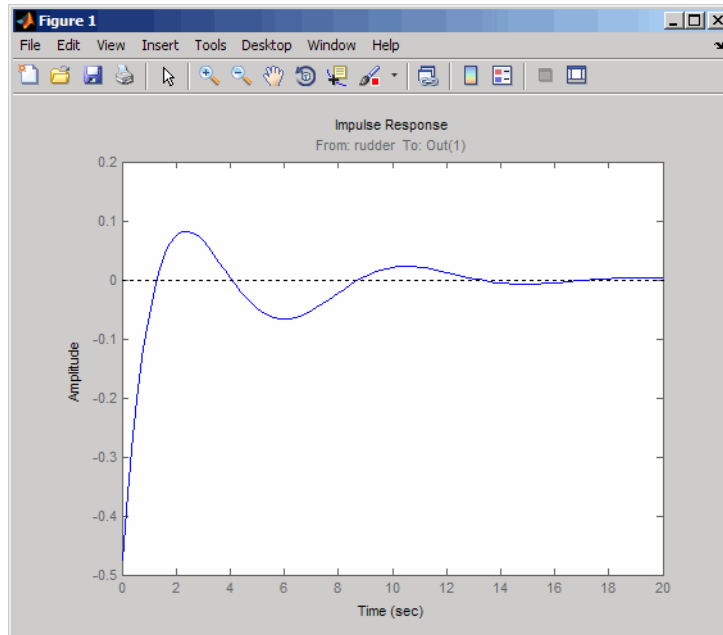
Create and drag a data marker around the upper curve to locate the maximum damping, which is about $\zeta = 0.3$.

This figure shows a data marker at the maximum damping ratio; the gain is approximately 2.07.



Look at the closed-loop response from rudder to yaw rate.

```
K = 2.07;  
c111 = feedback(olloop, -K);  
impulse(c111, 20)
```



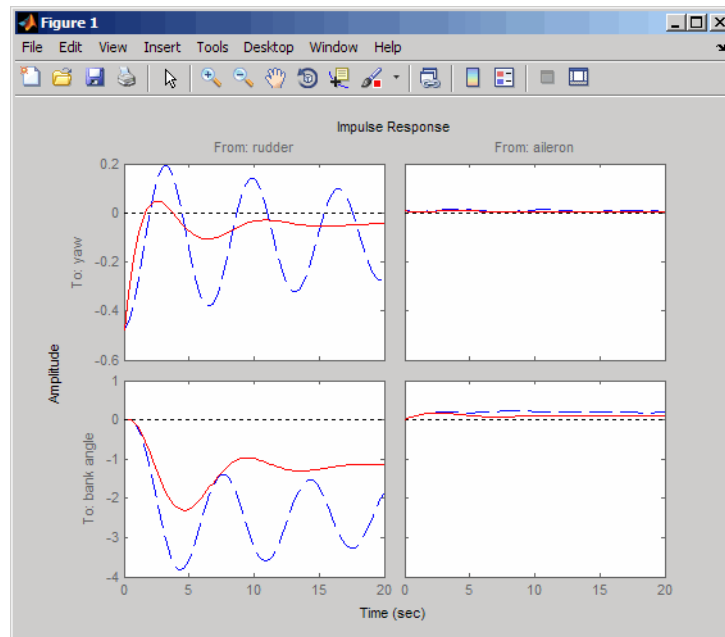
The response settles nicely but has less damping than your previous design. Finally, you can verify that the washout filter has fixed the spiral mode problem. First form the complete washout filter $kH(s)$ (washout + gain).

$$\text{WOF} = -K * H;$$

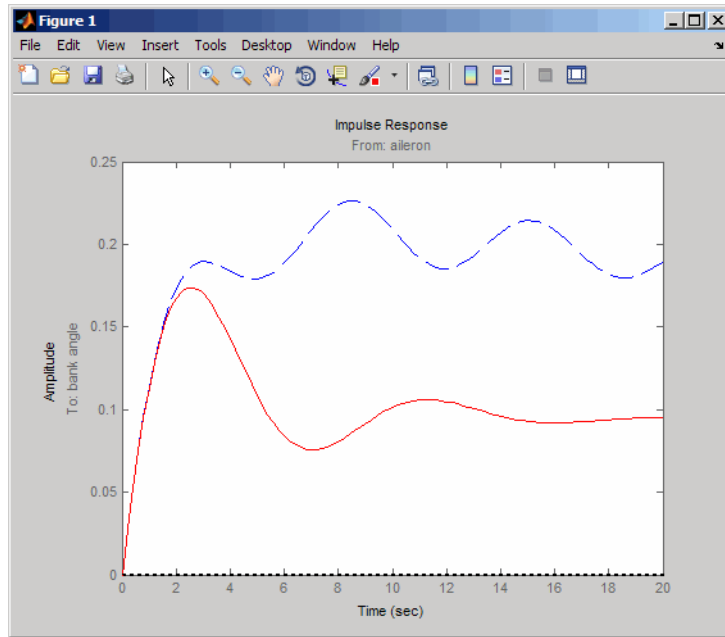
Then close the loop around the first I/O pair of the MIMO model `sys` and simulate the impulse response.

```
cloop = feedback(sys,WOF,1,1);

% Final closed-loop impulse response
impulse(sys,'b--',cloop,'r',20)
```



The bank angle response (output 2) due to an aileron impulse (input 2) now has the desired nearly constant behavior over this short time frame. To inspect the response more closely, use the I/O Selector in the right-click menu to select the (2,2) I/O pair.



Although you did not quite meet the damping specification, your design has increased the damping of the system substantially and now allows the pilot to fly the aircraft normally.

Hard-Disk Read/Write Head Controller

In this section...

“Overview of this Case Study” on page 11-19

“Creating the Read/Write Head Model” on page 11-19

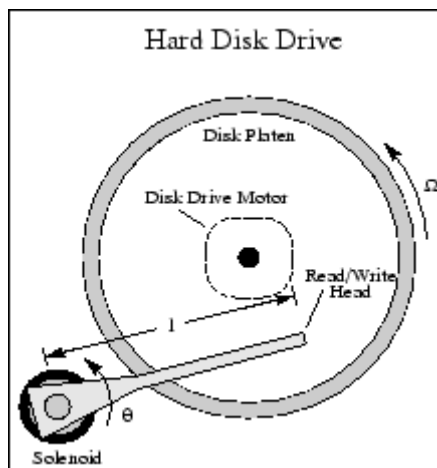
“Model Discretization” on page 11-20

“Adding a Compensator Gain” on page 11-22

“Adding a Lead Network” on page 11-23

“Design Analysis” on page 11-26

Overview of this Case Study



This case study demonstrates the ability to perform classical digital control design by going through the design of a computer hard-disk read/write head position controller.

Creating the Read/Write Head Model

Using Newton's law, a simple model for the read/write head is the differential equation

$$J \frac{d^2\theta}{dt^2} + C \frac{d\theta}{dt} + K\theta = K_i i$$

where J is the inertia of the head assembly, C is the viscous damping coefficient of the bearings, K is the return spring constant, K_i is the motor torque constant, θ is the angular position of the head, and i is the input current.

Taking the Laplace transform, the transfer function from i to θ is

$$H(s) = \frac{K_i}{Js^2 + Cs + K}$$

Using the values $J = 0.01 \text{ kg } m^2$, $C = 0.004 \text{ Nm/(rad/sec)}$, $K = 10 \text{ Nm/rad}$, and $K_i = 0.05 \text{ Nm/rad}$, form the transfer function description of this system. At the MATLAB prompt, type

```
J = .01; C = 0.004; K = 10; Ki = .05;
num = Ki;
den = [J C K];
H = tf(num,den)
```

These commands produce the following result.

```
Transfer function:
      0.05
-----
0.01 s^2 + 0.004 s + 10
```

Model Discretization

The task here is to design a digital controller that provides accurate positioning of the read/write head. The design is performed in the digital domain. First, discretize the continuous plant. Because our plant will be equipped with a digital-to-analog converter (with a zero-order hold) connected to its input, use `c2d` with the 'zoh' discretization method. Type

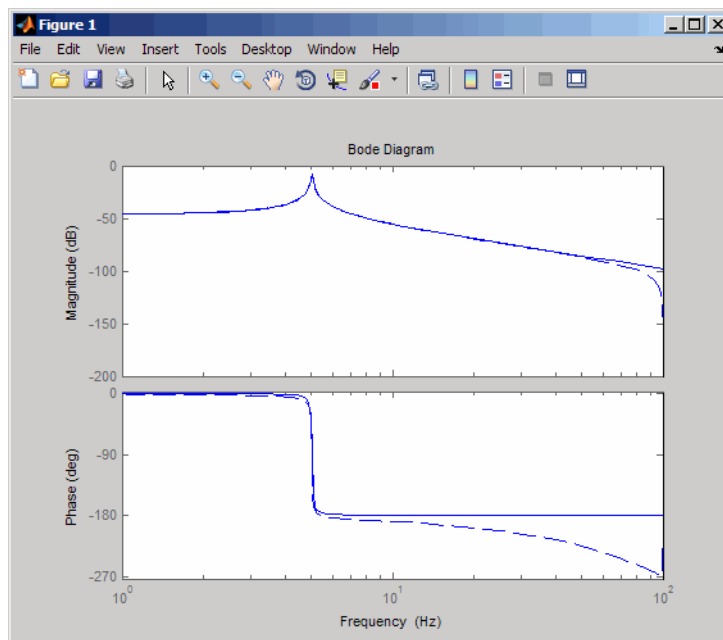
```
Ts = 0.005;      % sampling period = 0.005 second
Hd = c2d(H,Ts,'zoh')
Transfer function:
```


$$\frac{6.233e-05 z + 6.229e-05}{z^2 - 1.973 z + 0.998}$$

Sampling time: 0.005

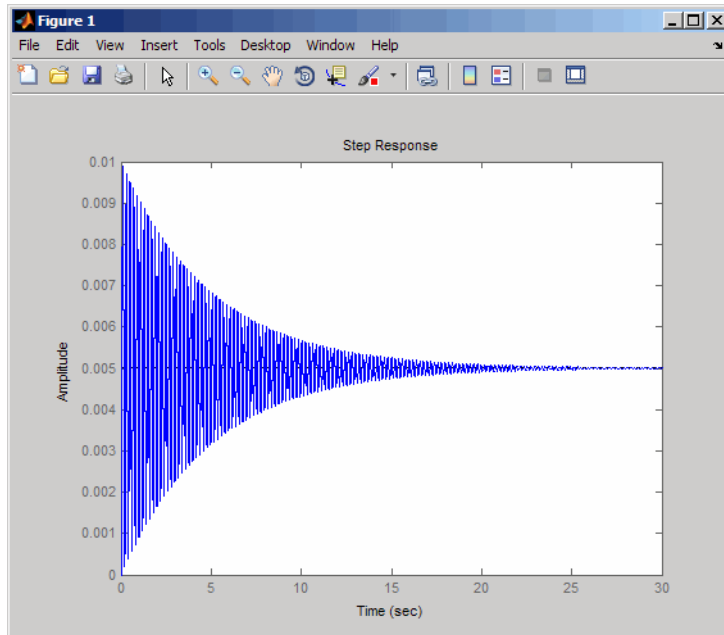
You can compare the Bode plots of the continuous and discretized models with

```
bodeplot(H, '-', Hd, '--')
```



To analyze the discrete system, plot its step response, type

```
step(Hd)
```



The system oscillates quite a bit. This is probably due to very light damping. You can check this by computing the open-loop poles. Type

```
% Open-loop poles of discrete model
damp(Hd)
```

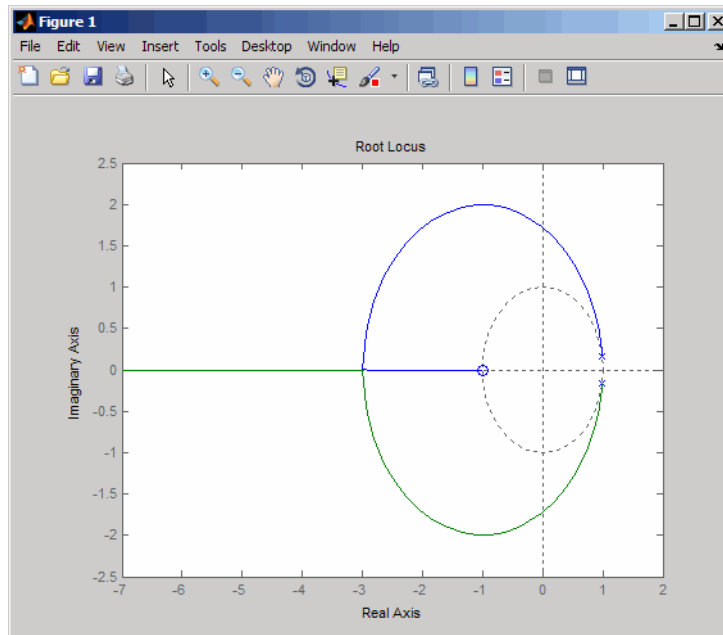
| Eigenvalue | Magnitude | Equiv. Damping | Equiv. Freq. |
|----------------------|-----------|----------------|--------------|
| 9.87e-01 + 1.57e-01i | 9.99e-01 | 6.32e-03 | 3.16e+01 |
| 9.87e-01 - 1.57e-01i | 9.99e-01 | 6.32e-03 | 3.16e+01 |

The poles have very light equivalent damping and are near the unit circle. You need to design a compensator that increases the damping of these poles.

Adding a Compensator Gain

The simplest compensator is just a gain, so try the root locus technique to select an appropriate feedback gain.

```
rlocus(Hd)
```



As shown in the root locus, the poles quickly leave the unit circle and go unstable. You need to introduce some lead or a compensator with some zeros.

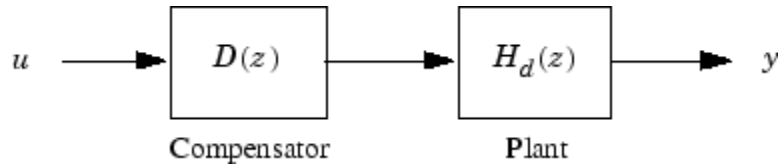
Adding a Lead Network

Try the compensator

$$D(z) = \frac{z + \alpha}{z + b}$$

with $\alpha = -0.85$ and $b = 0$.

The corresponding open-loop model



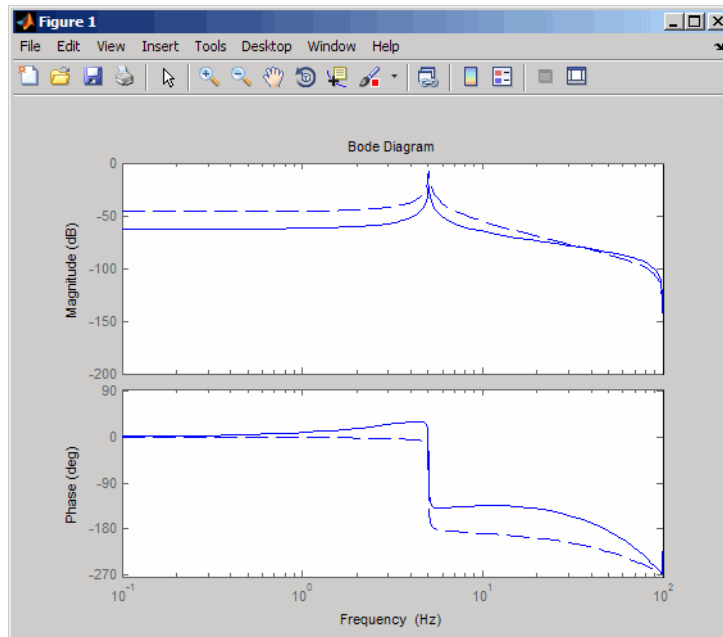
is obtained by the series connection

```
D = zpk(0.85,0,1,Ts)
olloop = Hd * D
```

Now see how this compensator modifies the open-loop frequency response.

```
bodeplot(Hd, '-', oloop, '-')
```

The plant response is the dashed line and the open-loop response with the compensator is the solid line.

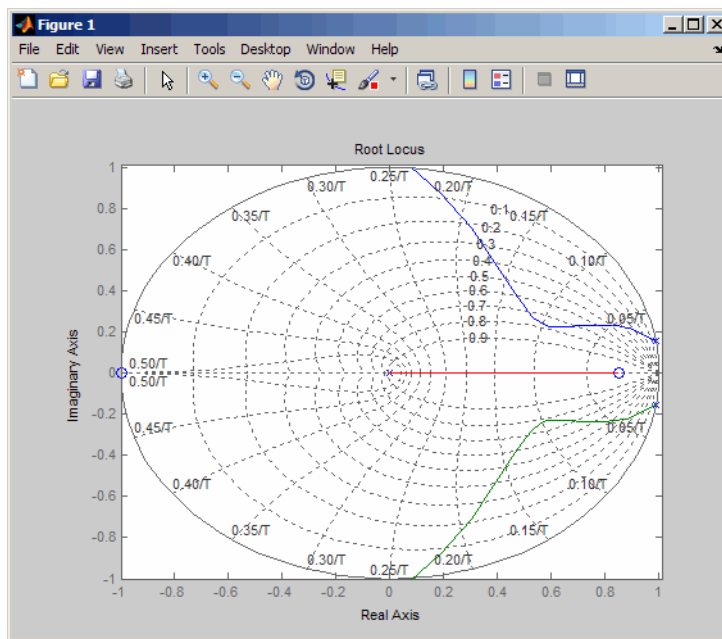


The plot above shows that the compensator has shifted up the phase plot (added lead) in the frequency range $\omega > 10$ rad/sec.

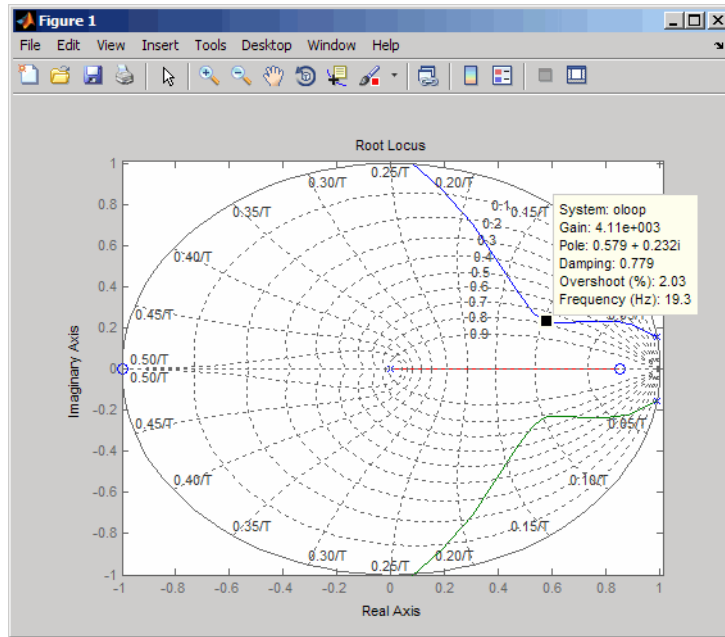
Now try the root locus again with the plant and compensator as open loop.

```
rlocus(olloop)
zgrid
```

Open the **Property Editor** by right-clicking in the plot away from the curve. On the **Limits** page, set the x - and y -axis limits from -1 to 1.01. This figure shows the result.



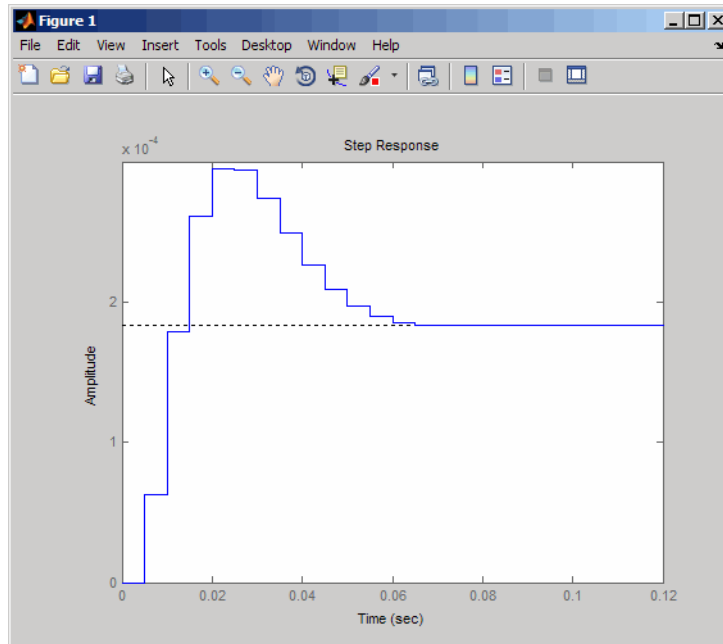
This time, the poles stay within the unit circle for some time (the lines drawn by `zgrid` show the damping ratios from $\zeta = 0$ to 1 in steps of 0.1). Use a data marker to find the point on the curve where the gain equals $4.111e+03$. This figure shows the data marker at the correct location.



Design Analysis

To analyze this design, form the closed-loop system and plot the closed-loop step response.

```
K = 4.11e+03;
cloop = feedback(olloop,K);
step(cloop)
```

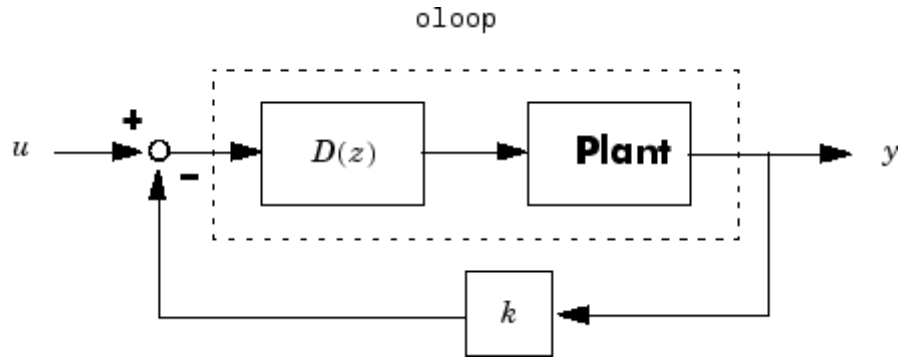


This response depends on your closed loop set point. The one shown here is relatively fast and settles in about 0.07 seconds. Therefore, this closed loop disk drive system has a seek time of about 0.07 seconds. This is slow by today's standards, but you also started with a very lightly damped system.

Now look at the robustness of your design. The most common classical robustness criteria are the gain and phase margins. Use the function `margin` to determine these margins. With output arguments, `margin` returns the gain and phase margins as well as the corresponding crossover frequencies. Without output argument, `margin` plots the Bode response and displays the margins graphically.

To compute the margins, first form the unity-feedback open loop by connecting the compensator $D(z)$, plant model, and feedback gain k in series.

```
olk = K * oloop;
```



Next apply margin to this open-loop model. Type

```
[Gm,Pm,Wcg,Wcp] = margin(olk);
Margins = [Gm Wcg Pm Wcp]
Margins =

    3.7987   296.7978   43.2031   106.2462
```

To obtain the gain margin in dB, type

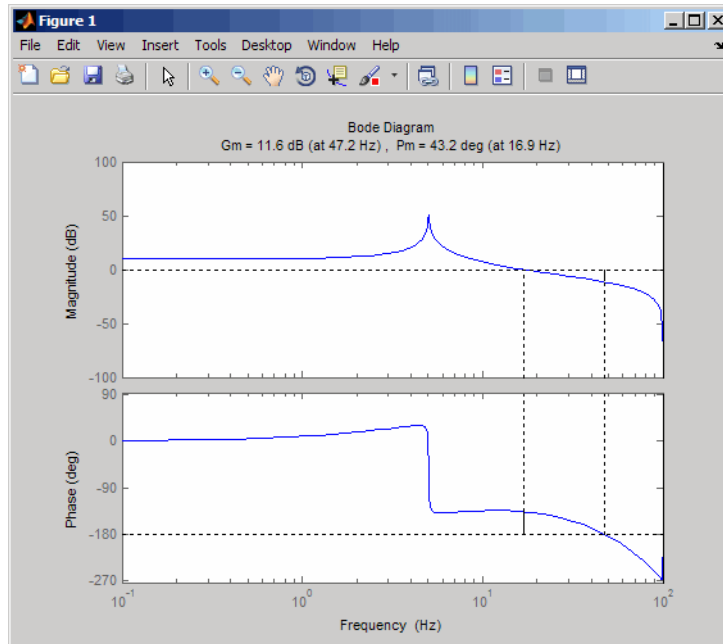
```
20*log10(Gm)
ans =

    11.5926
```

You can also display the margins graphically by typing

```
margin(olk)
```

The command produces the plot shown below.



This design is robust and can tolerate a 11 dB gain increase or a 40 degree phase lag in the open-loop system without going unstable. By continuing this design process, you may be able to find a compensator that stabilizes the open-loop system and allows you to reduce the seek time.

LQG Regulation: Rolling Mill Example

| In this section... |
|--|
| “Overview of this Case Study” on page 11-30 |
| “Process and Disturbance Models” on page 11-30 |
| “LQG Design for the x-Axis” on page 11-33 |
| “LQG Design for the y-Axis” on page 11-40 |
| “Cross-Coupling Between Axes” on page 11-42 |
| “MIMO LQG Design” on page 11-45 |

Overview of this Case Study

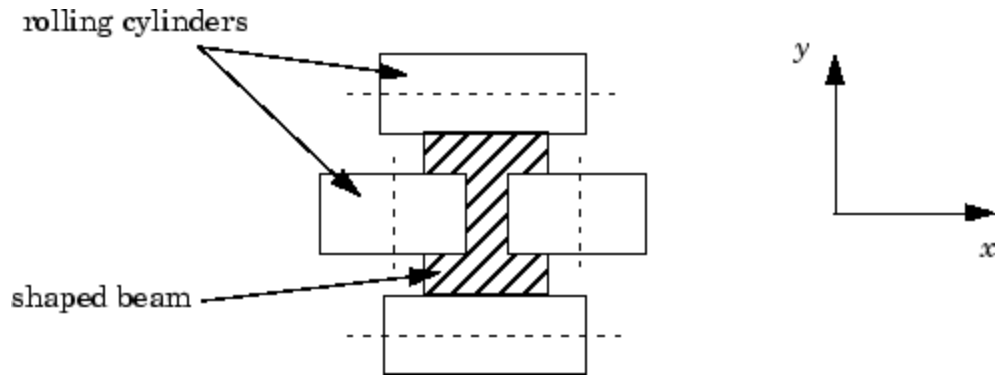
This case study demonstrates the use of the LQG design tools in a process control application. The goal is to regulate the horizontal and vertical thickness of the beam produced by a hot steel rolling mill. This example is adapted from [1]. The full plant model is MIMO and the example shows the advantage of direct MIMO LQG design over separate SISO designs for each axis. Type

```
milldemo
```

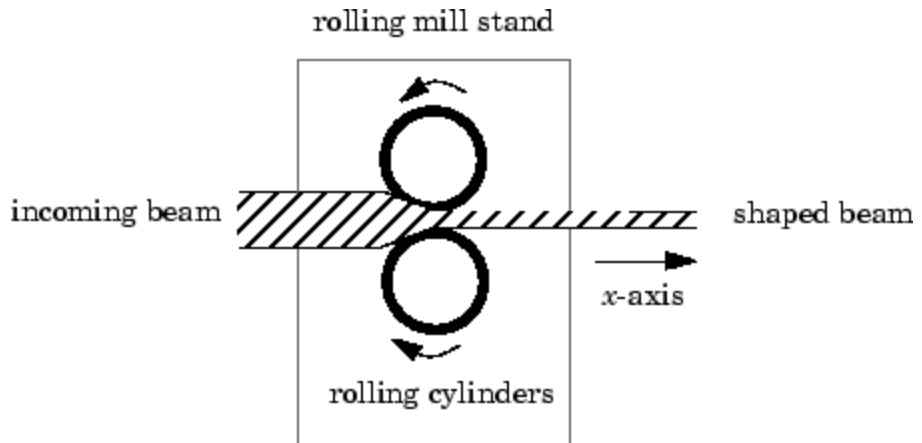
at the command line to run this demonstration interactively.

Process and Disturbance Models

The rolling mill is used to shape rectangular beams of hot metal. The desired outgoing shape is sketched below.



This shape is impressed by two pairs of rolling cylinders (one per axis) positioned by hydraulic actuators. The gap between the two cylinders is called the *roll gap*.



The objective is to maintain the beam thickness along the x - and y -axes within the quality assurance tolerances. Variations in output thickness can arise from the following:

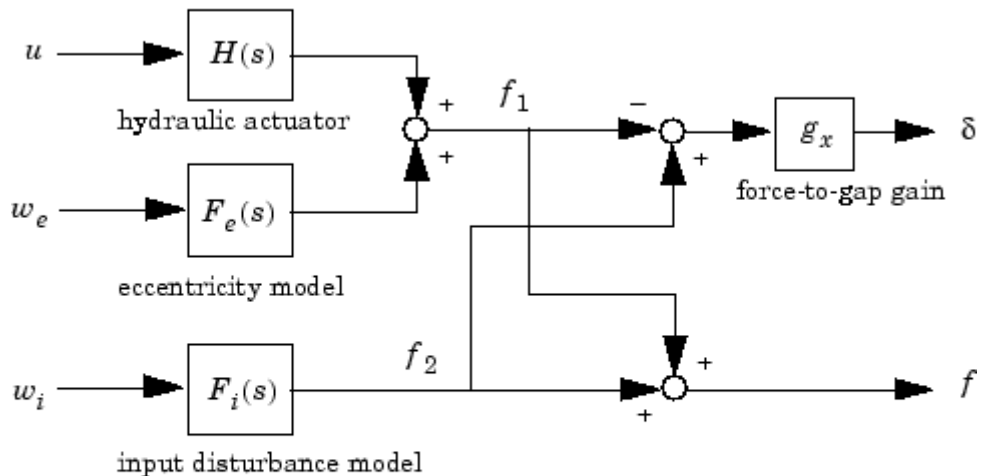
- Variations in the thickness/hardness of the incoming beam
- Eccentricity in the rolling cylinders

Feedback control is necessary to reduce the effect of these disturbances. Because the roll gap cannot be measured close to the mill stand, the rolling force is used instead for feedback.

The input thickness disturbance is modeled as a low pass filter driven by white noise. The eccentricity disturbance is approximately periodic and its frequency is a function of the rolling speed. A reasonable model for this disturbance is a second-order bandpass filter driven by white noise.

This leads to the following generic model for each axis of the rolling process.

Open-loop Model for x- or y-axis



- u **command**
- δ **thickness gap (in mm)**
- f **incremental rolling force**
- w_i, w_e **driving white noise for disturbance models**

The measured rolling force variation f is a combination of the incremental force delivered by the hydraulic actuator and of the disturbance forces due to eccentricity and input thickness variation. Note that:

- The outputs of $H(s)$, $F_e(s)$, and $F_i(s)$ are the incremental forces delivered by each component.
- An increase in hydraulic or eccentricity force *reduces* the output thickness gap δ .
- An increase in input thickness *increases* this gap.

The model data for each axis is summarized below.

Model Data for the x-Axis

$$H_x(s) = \frac{2.4 \times 10^8}{s^2 + 72s + 90^2}$$

$$F_{ix}(s) = \frac{10^4}{s + 0.05}$$

$$F_{ex}(s) = \frac{3 \times 10^4 s}{s^2 + 0.125s + 6^2}$$

$$g_x = 10^{-6}$$

Model Data for the y-Axis

$$H_y(s) = \frac{7.8 \times 10^8}{s^2 + 71s + 88^2}$$

$$F_{iy}(s) = \frac{2 \times 10^4}{s + 0.05}$$

$$F_{ey}(s) = \frac{10^5 s}{s^2 + 0.19s + 9.4^2}$$

$$g_y = 0.5 \times 10^{-6}$$

LQG Design for the x-Axis

As a first approximation, ignore the cross-coupling between the x- and y-axes and treat each axis independently. That is, design one SISO LQG regulator

for each axis. The design objective is to reduce the thickness variations δ_x and δ_y due to eccentricity and input thickness disturbances.

Start with the x -axis. First specify the model components as transfer function objects.

```
% Hydraulic actuator (with input "u-x")
Hx = tf(2.4e8,[1 72 90^2],'inputname','u-x')

% Input thickness/hardness disturbance model
Fix = tf(1e4,[1 0.05],'inputn','w-ix')

% Rolling eccentricity model
Fex = tf([3e4 0],[1 0.125 6^2],'inputn','w-ex')

% Gain from force to thickness gap
gx = 1e-6;
```

Next build the open-loop model shown in “Process and Disturbance Models” on page 11-30. You could use the function `connect` for this purpose, but it is easier to build this model by elementary `append` and `series` connections.

```
% I/O map from inputs to forces f1 and f2
Px = append([ss(Hx) Fex],Fix)

% Add static gain from f1,f2 to outputs "x-gap" and "x-force"
Px = [-gx gx;1 1] * Px

% Give names to the outputs:
set(Px,'outputn',{'x-gap' 'x-force'})
```

Note To obtain minimal state-space realizations, always convert transfer function models to state space *before* connecting them. Combining transfer functions and then converting to state space may produce nonminimal state-space models.

The variable `Px` now contains an open-loop state-space model complete with input and output names.

```
Px.inputname
```

```
ans =
    'u-x'
    'w-ex'
    'w-ix'
```

```
Px.outputname
```

```
ans =
    'x-gap'
    'x-force'
```

The second output 'x-force' is the rolling force measurement. The LQG regulator will use this measurement to drive the hydraulic actuator and reduce disturbance-induced thickness variations δ_x .

The LQG design involves two steps:

- 1 Design a full-state-feedback gain that minimizes an LQ performance measure of the form

$$J(u_x) = \int_0^{\infty} \{q\delta_x^2 + ru_x^2\} dt$$

- 2 Design a Kalman filter that estimates the state vector given the force measurements 'x-force'.

The performance criterion $J(u_x)$ penalizes low and high frequencies equally. Because low-frequency variations are of primary concern, eliminate the high-frequency content of δ_x with the low-pass filter $30/(s + 30)$ and use the filtered value in the LQ performance criterion.

```
lpf = tf(30,[1 30])
```

```
% Connect low-pass filter to first output of Px
Pxdes = append(lpf,1) * Px
set(Pxdes,'outputn',{ 'x-gap' 'x-force' })
```

```
% Design the state-feedback gain using LQR and q=1, r=1e-4
```

```
kx = lqry(Pxdes(1,1),1,1e-4)
```

Note `lqry` expects all inputs to be commands and all outputs to be measurements. Here the command 'u-x' and the measurement 'x-gap*' (filtered gap) are the first input and first output of `Pxdes`. Hence, use the syntax `Pxdes(1,1)` to specify just the I/O relation between 'u-x' and 'x-gap*'.

Next, design the Kalman estimator with the function `kalman`. The process noise

$$w_x = \begin{bmatrix} w_{ex} \\ w_{ix} \end{bmatrix}$$

has unit covariance by construction. Set the measurement noise covariance to 1000 to limit the high frequency gain, and keep only the measured output 'x-force' for estimator design.

```
estx = kalman(Pxdes(2,:),eye(2),1000)
```

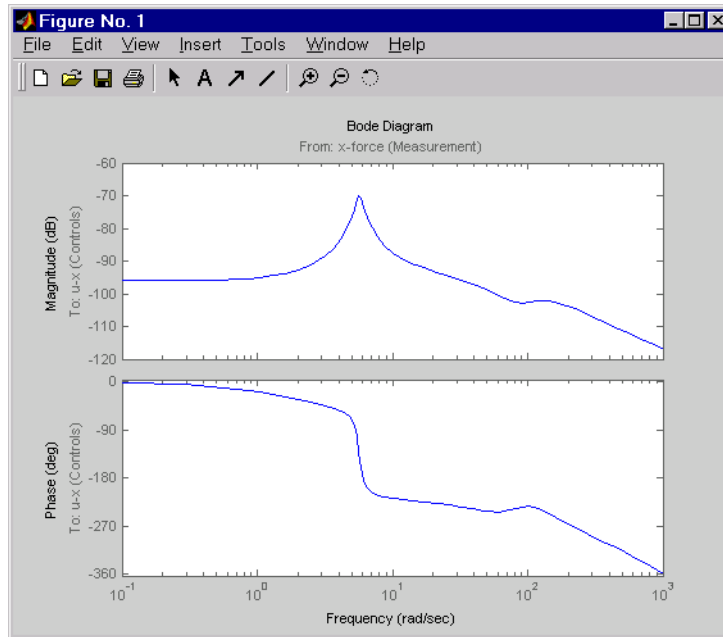
Finally, connect the state-feedback gain `kx` and state estimator `estx` to form the LQG regulator.

```
Regx = lqgreg(estx,kx)
```

This completes the LQG design for the x -axis.

Let's look at the regulator Bode response between 0.1 and 1000 rad/sec.

```
h = bodeplot(Regx,{0.1 1000})  
setoptions(h,'PhaseMatching','on')
```

The phase response has an interesting physical interpretation. First, consider an increase in input thickness. This low-frequency disturbance boosts both output thickness and rolling force. Because the regulator phase is approximately 0° at low frequencies, the feedback loop then adequately reacts by increasing the hydraulic force to offset the thickness increase. Now consider the effect of eccentricity. Eccentricity causes fluctuations in the roll gap (gap between the rolling cylinders). When the roll gap is minimal, the rolling force increases and the beam thickness diminishes. The hydraulic force must then be reduced (negative force feedback) to restore the desired thickness. This is exactly what the LQG regulator does as its phase drops to -180° near the natural frequency of the eccentricity disturbance (6 rad/sec).

Next, compare the open- and closed-loop responses from disturbance to thickness gap. Use `feedback` to close the loop. To help specify the feedback connection, look at the I/O names of the plant `Px` and regulator `Regx`.

```
Px.inputname
ans =
    'u-x'
```

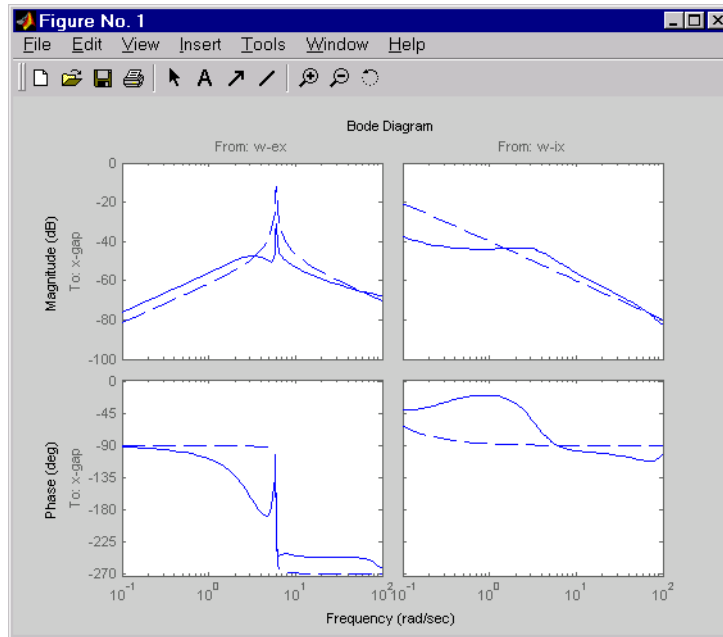
```
'w-ex'  
'w-ix'  
  
Regx.outputname  
ans =  
    'u-x'  
  
Px.outputname  
ans =  
    'x-gap'  
    'x-force'  
  
Regx.inputname  
ans =  
    'x-force'
```

This indicates that you must connect the first input and second output of Px to the regulator.

```
clx = feedback(Px,Regx,1,2,+1)    % Note: +1 for positive feedback
```

You are now ready to compare the open- and closed-loop Bode responses from disturbance to thickness gap.

```
h = bodeplot(Px(1,2:3),'--',clx(1,2:3),'-',{0.1 100})  
setoptions(h,'PhaseMatching','on')
```



The dashed lines show the open-loop response. Note that the peak gain of the eccentricity-to-gap response and the low-frequency gain of the input-thickness-to-gap response have been reduced by about 20 dB.

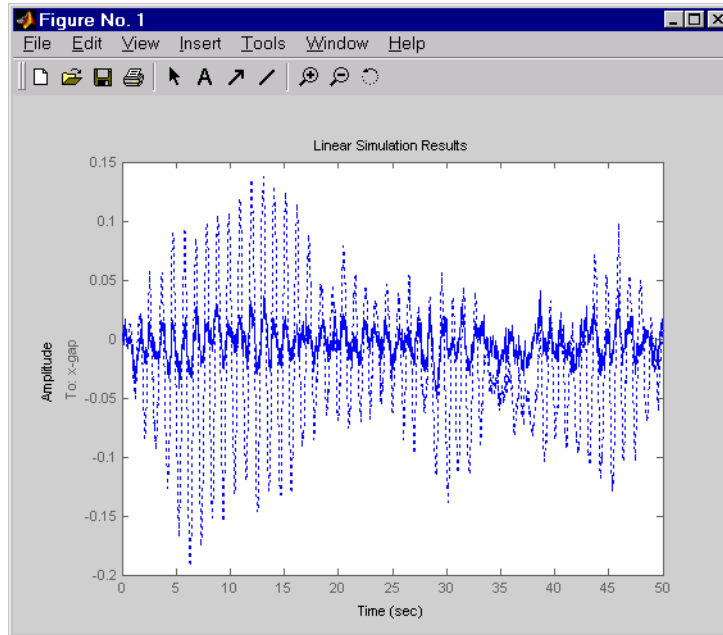
Finally, use `lsim` to simulate the open- and closed-loop time responses to the white noise inputs w_{ex} and w_{ix} . Choose $dt=0.01$ as sampling period for the simulation, and derive equivalent discrete white noise inputs for this sampling rate.

```
dt = 0.01
t = 0:dt:50    % time samples

% Generate unit-covariance driving noise wx = [w-ex;w-ix].
% Equivalent discrete covariance is 1/dt
wx = sqrt(1/dt) * randn(2,length(t))

lsim(Px(1,2:3), ':', clx(1,2:3), '-', wx, t)
```

Right-click on the plot that appears and select **Show Input** to turn off the display of the input.



The dotted lines correspond to the open-loop response. In this simulation, the LQG regulation reduces the peak thickness variation by a factor 4.

LQG Design for the y-Axis

The LQG design for the y-axis (regulation of the y thickness) follows the exact same steps as for the x-axis.

```
% Specify model components
Hy = tf(7.8e8,[1 71 88^2],'inputn','u-y')
Fiy = tf(2e4,[1 0.05],'inputn','w-iy')
Fey = tf([1e5 0],[1 0.19 9.4^2],'inputn','w-ey')
gy = 0.5e-6 % force-to-gap gain

% Build open-loop model
Py = append([ss(Hy) Fey],Fiy)
```

```

Py = [-gy gy;1 1] * Py
set(Py,'outputn',{'y-gap' 'y-force'})

% State-feedback gain design
Pydes = append(lpf,1) * Py % Add low-freq. weighing
set(Pydes,'outputn',{'y-gap*' 'y-force'})
ky = lqry(Pydes(1,1),1,1e-4)

% Kalman estimator design
esty = kalman(Pydes(2,:),eye(2),1e3)

% Form SISO LQG regulator for y-axis and close the loop
Regy = lqgreg(esty,ky)
cly = feedback(Py,Regy,1,2,+1)

```

Compare the open- and closed-loop response to the white noise input disturbances.

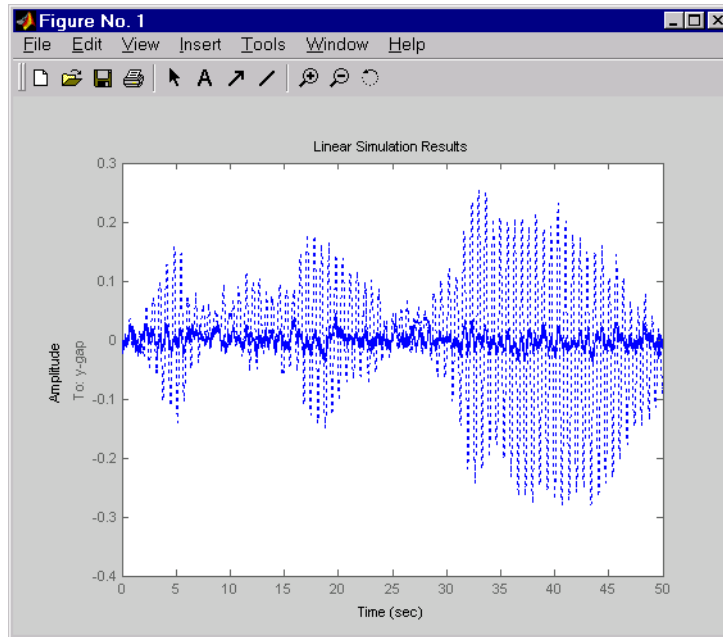
```

dt = 0.01
t = 0:dt:50
wy = sqrt(1/dt) * randn(2,length(t))

lsim(Py(1,2:3),':',cly(1,2:3),'-',wy,t)

```

Right-click on the plot that appears and select **Show Input** to turn off the display of the input.



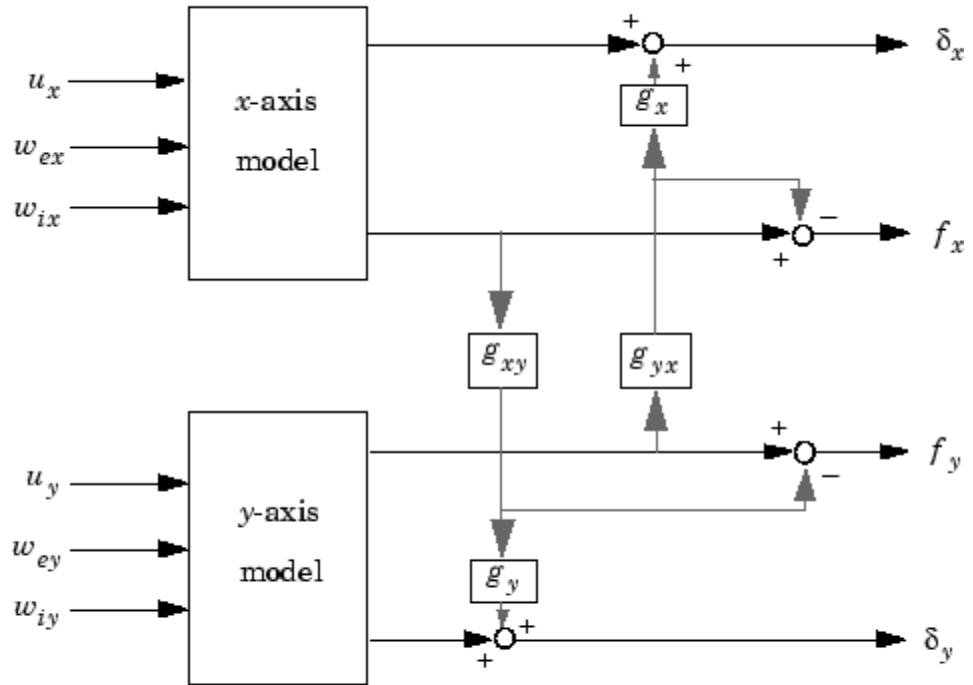
The dotted lines correspond to the open-loop response. The simulation results are comparable to those for the x -axis.

Cross-Coupling Between Axes

The x/y thickness regulation, is a MIMO problem. So far you have treated each axis separately and closed one SISO loop at a time. This design is valid as long as the two axes are fairly decoupled. Unfortunately, this rolling mill process exhibits some degree of cross-coupling between axes. Physically, an increase in hydraulic force along the x -axis compresses the material, which in turn boosts the repelling force on the y -axis cylinders. The result is an increase in y -thickness and an equivalent (relative) decrease in hydraulic force along the y -axis.

The figure below shows the coupling.

Coupling Between the x- and y-axes



$$g_{xy} = 0.1$$

$$g_{yx} = 0.4$$

Accordingly, the thickness gaps and rolling forces are related to the outputs $\bar{\delta}_x, \bar{f}_x, \dots$ of the x- and y-axis models by

$$\begin{bmatrix} \delta_x \\ \delta_y \\ f_x \\ f_x \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & g_{yx}g_x \\ 0 & 1 & g_{xy}g_y & 0 \\ 0 & 0 & 1 & -g_{yx} \\ 0 & 0 & -g_{xy} & 1 \end{bmatrix}}_{\text{cross-coupling matrix}} \begin{bmatrix} \bar{\delta}_x \\ \bar{\delta}_y \\ \bar{f}_x \\ \bar{f}_y \end{bmatrix}$$

Let's see how the previous "decoupled" LQG design fares when cross-coupling is taken into account. To build the two-axes model, shown above, append the models Px and Py for the x- and y-axes.

```
P = append(Px,Py)
```

For convenience, reorder the inputs and outputs so that the commands and thickness gaps appear first.

```
P = P([1 3 2 4],[1 4 2 3 5 6])
P.outputname
```

```
ans =
    'x-gap'
    'y-gap'
    'x-force'
    'y-force'
```

Finally, place the cross-coupling matrix in series with the outputs.

```
gxy = 0.1; gyx = 0.4;
CCmat = [eye(2) [0 gyx*gx;gxy*gy 0] ; zeros(2) [1 -gyx;-gxy 1]]
Pc = CCmat * P
Pc.outputname = P.outputname
```

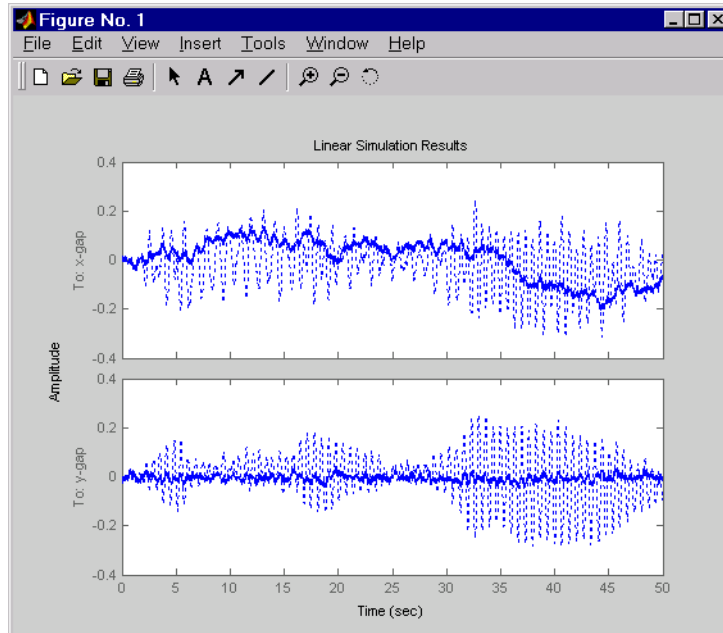
To simulate the closed-loop response, also form the closed-loop model by

```
feedin = 1:2 % first two inputs of Pc are the commands
feedout = 3:4 % last two outputs of Pc are the measurements
cl = feedback(Pc,append(Regx,Regy),feedin,feedout,+1)
```

You are now ready to simulate the open- and closed-loop responses to the driving white noises wx (for the x-axis) and wy (for the y-axis).


```
wxy = [wx ; wy]
lsim(Pc(1:2,3:6), ':-', cl(1:2,3:6), '- ', wxy, t)
```

Right-click on the plot that appears and select **Show Input** to turn off the display of the input.



The response reveals a severe deterioration in regulation performance along the x -axis (the peak thickness variation is about four times larger than in the simulation without cross-coupling). Hence, designing for one loop at a time is inadequate for this level of cross-coupling, and you must perform a joint-axis MIMO design to correctly handle coupling effects.

MIMO LQG Design

Start with the complete two-axis state-panespace model P_c derived in “Cross-Coupling Between Axes” on page 11-42. The model inputs and outputs are

```
Pc.inputname
```

```
ans =
    'u-x'
    'u-y'
    'w-ex'
    'w-ix'
    'w_ey'
    'w_iy'
```

```
P.outputname
```

```
ans =
    'x-gap'
    'y-gap'
    'x-force'
    'y-force'
```

As earlier, add low-pass filters in series with the 'x-gap' and 'y-gap' outputs to penalize only low-frequency thickness variations.

```
Pdes = append(lpf,lpf,eye(2)) * Pc
Pdes.outputn = Pc.outputn
```

Next, design the LQ gain and state estimator as before (there are now two commands and two measurements).

```
k = lqry(Pdes(1:2,1:2),eye(2),1e-4*eye(2))    % LQ gain
est = kalman(Pdes(3:4,:),eye(4),1e3*eye(2))  % Kalman estimator
```

```
RegMIMO = lqgreg(est,k)    % form MIMO LQG regulator
```

The resulting LQG regulator RegMIMO has two inputs and two outputs.

```
RegMIMO.inputname
```

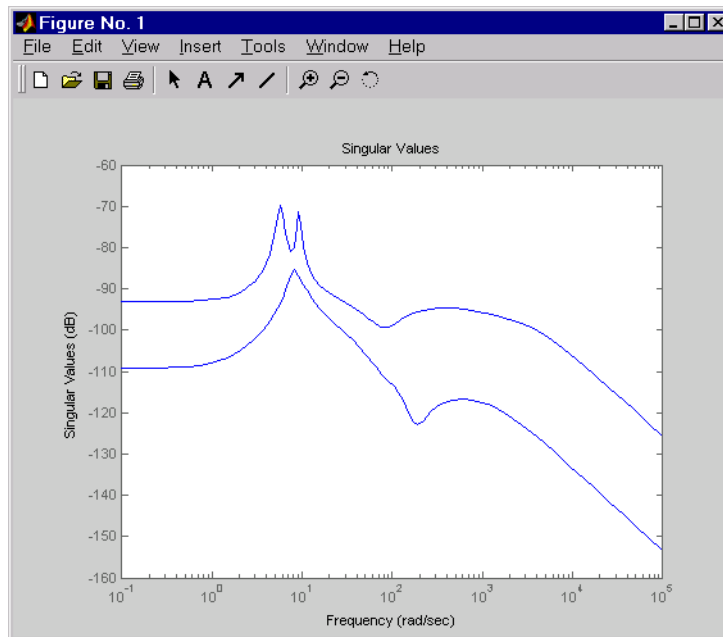
```
ans =
    'x-force'
    'y-force'
```

```
RegMIMO.outputname
```

```
ans =
    'u-x'
    'u-y'
```

Plot its singular value response (principal gains).

```
sigma(RegMIMO)
```

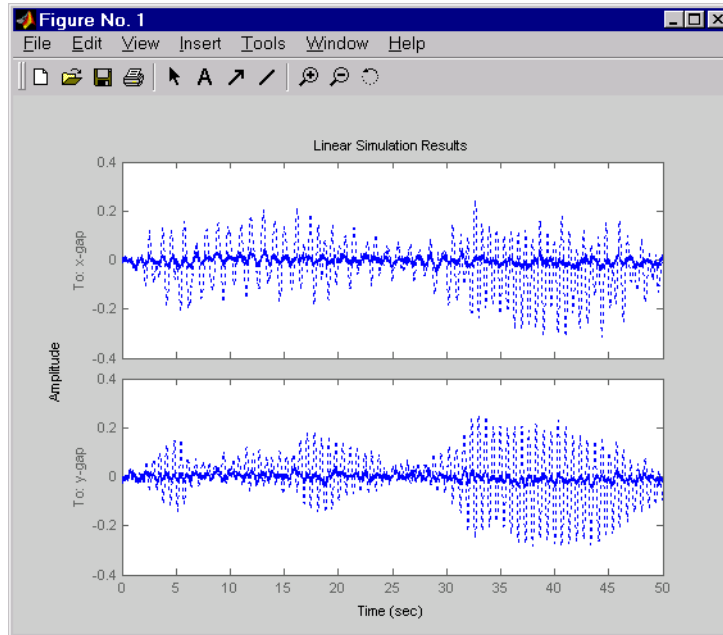


Next, plot the open- and closed-loop time responses to the white noise inputs (using the MIMO LQG regulator for feedback).

```
% Form the closed-loop model
cl = feedback(Pc,RegMIMO,1:2,3:4,+1);

% Simulate with lsim using same noise inputs
lsim(Pc(1:2,3:6),':',cl(1:2,3:6),'-',wxy,t)
```

Right-click on the plot that appears and select **Show Input** to turn off the display of the input.



The MIMO design is a clear improvement over the separate SISO designs for each axis. In particular, the level of x/y thickness variation is now comparable to that obtained in the decoupled case. This example illustrates the benefits of direct MIMO design for multivariable systems.

Kalman Filtering

In this section...

“Overview of this Case Study” on page 11-49

“Discrete Kalman Filter” on page 11-50

“Steady-State Design” on page 11-51

“Time-Varying Kalman Filter” on page 11-57

“Time-Varying Design” on page 11-58

“References” on page 11-62

Overview of this Case Study

This final case study illustrates Kalman filter design and simulation. Both steady-state and time-varying Kalman filters are considered.

Consider the discrete plant

$$\begin{aligned}x[n+1] &= Ax[n] + B(u[n] + w[n]) \\ y[n] &= Cx[n]\end{aligned}$$

with additive Gaussian noise $w[n]$ on the input $u[n]$ and data

$$A = \begin{bmatrix} 1.1269 & -0.4940 & 0.1129 \\ 1.0000 & 0 & 0 \\ 0 & 1.0000 & 0 \end{bmatrix};$$

$$B = \begin{bmatrix} -0.3832 \\ 0.5919 \\ 0.5191 \end{bmatrix};$$

$$C = [1 \ 0 \ 0];$$

Our goal is to design a Kalman filter that estimates the output $y[n]$ given the inputs $u[n]$ and the noisy output measurements

$$y_v[n] = Cx[n] + v[n]$$

where $v[n]$ is some Gaussian white noise.

Discrete Kalman Filter

The equations of the steady-state Kalman filter for this problem are given as follows.

Measurement update

$$\hat{x}[n|n] = \hat{x}[n|n-1] + M(y_v[n] - C\hat{x}[n|n-1])$$

Time update

$$\hat{x}[n+1|n] = A\hat{x}[n|n] + Bu[n]$$

In these equations:

- $\hat{x}[n|n-1]$ is the estimate of $x[n]$ given past measurements up to $y_v[n-1]$
- $\hat{x}[n|n]$ is the updated estimate based on the last measurement $y_v[n]$

Given the current estimate $\hat{x}[n|n]$, the time update predicts the state value at the next sample $n+1$ (one-step-ahead predictor). The measurement update then adjusts this prediction based on the new measurement $y_v[n+1]$. The correction term is a function of the *innovation*, that is, the discrepancy.

$$y_v[n+1] - C\hat{x}[n+1|n]$$

between the measured and predicted values of $y[n+1]$. The innovation gain M is chosen to minimize the steady-state covariance of the estimation error given the noise covariances

$$E(w[n]w[n]^T) = Q \quad E(v[n]v[n]^T) = R \quad N = E(w[n]v[n]^T) = 0$$

You can combine the time and measurement update equations into one state-space model (the Kalman filter).

$$\hat{x}[n+1|n] = A(I - MC)\hat{x}[n|n-1] + [B \quad AM] \begin{bmatrix} u[n] \\ y_v[n] \end{bmatrix}$$

$$\hat{y}[n|n] = C(I - MC)\hat{x}[n|n-1] + CM y_v[n]$$

This filter generates an optimal estimate $\hat{y}[n|n]$ of $y[n]$. Note that the filter state is $\hat{x}[n|n-1]$.

Steady-State Design

You can design the steady-state Kalman filter described above with the function `kalman`. First specify the plant model with the process noise.

$$x[n+1] = Ax[n] + Bu[n] + Bw[n] \quad (\text{state equation})$$

$$y[n] = Cx[n] \quad (\text{measurement equation})$$

This is done by

```
% Note: set sample time to -1 to mark model as discrete
Plant = ss(A,[B B],C,0,-1,'inputname',{ 'u' 'w'},...
           'outputname','y');
```

Assuming that $Q = R = 1$, you can now design the discrete Kalman filter by

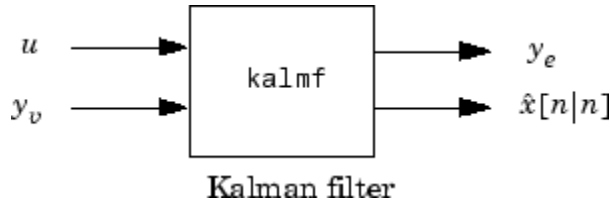
```
Q = 1; R = 1;
[kalmf,L,P,M] = kalman(Plant,Q,R);
```

This returns a state-space model `kalmf` of the filter as well as the innovation gain

```
M
M =
0.3798
```

0.0817
-0.2570

The inputs of `kalmf` are u and y_v , and its outputs are the plant output and state estimates $y_e = \hat{y}[n|n]$ and $\hat{x}[n|n]$.



Because you are interested in the output estimate y_e , keep only the first output of `kalmf`. Type

```

kalmf = kalmf(1,:);
kalmf
a =
      x1_e      x2_e      x3_e
x1_e  0.7683   -0.494    0.1129
x2_e  0.6202         0         0
x3_e -0.08173         1         0

b =
      u      y
x1_e -0.3832  0.3586
x2_e  0.5919  0.3798
x3_e  0.5191  0.08173

c =
      x1_e      x2_e      x3_e
y_e  0.6202         0         0

d =
      u      y
y_e  0  0.3798
  
```

Input groups:
Name Channels


```

KnownInput      1
Measurement     2

```

```

Output groups:
  Name          Channels
OutputEstimate 1

```

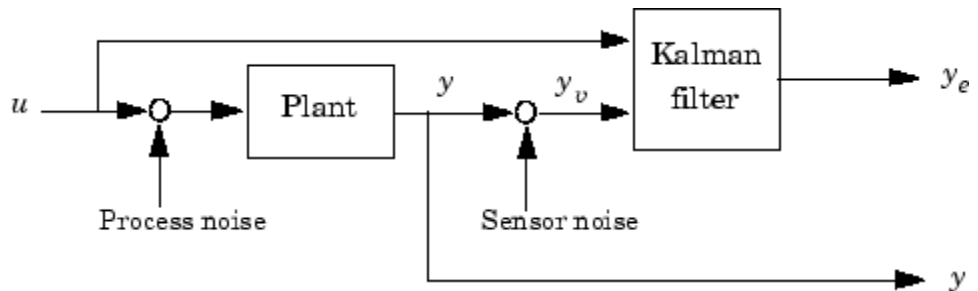
```

Sampling time: unspecified
Discrete-time model.

```

To see how the filter works, generate some input data and random noise and compare the filtered response y_e with the true response y . You can either generate each response separately, or generate both together. To simulate each response separately, use `lsim` with the plant alone first, and then with the plant and filter hooked up together. The joint simulation alternative is detailed next.

The block diagram below shows how to generate both true and filtered outputs.



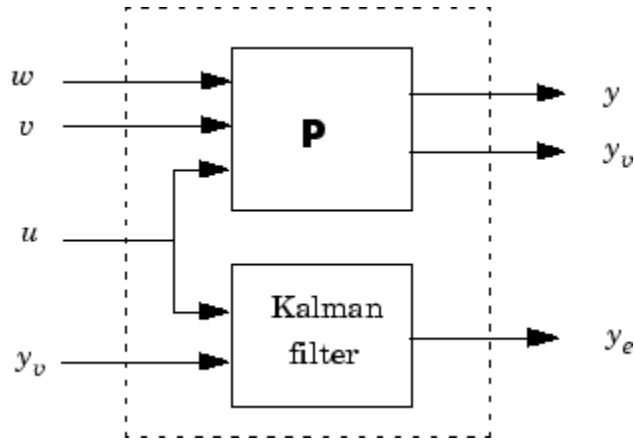
You can construct a state-space model of this block diagram with the functions `parallel` and `feedback`. First build a complete plant model with u , w , v as inputs y and y_v (measurements) as outputs.

```

a = A;
b = [B B 0*B];
c = [C;C];
d = [0 0 0;0 0 1];
P = ss(a,b,c,d,-1,'inputname',{'u' 'w' 'v'},...
'outputname',{'y' 'yv'});

```

Then use `parallel` to form the following parallel connection.



```
sys = parallel(P,kalmf,1,1,[],[])
```

Finally, close the sensor loop by connecting the plant output y_v to the filter input y_v with positive feedback.

```
% Close loop around input #4 and output #2
SimModel = feedback(sys,1,4,2,1)
% Delete yv from I/O list
SimModel = SimModel([1 3],[1 2 3])
```

The resulting simulation model has w , v , u as inputs and y , y_e as outputs.

```
SimModel.inputname
```

```
ans =
    'w'
    'v'
    'u'
```

```
SimModel.outputname
```

```
ans =
    'y'
    'y_e'
```

You are now ready to simulate the filter behavior. Generate a sinusoidal input u and process and measurement noise vectors w and v .

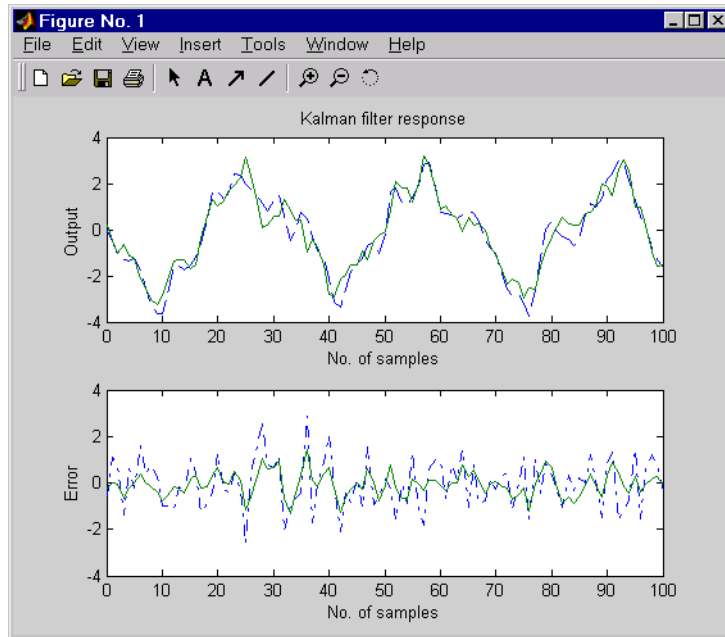
```
t = [0:100]';  
u = sin(t/5);  
  
n = length(t)  
randn('seed',0)  
w = sqrt(Q)*randn(n,1);  
v = sqrt(R)*randn(n,1);
```

Now simulate with `lsim`.

```
[out,x] = lsim(SimModel,[w,v,u]);  
  
y = out(:,1); % true response  
ye = out(:,2); % filtered response  
yv = y + v; % measured response
```

and compare the true and filtered responses graphically.

```
subplot(211), plot(t,y,'--',t,ye,'-'),  
xlabel('No. of samples'), ylabel('Output')  
title('Kalman filter response')  
subplot(212), plot(t,y-yv,'-.',t,y-ye,'-'),  
xlabel('No. of samples'), ylabel('Error')
```



The first plot shows the true response y (dashed line) and the filtered output y_e (solid line). The second plot compares the measurement error (dash-dot) with the estimation error (solid). This plot shows that the noise level has been significantly reduced. This is confirmed by the following error covariance computations.

```

MeasErr = y-yv;
MeasErrCov = sum(MeasErr.*MeasErr)/length(MeasErr);
EstErr = y-ye;
EstErrCov = sum(EstErr.*EstErr)/length(EstErr);

```

The error covariance before filtering (measurement error) is

```
MeasErrCov
```

```

MeasErrCov =
    1.1138

```

while the error covariance after filtering (estimation error) is only

EstErrCov

EstErrCov =
0.2722

Time-Varying Kalman Filter

The time-varying Kalman filter is a generalization of the steady-state filter for time-varying systems or LTI systems with nonstationary noise covariance. Given the plant state and measurement equations

$$\begin{aligned}x[n+1] &= Ax[n] + Bu[n] + Gw[n] \\y_v[n] &= Cx[n] + v[n]\end{aligned}$$

the time-varying Kalman filter is given by the recursions

Measurement update

$$\begin{aligned}\hat{x}[n|n] &= \hat{x}[n|n-1] + M[n](y_v[n] - C\hat{x}[n|n-1]) \\M[n] &= P[n|n-1]C^T(R[n] + CP[n|n-1]C^T)^{-1} \\P[n|n] &= (I - M[n]C)P[n|n-1]\end{aligned}$$

Time update

$$\begin{aligned}\hat{x}[n+1|n] &= A\hat{x}[n|n] + Bu[n] \\P[n+1|n] &= AP[n|n]A^T + GQ[n]G^T\end{aligned}$$

with $\hat{x}[n|n-1]$ and $\hat{x}[n|n]$ as defined in “Discrete Kalman Filter” on page 11-50, and in the following.

$$Q[n] = E(w[n]w[n]^T)$$

$$R[n] = E(v[n]v[n]^T)$$

$$P[n|n] = E(\{x[n] - x[n|n]\}\{x[n] - x[n|n]\}^T)$$

$$P[n|n-1] = E(\{x[n] - x[n|n-1]\}\{x[n] - x[n|n-1]\}^T)$$

For simplicity, we have dropped the subscripts indicating the time dependence of the state-space matrices.

Given initial conditions $x[1|0]$ and $P[1|0]$, you can iterate these equations to perform the filtering. Note that you must update both the state estimates

$x[n|.]$ and error covariance matrices $P[n|.]$ at each time sample.

Time-Varying Design

Although the Control System Toolbox software does not offer specific commands to perform time-varying Kalman filtering, it is easy to implement the filter recursions in the MATLAB environment. This section shows how to do this for the stationary plant considered above.

First generate noisy output measurements

```
% Use process noise w and measurement noise v generated above
sys = ss(A,B,C,0,-1);
y = lsim(sys,u+w);      % w = process noise
yv = y + v;           % v = measurement noise
```

Given the initial conditions

$$x[1|0] = 0, \quad P[1|0] = BQB^T$$

you can implement the time-varying filter with the following for loop.

```
P = B*Q*B';          % Initial error covariance
x = zeros(3,1);     % Initial condition on the state
ye = zeros(length(t),1);
ycov = zeros(length(t),1);
```

```

for i=1:length(t)
    % Measurement update
    Mn = P*C'/(C*P*C'+R);
    x = x + Mn*(yv(i)-C*x);    % x[n|n]
    P = (eye(3)-Mn*C)*P;      % P[n|n]

    ye(i) = C*x;
    errcov(i) = C*P*C';

    % Time update
    x = A*x + B*u(i);         % x[n+1|n]
    P = A*P*A' + B*Q*B';     % P[n+1|n]
end

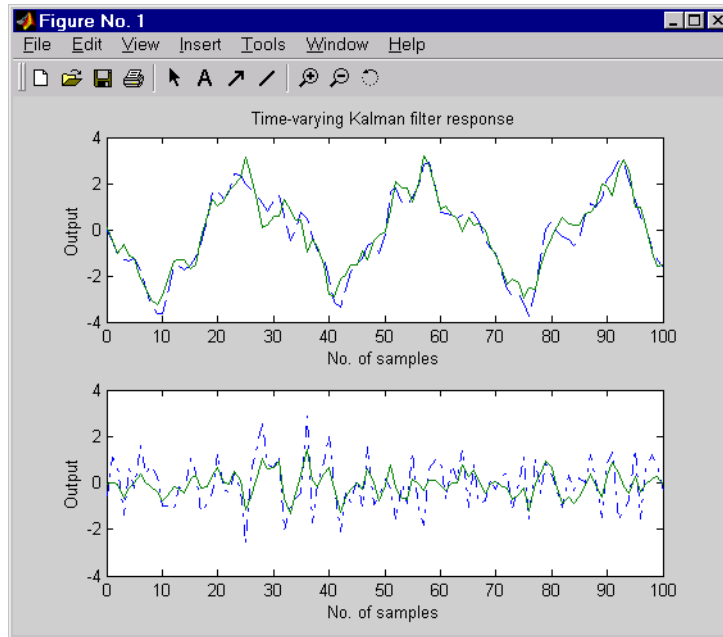
```

You can now compare the true and estimated output graphically.

```

subplot(211), plot(t,y,'--',t,ye,'-')
title('Time-varying Kalman filter response')
xlabel('No. of samples'), ylabel('Output')
subplot(212), plot(t,y-yv,'-.',t,y-ye,'-')
xlabel('No. of samples'), ylabel('Output')

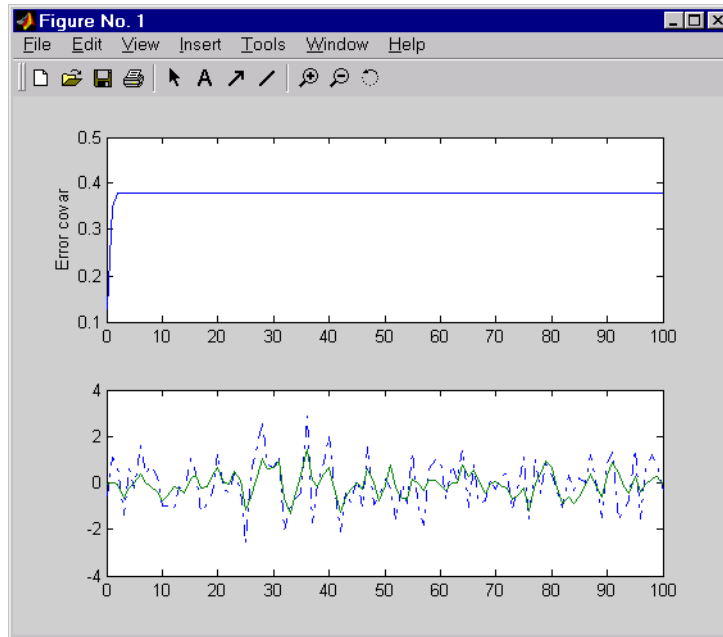
```



The first plot shows the true response y (dashed line) and the filtered response y_e (solid line). The second plot compares the measurement error (dash-dot) with the estimation error (solid).

The time-varying filter also estimates the covariance errcov of the estimation error $y - y_e$ at each sample. Plot it to see if your filter reached steady state (as you expect with stationary input noise).

```
subplot(211)
plot(t,errcov), ylabel('Error covar')
```

From this covariance plot, you can see that the output covariance did indeed reach a steady state in about five samples. From then on, your time-varying filter has the same performance as the steady-state version.

Compare with the estimation error covariance derived from the experimental data. Type

```
EstErr = y-ye;
EstErrCov = sum(EstErr.*EstErr)/length(EstErr)
EstErrCov =
    0.2718
```

This value is smaller than the theoretical value `errcov` and close to the value obtained for the steady-state design.

Finally, note that the final value $M[n]$ and the steady-state value M of the innovation gain matrix coincide.

M_n, M

$$\begin{aligned} M_n = & \\ & 0.3798 \\ & 0.0817 \\ & -0.2570 \end{aligned}$$

$$\begin{aligned} M = & \\ & 0.3798 \\ & 0.0817 \\ & -0.2570 \end{aligned}$$

References

- [1] [Grimble, M.J., *Robust Industrial Control: Optimal Design Approach for Polynomial Systems*, Prentice Hall, 1994, p. 261 and pp. 443-456.

Reliable Computations

- “Scaling State-Space Models” on page 12-2
- “How To Get Accurate Results” on page 12-8

Scaling State-Space Models

In this section...

“Why Scaling Is Important” on page 12-2

“When to Scale Your Model” on page 12-2

“Manually Scaling Your Model” on page 12-3

Why Scaling Is Important

When working with state-space models, proper scaling is important for accurate computations. A state-space model is well scaled when the following conditions exist:

- The entries of the A , B , and C matrices are homogenous in magnitude.
- The model characteristics are insensitive to small perturbations in A , B , and C (in comparison to their norms).

Working with poorly scaled models can cause your model a severe loss of accuracy and puzzling results. An example of a poorly scaled model is a dynamic system with two states in the state vector that have units of light years and millimeters. Such disparate units may introduce both very large and very small entries into the A matrix. Over the course of computations, this mix of small and large entries in the matrix could destroy important characteristics of the model and lead to incorrect results.

For more information on the harmful affects of a poorly scaled model, see the [Scaling Models to Maximize Accuracy](#) demo.

When to Scale Your Model

You can avoid scaling issues altogether by carefully selecting units to reduce the spread between small and large coefficients.

In general, you do not have to perform your own scaling when using the Control System Toolbox software. The algorithms automatically scale your model to prevent loss of accuracy. The automated scaling chooses a frequency range to maximize accuracy based on the dominant dynamics of the model.

In most cases, automated scaling provides high accuracy without your intervention. For some models with dynamics spanning a wide frequency range, however, it is impossible to achieve good accuracy at *all* frequencies and some tradeoff of accuracy in different frequency bands is necessary. In such cases, a warning alerts you of potential inaccuracies. If you receive this warning, evaluate the tradeoffs and consider manually adjusting the frequency interval where you most need high accuracy. For information on how to manually scale your model, see “Manually Scaling Your Model” on page 12-3.

Note For models with satisfactory scaling, you can bypass automated scaling in the Control System Toolbox software. To do so, set the `Scaled` property of your state-space model to 1 (true). For information on how to set this property, see the `set` reference page.

Manually Scaling Your Model

If automatic scaling produces a warning, you can use the `prescale` command to manually scale your model and adjust the frequency interval where you most need high accuracy.

The `prescale` command includes a Scaling Tool GUI, which you can use to visualize accuracy tradeoffs and to adjust the frequency interval where this accuracy is maximized.

To scale your model using the Scaling Tool GUI, you perform the following steps:

- “Opening the Scaling Tool GUI” on page 12-4
- “Specifying the Frequency Axis Limits in the Scaling Tool GUI” on page 12-5
- “Specifying the Frequency Band for Maximum Accuracy in the Scaling Tool GUI” on page 12-6
- “Saving the Scaling in the Scaling Tool GUI” on page 12-6

For an example of using the Scaling Tool GUI on a real model, see the Scaling Models to Maximize Accuracy demo.

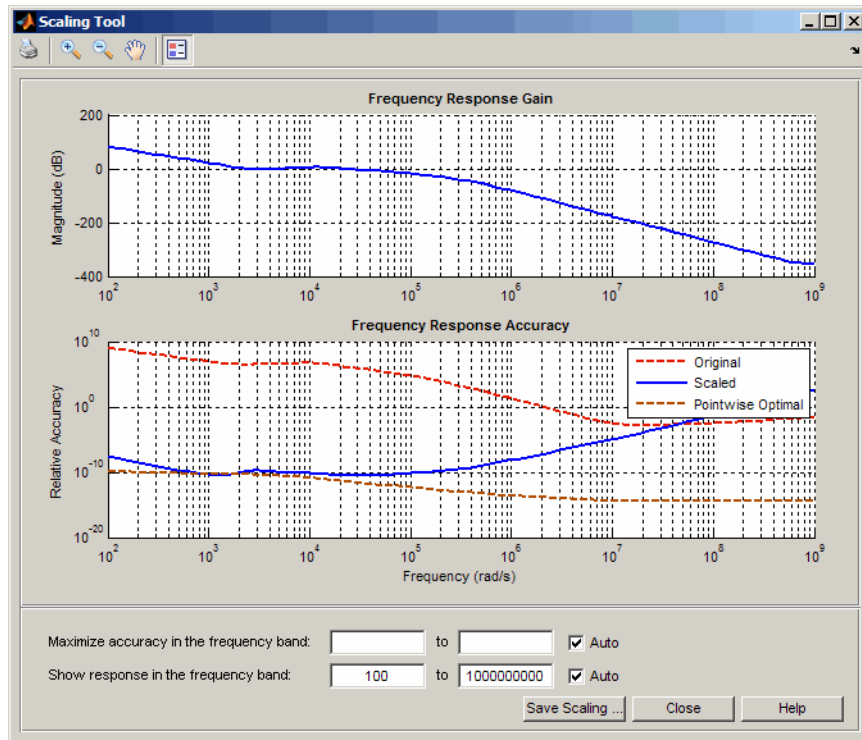
For more information about scaling models from the command line, see the `prescale` reference page.

Opening the Scaling Tool GUI

To open the Scaling Tool GUI for a state-space model named `sys`, type

```
prescale(sys)
```

The Scaling Tool GUI resembles one shown in the following figure.



The Scaling Tool GUI contains the following plots:

- The **Frequency Response Gain** plot helps you determine the frequency band over which you want to maximize scaling.

For SISO systems, this plot shows the gain of your model. For MIMO systems, the plot shows the principle gain (largest singular value) of your model.

- The **Frequency Response Accuracy** plot allows you to view the accuracy tradeoffs for your model when maximizing accuracy in a particular frequency bands.

This plot shows the following information:

- Relative accuracy of the response of the original unscaled model in red
- Relative accuracy of the response of the scaled model in blue
- Best achievable accuracy when using independent scaling at each frequency in brown

When you compute some model characteristics, such as the frequency response or the system zeros, the software produces the exact answer for some perturbation of the model you specified. The *relative accuracy* is a measure of the worst-case relative gap between the frequency response of the original and perturbed models. The perturbation accounts for rounding errors during calculation. Any relative accuracy value greater than 1 implies poor accuracy.

Tip If the blue Scaled curve is close to the brown Pointwise Optimal curve in a particular frequency band, you already have the best possible accuracy in that frequency band.

Specifying the Frequency Axis Limits in the Scaling Tool GUI

You can change the limits of the plot axis to view a particular frequency band of interest in the Scaling Tool GUI. To view a particular frequency band, specify the band in the **Show response in the frequency band** fields.

This action updates the frequency axis of the Scaling tool to show the specified frequency band.

Tip To return to the default display, select the **Auto** check box.

Specifying the Frequency Band for Maximum Accuracy in the Scaling Tool GUI

To adjust the frequency band where you want maximum accuracy, set a new frequency band in the **Maximize accuracy in the frequency band** fields. You can visualize accuracy tradeoffs by trying out different frequency bands and viewing the resulting relative accuracy across the frequency band of interest.

Note You can use the **Frequency Response Gain** plot, which plots the gain of your model, to view the dynamics in your model to help determine the frequency band to maximize accuracy.

Each time you specify a new frequency band, the **Frequency Response Accuracy** plot updates with the result of the new scaling. Compare the Scaled curve (blue) to the Pointwise Optimal curve (brown) to determine where the new scaling is nearly optimal and where you need more accuracy.

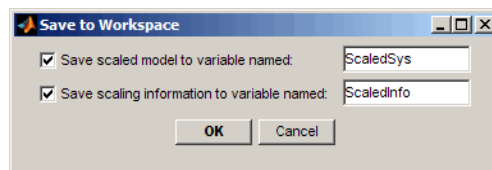
Tip To return to the default scaling, select the **Auto** check box.

Saving the Scaling in the Scaling Tool GUI

When you find a good scaling for your model, save the scaled model as follows:

- 1 Click **Save Scaling**.

This action opens the **Save to Workspace** dialog box.



- 2 In the **Save to Workspace** dialog box, verify that any of the following items you want to save are selected, and specify variable names for these items.

- Scaled model
- Scaling information, including:
 - Scaling factors
 - Frequencies used to test accuracy
 - Relative accuracy at each test frequencyFor details about the scaling information, see the `prescale` reference page.

3 Click **OK**.

This action sets the State-Space (@ss) object `Scaled` property of your model to true. When you set this property to `True`, the Control System Toolbox algorithms skip the automated scaling of the model.

How To Get Accurate Results

To ensure that you get accurate results, watch for the issues described in the How to Get Accurate Results category of demos.

Using the SISO Design Tool and the LTI Viewer

- Chapter 13, “SISO Design Tool”
- Chapter 14, “LTI Viewer”

SISO Design Tool

- “Overview of the SISO Design Tool” on page 13-2
- “Opening the SISO Design Tool” on page 13-3
- “Using the SISO Design Task Node” on page 13-4
- “Using the SISO Design Task in the Controls & Estimation Tools Manager” on page 13-11
- “SISO Design Task Graphical Tuning Window” on page 13-42
- “Using the Graphical Tuning Window Menu Bar” on page 13-44
- “Using the Graphical Tuning Window Toolbar” on page 13-56
- “Using the Right-Click Menus in the Graphical Tuning Window” on page 13-57
- “LTI Viewer for SISO Design Task Design Requirements” on page 13-79

Overview of the SISO Design Tool

The SISO Design Tool is a graphical-user interface (GUI) to design compensators.

The SISO Design Tool has the following components:

- The SISO Design Task Node in the Control and Estimation Tools Manager, a user interface (UI) that facilitates the design of compensators for single-input, single-output feedback loops through a series of interactive panes.
- The Graphical Tuning Window, a graphical user interface (GUI) for displaying and manipulating the Bode, root locus, and Nichols plots for the controller currently being designed. This window is titled SISO Design for *Design Name*. The Graphical Tuning Window by default displays the root locus and Bode diagrams for your imported systems. The two are dynamically linked; for example, if you change the gain in the root locus, it immediately affects the Bode diagrams as well.
- The SISO Design Task-associated LTI Viewer (For instructions on how to operate the LTI Viewer, see Chapter 14, “LTI Viewer”)
- A tool that automatically generates compensators using PID, internal model control (IMC), or linear-quadratic-Gaussian (LQG) methods.
- Optimization-based tuning methods that automatically tune the system to satisfy design requirements (available if you have Simulink Design Optimization software installed).

See “Designing Compensators” in the *Control System Toolbox Getting Started Guide* to learn about the typical design tasks you can perform using the SISO Design Tool. The following topics describe all available options for the SISO Design Tool.

Opening the SISO Design Tool

To open the SISO Design Task node in the Control and Estimation Tools Manager and the Graphical Tuning Window, type

```
sisotool
```

Load the Gservo model that you want to control in the MATLAB workspace by typing

```
load ltiexamples
```

In the **Architecture** tab of the Control and Estimation Tools Manager GUI, click **System Data** to import the model into the SISO Design Tool.

Tip You can import LTI models and arrays of LTI models for the plant **G** and sensor **H** into the SISO Design Tool. For more information, see “Importing Models into the SISO Design Tool” in the *Getting Started Guide*.

Alternatively, to open the SISO Design Tool with the Gservo system, type

```
sisotool(Gservo)
```

Using the SISO Design Task Node

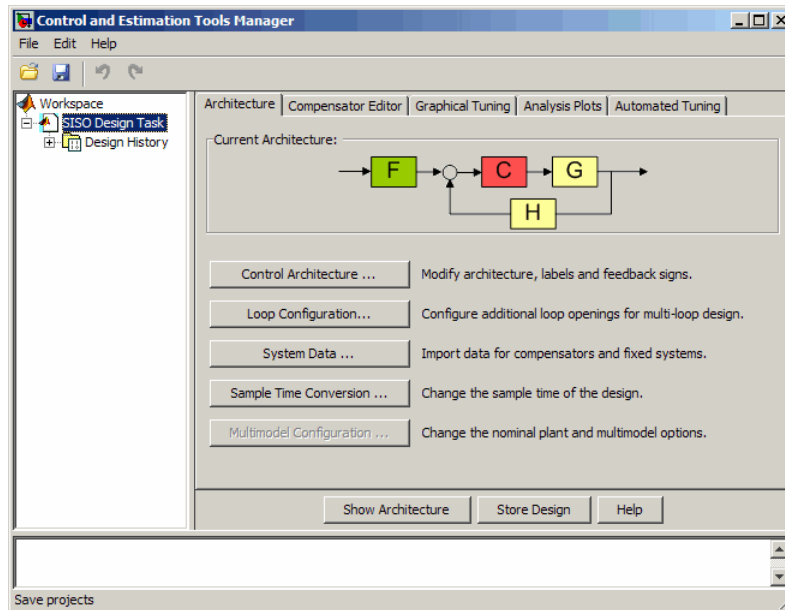
In this section...

“The SISO Design Task Node” on page 13-4

“SISO Design Task Node Menu Bar” on page 13-4

The SISO Design Task Node

The following figure shows the SISO Design Task node in the Control and Estimation Tools Manager.



SISO Design Task Node on the Control and Estimation Tools Manager

SISO Design Task Node Menu Bar

The SISO Design Task node menu bar contains the following menus:

File Edit Help

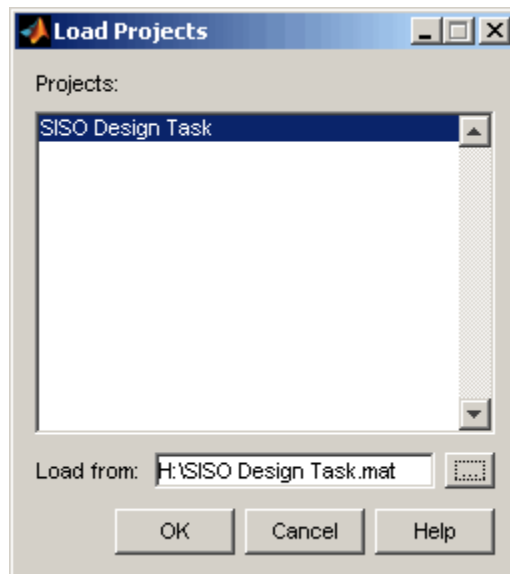
File Menu Options

Load
Save
Export
Close

- **Load**

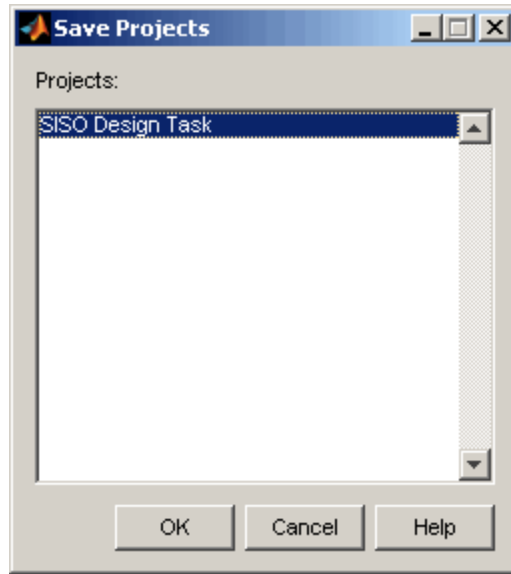
To load a saved SISO Design Tool project, select **Load** from the **File** menu. This opens the Load Projects window.

Projects are saved as MAT-files. Select the project you want to load from the list, or click ... to browse for projects you can select from, and click **OK**.



- **Save**

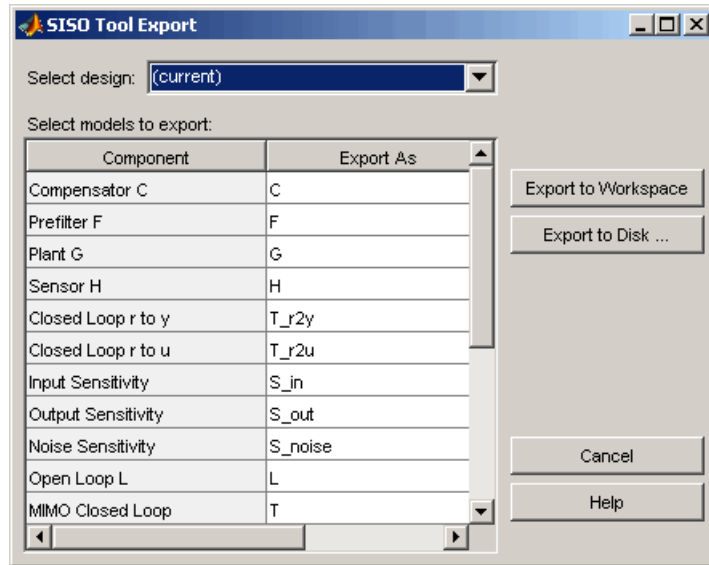
You can exit the MATLAB technical computing environment and later restore the SISO Design Tool to the state you left it in by saving the project. Select **Save** from the **File** menu. This opens the **Save Projects** window.



To save a project, specify a file name and click **OK**. The current state and configuration of your SISO Design Tool are saved as a MAT-file. To load a saved project, select **Load** from the **File** menu (see previous bullet).

- **Export**

Selecting **Export** from the **File** menu opens the SISO Tool Export window.



You can perform the following tasks in this window:

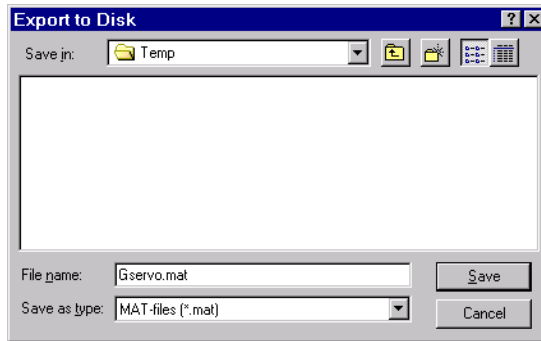
- Export models to the MATLAB workspace or to a disk. The exported models are:
 - LTI objects if the plant and sensor are LTI models
 - Arrays of LTI objects if the plant or sensor are arrays of LTI models
- Rename models when exporting
- Save variations on models, including open and closed loop models, sensitivity transfer functions, and state-space representations

To export models to the MATLAB workspace, follow these steps:

- 1** Select the model you want to export from the Component list by left-clicking the model name. To select more than one model, hold down the **Shift** key if they are adjacent on the list. If you want to save nonadjacent models, hold down the **Ctrl** key while selecting the models.
- 2** For each model you want to save, specify a name in the model's cell in the Export As list. A default name exists if you do not want to assign a new name.

3 Click **Export to Workspace**.

If you want to save your models as a MAT-file, follow steps 1 and 2 and click **Export to Disk**, which opens this window.



Choose where you want to save the file in the **Save in** field and specify the name of the MAT-file in the **File name** field. Click **Save** to save the file.

- **Close**

Use **Close** to close the SISO Design Tool. This closes all components of the SISO Design Tool.

Edit Menu Options

Undo
Redo
SISO Tool Preferences

- **Undo**

Use **Undo** to go back in design steps. Note that the **Undo** menu changes when the task you have just performed changes. For example, if you change the compensator gain, the **Undo** menu item now reads **Undo Edit Gain**.

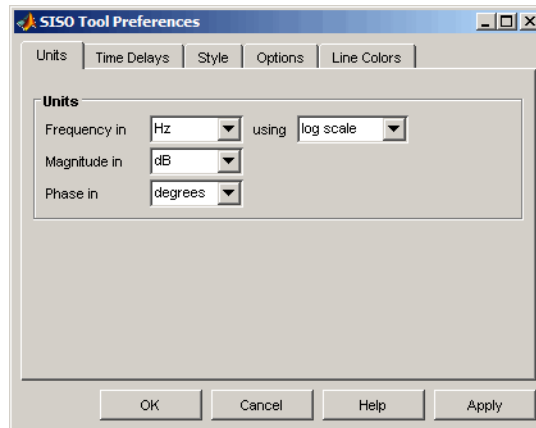
- **Redo**

Use **Redo** to go forward in the design steps. You can only use **Redo** if you have previously used **Undo**. Like the **Undo** menu, the **Redo** menu changes when the task you have just performed changes. For example, if

you change the compensator gain, and then select **Undo Edit Gain**, the Redo menu item becomes **Redo Edit Gain**.

- **SISO Tool Preferences**

Select **SISO Tool Preferences** from the **Edit** menu to open the **SISO Tool Preferences** dialog box.



You can use this window to do the following:

- Change units
- Add plot grids, change font styles for titles, labels, etc., and change axes foreground colors
- Change the compensator format
- Show or hide system poles and zeros in Bode diagrams

For a discussion of this window's features, see "Setting Toolbox Preferences" online in the Control System Toolbox documentation.

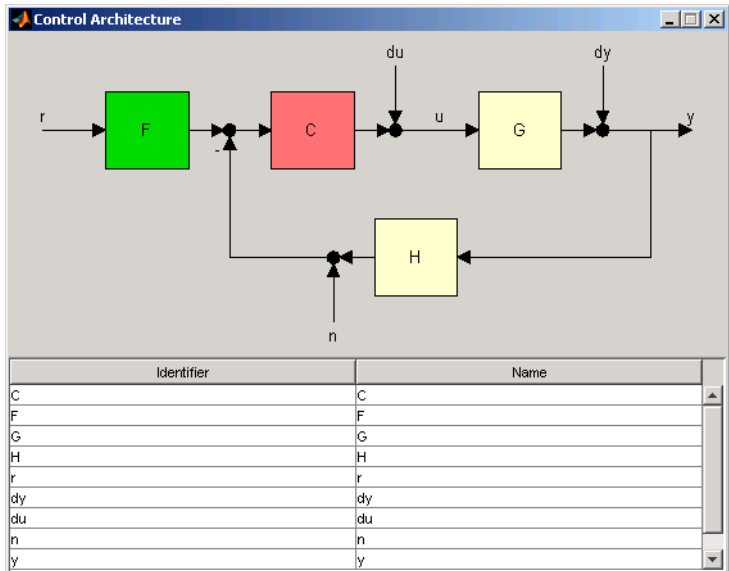
Help

Selecting **About the Control and Estimation Tools Manager** in the **Help** menu opens a window with the version number and a copyright notice for this product.

Buttons Available from Any Pane

- “Showing the Control Architecture” on page 13-10
- “Store Design” on page 13-10

Showing the Control Architecture. Click **Show Architecture** to open a window that displays the block diagram for your model. For example,



Below the block diagram is a table that shows the default names for each part of the block diagram and the assigned name, if you have one.

Store Design. Click **Store Design** to save your design to your SISO Design Task node. Click on **Design** under the node to see a snapshot summary of your design. Click on the **Design History** node to show a list of all stored designs.

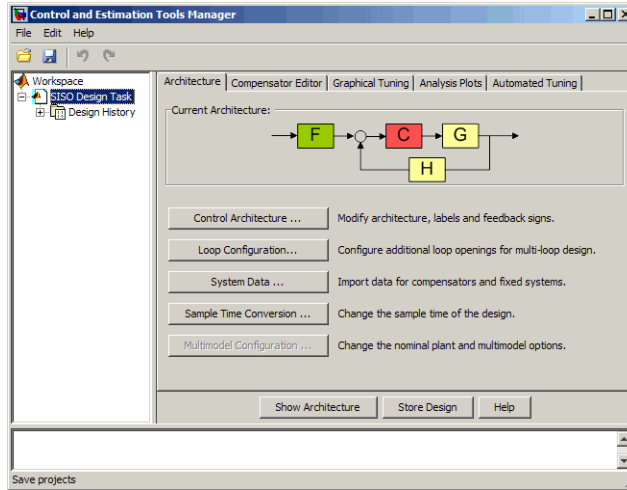
Using the SISO Design Task in the Controls & Estimation Tools Manager

| In this section... |
|------------------------------------|
| “Architecture” on page 13-11 |
| “Compensator Editor” on page 13-18 |
| “Graphical Tuning” on page 13-19 |
| “Analysis Plots” on page 13-23 |
| “Automated Tuning” on page 13-24 |

Architecture

Use the **Architecture** tab for:

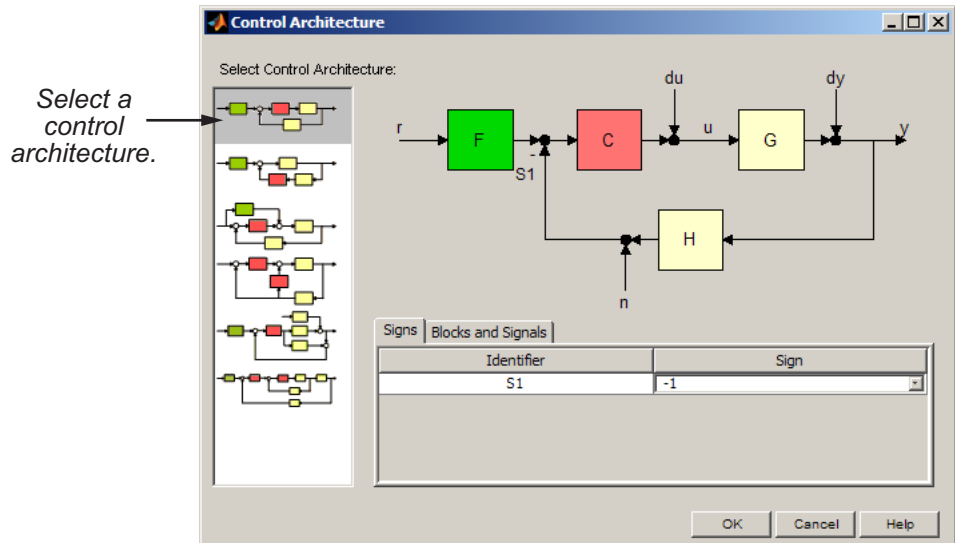
- “Block Diagram Structure Modifications” on page 13-12
- “Loop Configuration” on page 13-14
- “Model Import” on page 13-14
- “Sample Times for Continuous/Discrete Conversions” on page 13-16
- “Multimodel Configuration” on page 13-17



Architecture Pane on the SISO Design Task Node

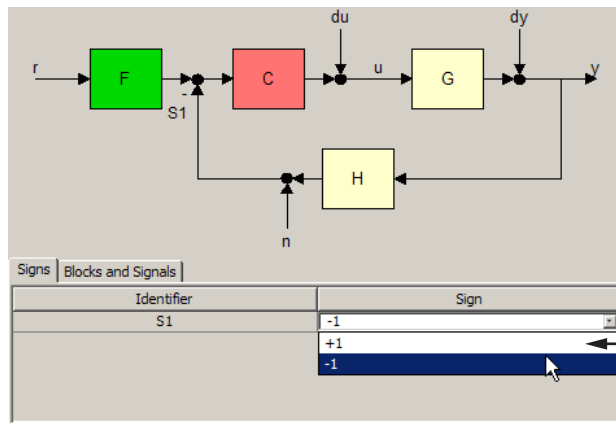
Block Diagram Structure Modifications

Click **Control Architecture** to change the feedback structure and label signals and blocks. The following pane appears:



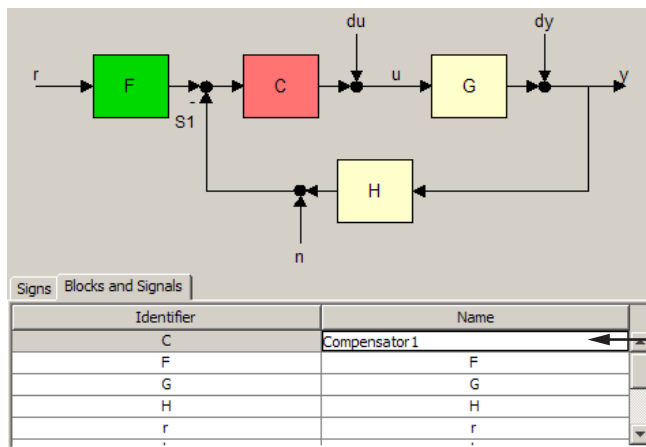
Select an architecture from the list of block configurations. These include compensator in the forward path, compensator in the feedback path, feedforward controller, and various multi-loop configurations. The window automatically updates to show the selected configuration.

Each configuration has associated Signs and Blocks and Signals panes. This figure shows the Signs pane.



Use menu to toggle between + and - for feedback signals at summing junction.

The Blocks and Signals pane displays the generic identifier, for example F for the prefilter block, and a default name.

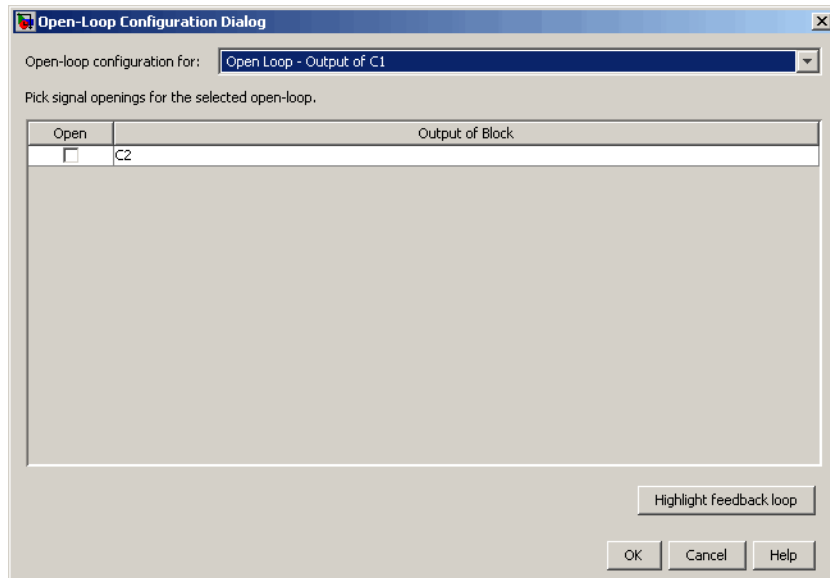


Double-click to change name.

On the Signs pane, use the menu to toggle between.

Loop Configuration

Click **Loop Configuration** to configure loops for multi-loop design by opening signals to remove the effects of other feedback loops.

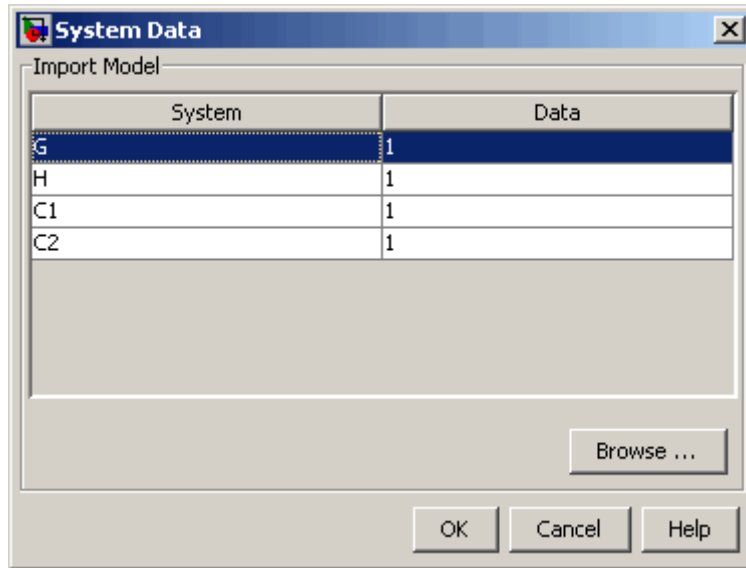


To specify openings for a given open loop, select the loop in the combo box. Click **Highlight Feedback Loop** to see the effects of the selected openings.

For an example of how to use this window in design, see Multi-Loop Compensator Design.

Model Import

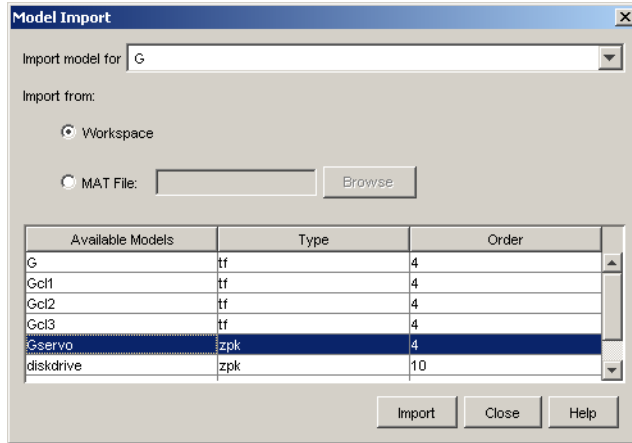
Click **System Data** on the Architecture pane to import models into your system. This opens the System Data dialog box, shown below.



You can import models for the plant (**G**), compensator (**C**), prefilter (**F**), and/or sensor (**H**). **G** or **H** or both are LTI models or row or column arrays of LTI models. If both **G** and **H** are arrays, their sizes must match.

To import a model:

- 1 Select a system in the **System** column and click **Browse**. The Model Import dialog box opens, as shown in the next figure.



2 Select a model from the **Available Models** list. You can import models from:

- The MATLAB workspace
- A MAT-file

3 Click **Import**, then click **Close**. You can now see the model loaded into the system selected in the **System Data** dialog.

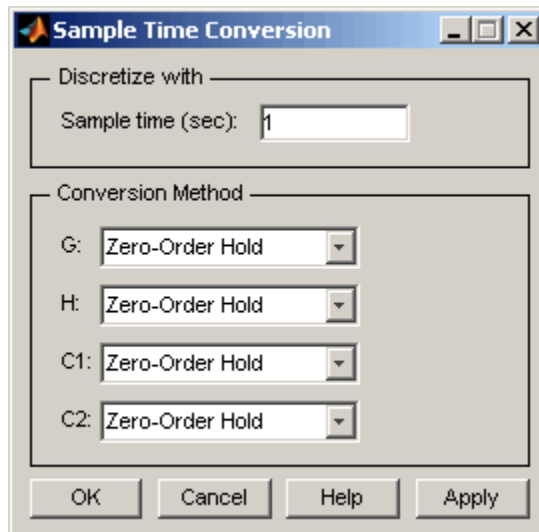
4 Click **OK**. The Graphical Tuning window is updated with the model you loaded.

Alternatively, you can import by entering a valid expression or variable (double, LTI object or row or column array of LTI objects) in the Data column in the System Data window.

For more information, see “Importing Models into the SISO Design Tool”.

Sample Times for Continuous/Discrete Conversions

Click **Sample Time Conversion** to convert the sample time of the system or switch between different sample times to design different compensators.



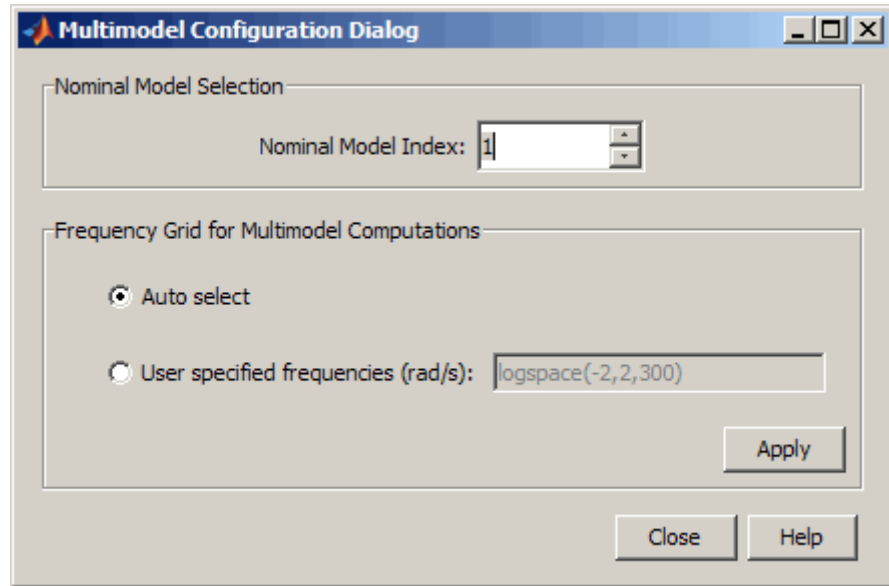
Choose from Zero-Order Hold, First-Order Hold, Impulse Variant, Tustin, Tustin w/Prewarping, and Matched Pole-Zero.

For a full description, see “Continuous/Discrete Conversions Using the Sample Time Conversion Dialog Box” on page 13-50.

Multimodel Configuration

The **Multimodel Configuration** button is enabled only when you import or open the SISO Design Tool GUI with a row or column arrays of LTI models for the plant **G** or sensor **H** or both. The LTI arrays model system variations in the plant and sensor. If both **G** and **H** are arrays, their sizes must match.

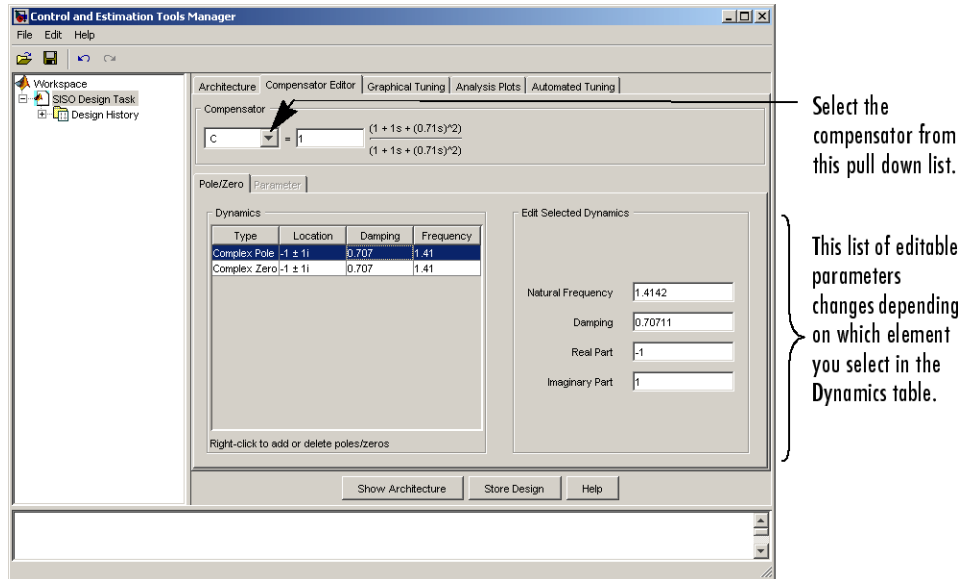
Click **Multimodel Configuration** to specify the nominal model and frequency grid for multimodel computations. This action opens the Multimodel Configuration Dialog window, as shown in the next figure.



For more information, see “Control Design Analysis of Multiple Models”.

Compensator Editor

Use the **Compensator Editor** for adding or editing gains, poles, and zeros.



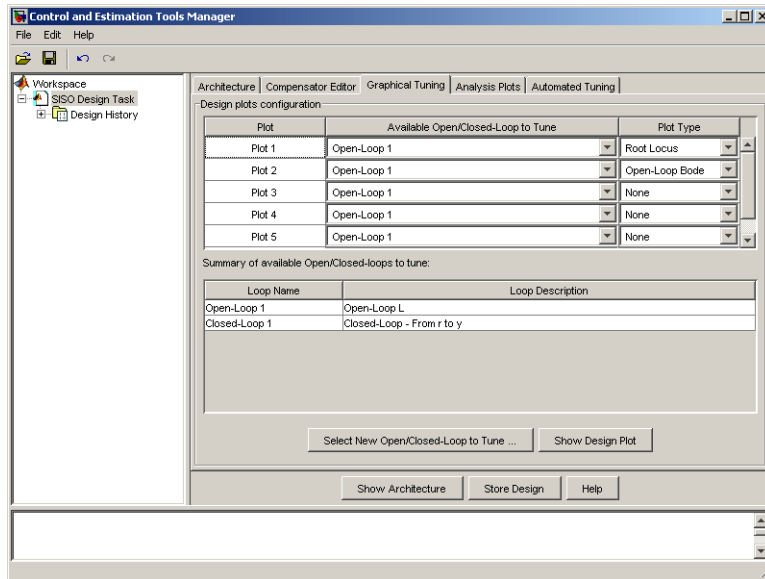
Compensator Editor Pane on the SISO Design Task Node

- 1 Enter the compensator gain in the text box in the top part of the pane.
- 2 Add or remove compensator poles and zeros by right-clicking in the **Dynamics** table.
- 3 Adjust pole and zero settings by entering values directly in the **Edit Selected Dynamics** group box.

Graphical Tuning

Use the Graphical Tuning pane for

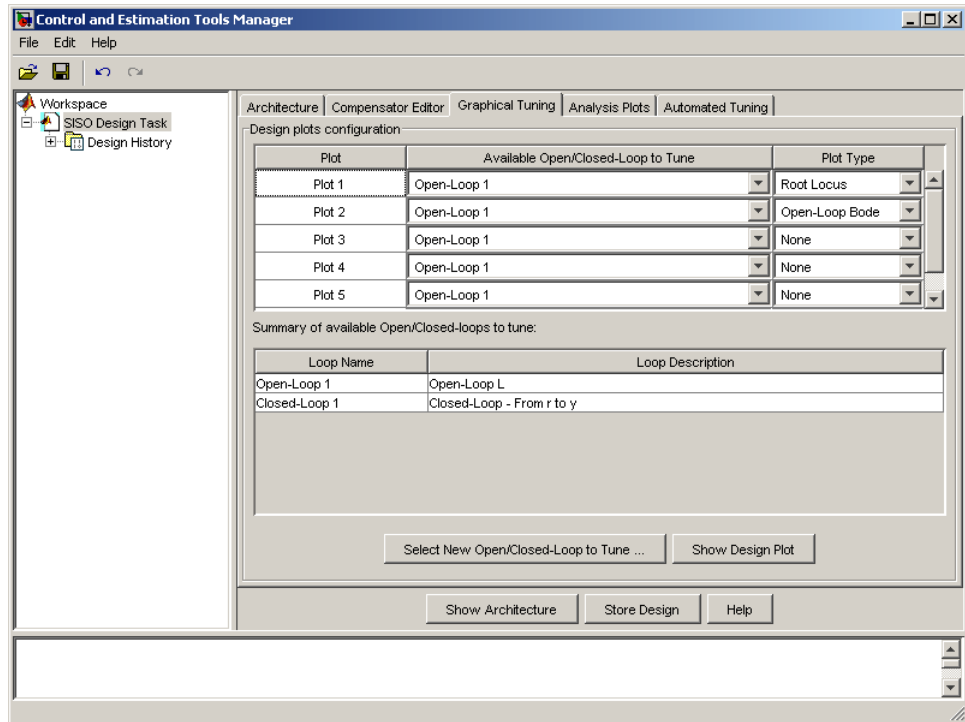
- “Configuring Design Plots for the Graphical Tuning Window” on page 13-20
- “Selecting New Loops to Tune” on page 13-22
- “Refocusing on the Graphical Tuning Window” on page 13-22



Graphical Tuning Pane on the SISO Design Task Node

Configuring Design Plots for the Graphical Tuning Window

Click the **Graphical Tuning** tab to configure design plots displayed in the Graphical Tuning Window.



In the Graphical Tuning window, use design plots to graphically manipulate system response. These design plots are dynamically linked to the SISO Design Task. When you change the dynamics of your compensator in either the SISO Design Task or the Graphical Tuning window, the design updates in both places.

For open-loop responses, the available plot types are:

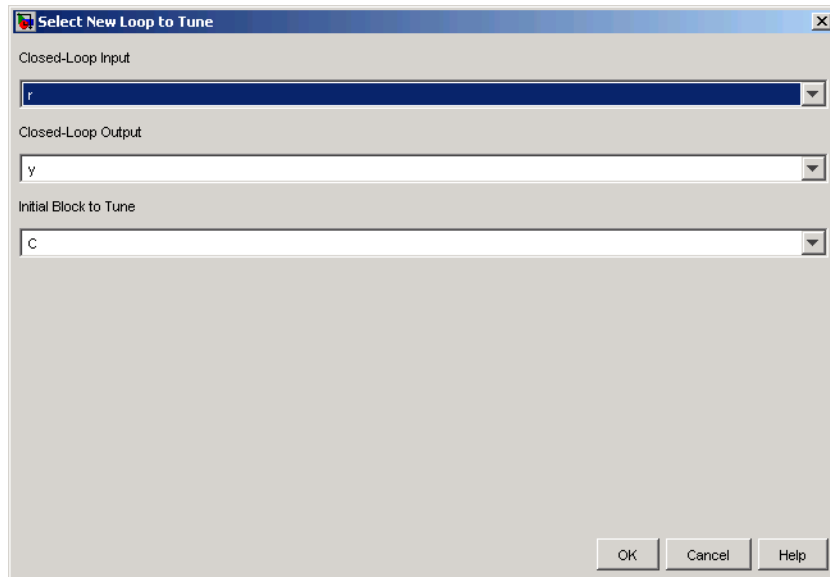
- Root locus
- Nichols
- Bode

For closed-loop responses, the available plot type is Bode.

For row or column arrays of LTI models, the design plots show the individual response of all models in the array by default. For more information, see “Using the Graphical Tuning Window” in the Getting Started Guide.

Selecting New Loops to Tune

Click **Select New Open/Closed Loops to Tune** to open a window for specifying new loops to tune.



Use the pull down menus to select the desired closed loop to tune by specifying the input, output, and blocks for tuning. Using the dialog box, you can select additional closed loops to tune.

Any loop you specify is displayed in the **Summary of Available Loops to Tune** in the Graphical Tuning pane. The list is also available in the **Design plots configuration** table of the same pane. You can use the latter for configuring design plots.

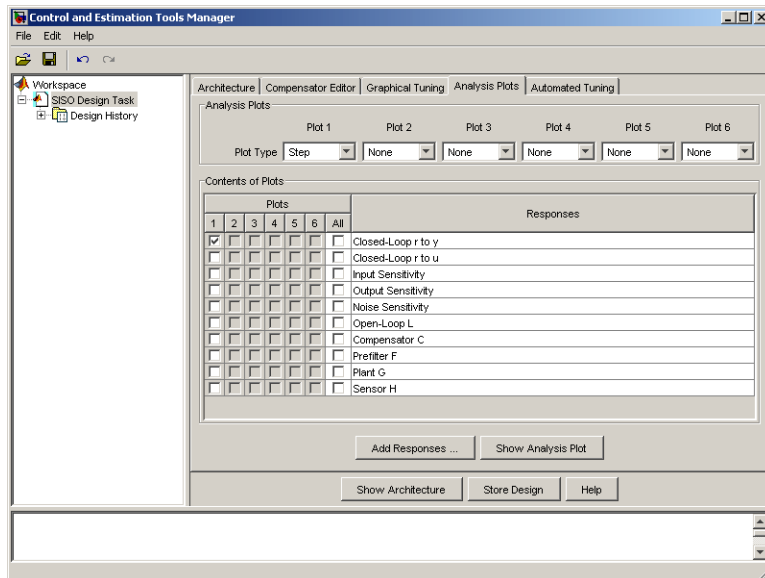
Refocusing on the Graphical Tuning Window

Click **Show Design Plot** to change the focus to the Graphical Tuning window.

Analysis Plots

Use the **Analysis Plots** pane for

- “Customizing Loop Responses” on page 13-23
- “Adding New Response Plots” on page 13-24
- “Opening or Changing the Focus to the LTI Viewer” on page 13-24



Analysis Plots Pane on the SISO Design Task Node

Customizing Loop Responses

The following sections describe the main components of the **Analysis Plots** pane.

Analysis Plots. You can have up to six plots in one LTI Viewer. To add a plot, start by selecting "Plot 1" from the list of plots. Then select a new plot type from the pull down menu. You can choose any of the plots available in the LTI Viewer. Select "None" to remove a plot.

Contents of plots. Once you have selected a plot type, you can include several open- and closed-loop transfer function responses for display. You can plot open-loop responses for each of the components of your system, including your compensator (**C**), plant (**G**), prefilter (**F**), or sensor (**H**). In addition, various closed loop and sensitivity response plots are available.

For row or column arrays of LTI models, the analysis plots show the response of the nominal model only by default. For more information, see “Analysis Plots for Loop Responses” in the Getting Started Guide.

Adding New Response Plots

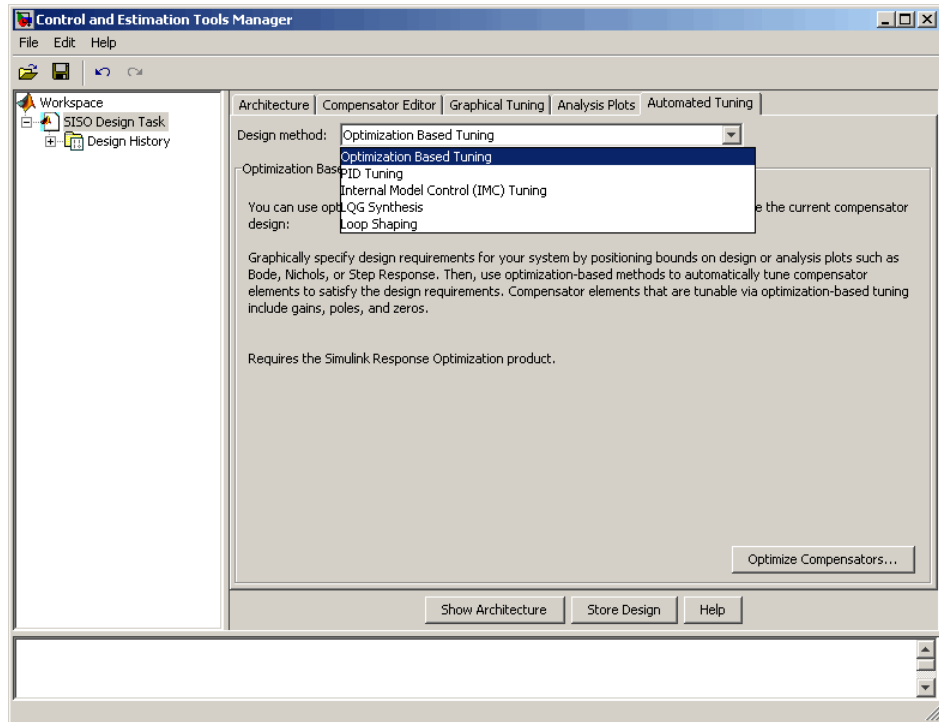
Click **Add Responses** to open a window with three drop-down menus for selecting open and closed loop responses for various input and output nodes in the control architecture block diagram. This allows you to select additional responses for viewing. The **Response** table updates automatically to include the selected response.

Opening or Changing the Focus to the LTI Viewer

Click **Show Analysis Plot** to open a new LTI Viewer for SISO Design with the response plots that you selected. All the plots open in one instance of the LTI Viewer.

Automated Tuning

Use the **Automated Tuning** pane to select a method for automatic tuning of your compensator design. Automated tuning methods help you design an initial compensator for a SISO loop that satisfies your design specifications.



You can choose among the following design methods:

- “Optimization-Based Tuning” on page 13-27 — Optimize compensator parameters using design requirements implemented in graphical tuning and analysis plots
- “PID Tuning” on page 13-28 — Tune PID controller parameters using the Robust response time tuning algorithm or classic tuning formulas
- “Internal Model Control (IMC) Tuning” on page 13-36 — Obtain a full-order stabilizing feedback controller using the IMC design method
- “LQG Synthesis” on page 13-37 — Design a full-order stabilizing feedback controller as a Linear-Quadratic-Gaussian (LQG) tracker
- “Loop Shaping” on page 13-39 — Find a full-order stabilizing feedback controller with a desired open loop bandwidth or shape

After you select a design method, the pane updates to display the corresponding options.

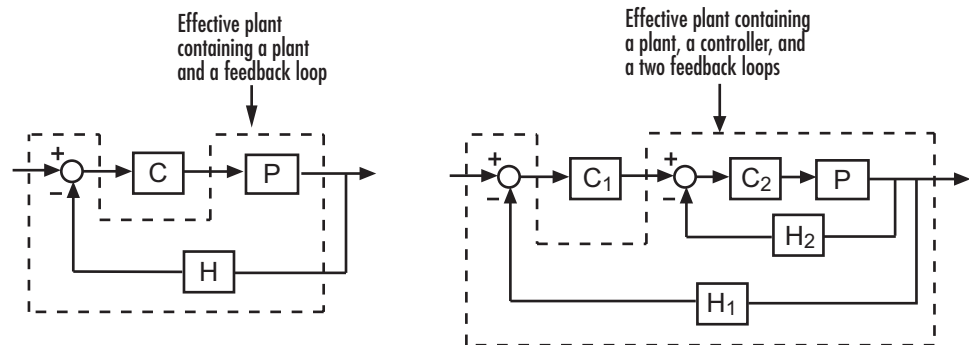
Note If the particular design method you are using does not apply or fails, try selecting different tuning specifications or switch to a different design method.

Stability of an Effective Plant for Automated Tuning

Knowing the stability of the effective plant in your model may help you understand which automated tuning methods work for your model. Some of the automated tuning methods only apply to compensators whose open loops

($L = C\hat{P}$) have stable effective plants (\hat{P}).

An *effective plant* is the system controlled by the compensator you design and contains all elements of the open loop in your model other than this compensator. The following figure shows two examples of effective plants.



Generic Work Flow

For each method, follow these steps to do your design:

- 1 Select an automated tuning algorithm from the **Design method** drop-down menu.
- 2 If you select Optimization-Based Tuning, stop here and see “Optimization-Based Tuning” on page 13-27.

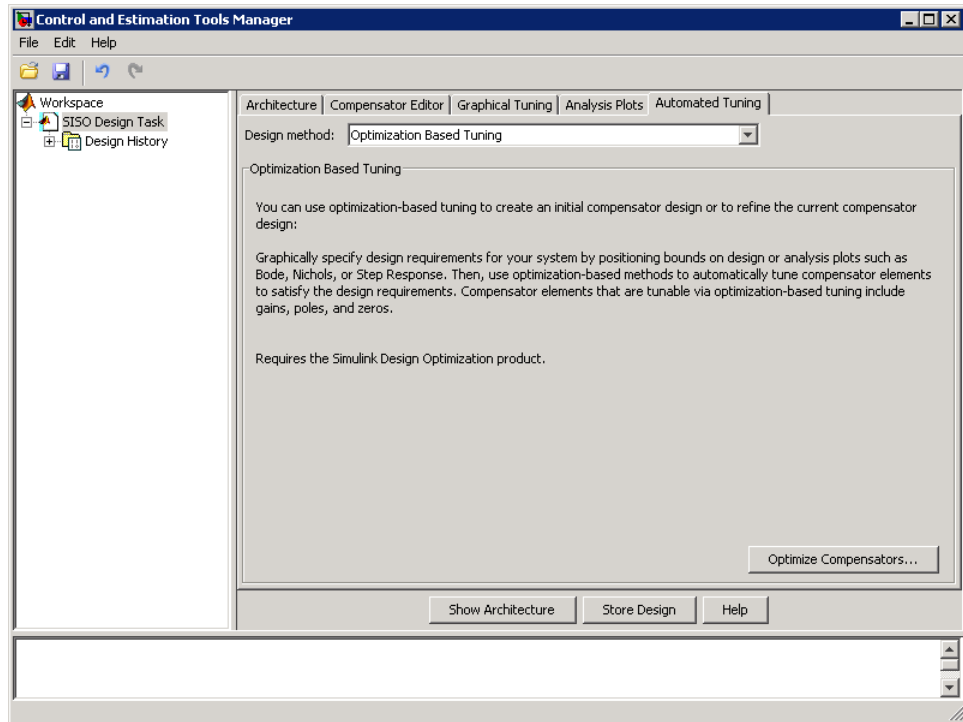
- 3** Select a compensator from the drop-down menu.
- 4** Determine how you want the compensator to perform and set the tuning specifications.
- 5** Click **Update Compensator** and notice the changes in the associated design and analysis plots.

Note If you encounter a disabled **Update Compensator** button, try selecting different tuning specifications (Step 4) or switch to a different tuning algorithm (Step 1). The disabled button means that the current method does not work for your model.

Optimization-Based Tuning

Optimization-based tuning is available only if you have Simulink Design Optimization software installed. You can use this method to either:

- Directly tune response signals within Simulink models.
- Tune responses of LTI systems using a SISO Design Task.

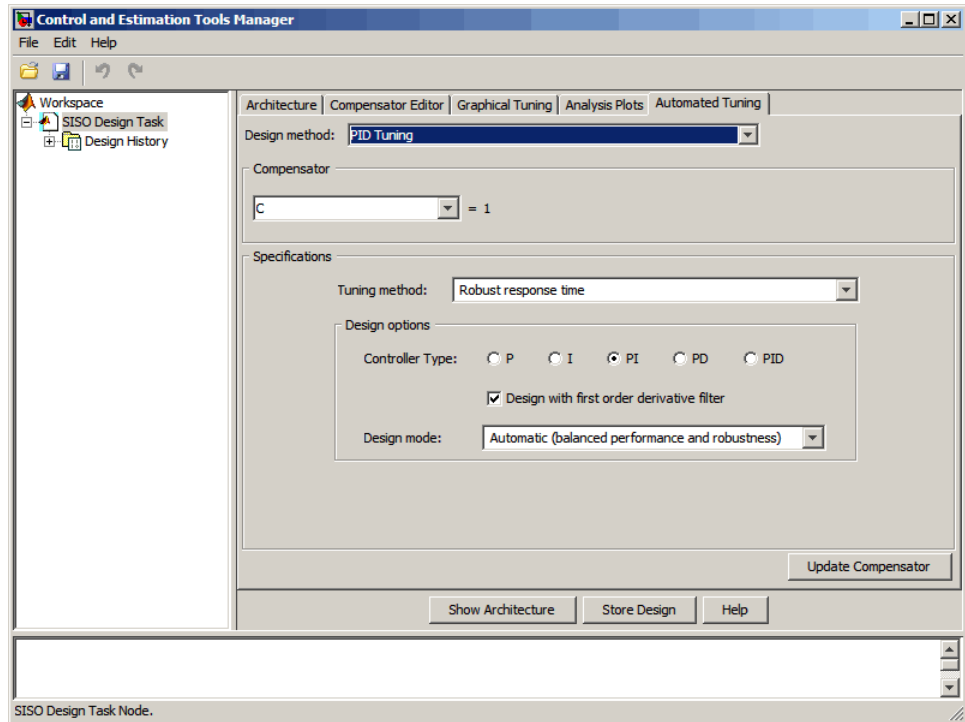


See “Frequency Domain Response Optimization Example” in the Simulink Design Optimization documentation for more details.

PID Tuning

PID (proportional-integral-derivative) control is the most popular control technique used in modern industry.

To use automatic PID tuning, select PID Tuning from the Automated Tuning pane of the Control and Estimation Tools Manager. In most cases, the PID controllers resulting from PID tuning provide acceptable performance. Use the “Analysis Plots” on page 13-23 to verify design results.



Types of PID Controllers.

SISO Design Tool provides automated tuning for the following PID controller types.

- P — Proportional-only control
- I — Integral-only control (available only for the Robust response time tuning method)
- PI — Proportional-integral control
- PD — Proportional-derivative control (available only for the Robust response time tuning method)
- PDF — Proportional-derivative control with a low-pass filter on the derivative term (available only for the Robust response time tuning method)
- PID — Proportional-integral-derivative control

- PIDF — Proportional-integral-derivative control with a low-pass filter on the derivative term

PID Tuning Methods.

SISO Design Tool provides a Robust response time algorithm for interactive tuning, as well as six well-known classical tuning methods.

Robust response time. This method computes PID parameters to robustly stabilize your system based on the bandwidth and phase margin that you specify. Using the robust response time method you can:

- Tune interactively, adjusting bandwidth and phase margin to achieve your desired balance between performance and robustness
- Tune any type of PID controller (P, I, PI, PD, PDF, PID, or PIDF)
- Tune all PID parameters, including the derivative filter
- Design for plants that are stable, unstable, or integrating

Classical design formulas. SISO Design Tool includes the following well-known PID design formulas:

- Approximate MIGO frequency response — Closed-loop frequency-domain approximate M-constrained integral gain approximation (see [1], Section 7.5).
- Approximate MIGO step response — Open-loop time-domain approximate M-constrained integral gain approximation (see [1], Sections 7.3–7.4).
- Chien-Hrones-Reswick — Approximates the plant as a first-order model with a time delay and computes PID parameters using a Chien-Hrones-Reswick look-up table for zero overshoot and disturbance rejection (see [1], Section 6.2).
- Skogestad IMC — Approximates the plant as a first-order model with a time delay and computes PID parameters using Skogestad design rules (see [2]).

(This method is different from selecting “Internal Model Control (IMC) Tuning” on page 13-36 as the full-order compensator tuning method).

- Ziegler-Nichols frequency response — Computes controller parameters from a Ziegler-Nichols lookup table, based on the ultimate gain and frequency of the system (see [1], Section 6.2).
- Ziegler-Nichols step response — Approximates the plant as a first-order model with a time delay that and computes PID parameters using the Ziegler-Nichols design method (see [1], Section 6.2).

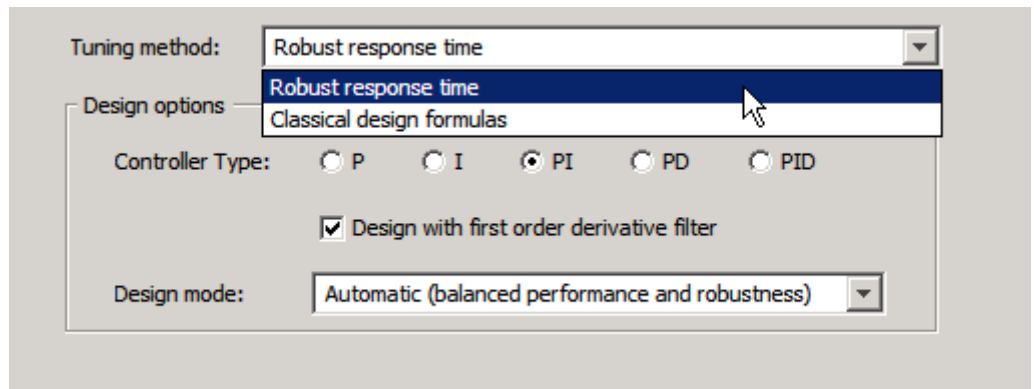
The classical design formulas:

- Require a stable or integrating plant.
- Can design for P, PI, PID, or PID with derivative filter.
- Cannot tune the derivative filter. If you select PID with derivative filter, classical design formulas set the filter time constant to $Td/10$, where Td is the tuned derivative time.

Automated Tuning using the Robust Response Time Method.

To use the robust response time tuning method:

- 1 Select Robust response time from the **Tuning method** menu.



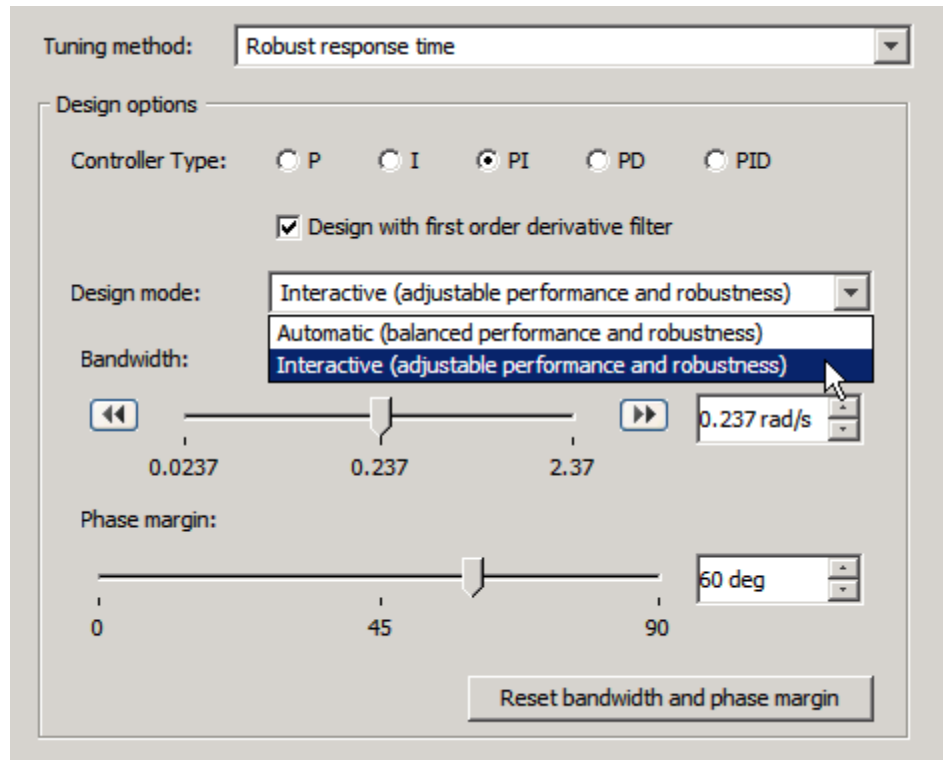
- 2 Choose a controller type by clicking the corresponding radio button. To include a first-order filter on the derivative action for PD or PID controller type, check the **Design with first order derivative filter** checkbox.

Note If you are tuning a PID Controller block in a Simulink model and your block is type P, I, or PI, select the same controller type as the block. If your block is type PD or PID, select the corresponding button and check the **Design with first order derivative filter** checkbox.

- 3** Click **Update Compensator** to design a controller of the selected type.

By default, the SISO Design Tool automatically computes controller parameters for balanced performance and robustness.

- 4** Analyze the response using the analysis plots you select on the Analysis Plots pane.
- 5** To design a controller interactively, select **Interactive** (adjustable performance and robustness) from the **Design mode** menu. This selection activates the **Bandwidth** and **Phase Margin** sliders.



Adjust the bandwidth and phase margin to achieve your desired controller performance. For example:

- Increase the bandwidth for a more aggressive controller.
- Increase the phase margin to reduce overshoot.

You can adjust the bandwidth and phase margin values by:

- Moving the sliders
- Entering values in the text field
- Incrementally adjusting the values in the text field using the up and down arrows.

To increase or decrease the bandwidth by a factor of 10, click the right or left double arrows, respectively.

Note Click the **Reset bandwidth and phase margin** button at any time to return the sliders to the bandwidth and phase margin of the default compensator design for your plant.

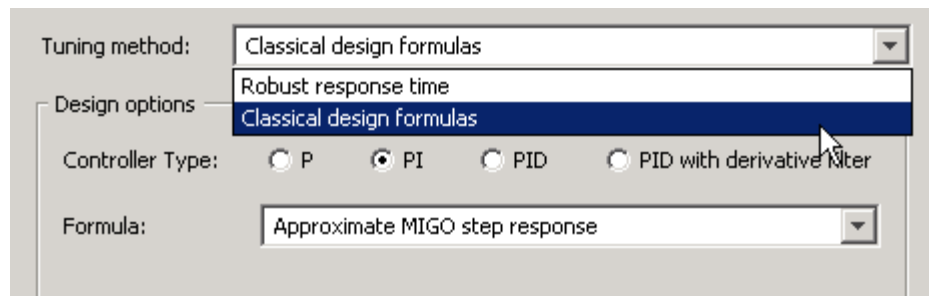
- 6 Click **Update Compensator** again. SISO Design Tool computes new controller parameters for the specified target bandwidth and phase margin. Repeat steps 4-6 as necessary to achieve your desired performance.

Note SISO Design Tool displays the tuned compensator in zpk form in the **Compensator** section of the Automated Tuning Pane. To convert the compensator to parallel or standard PID form, export the compensator to the MATLAB workspace, as described in “SISO Design Task Node Menu Bar” on page 13-4. Then use the `pid` or `pidstd` commands to convert the exported compensator to parallel or standard PID parameters, respectively.

Automated Tuning using Classical Design Formulas.

To use one of the classical design formulas:

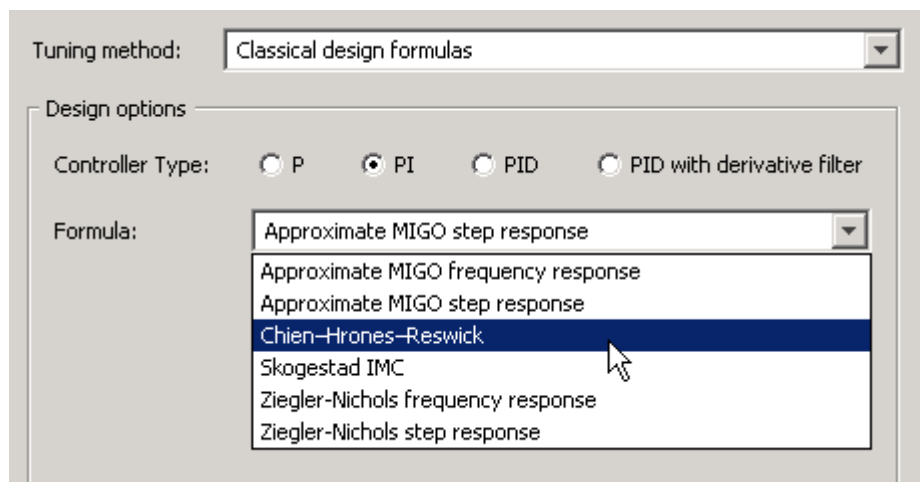
- 1 Select Classical design formulas from the **Tuning method** menu.



- 2 Select a controller type (P, PI, PID, or PID with derivative filter) by clicking the corresponding radio button.

Note If you are tuning a PID Controller block in a Simulink model and your block is type P or PI, select the same controller type as the block. If your block is type PID, select **PID with derivative filter**. If your block is type PD or I, use the Robust response time tuning method.

- 3 Select the classical design formula you want to use from the **Formula** menu.



- 4 Click the **Update Compensator** button to design a controller using the selected formula.

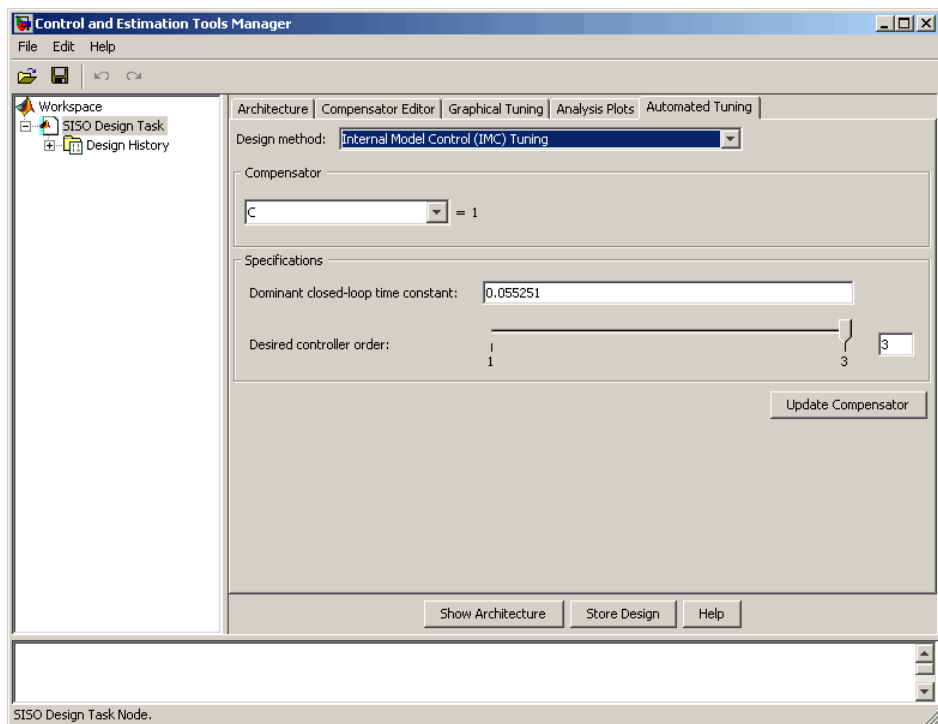
Note SISO Design Tool displays the tuned compensator in zpk form in the **Compensator** section of the Automated Tuning Pane. To convert the compensator to parallel or standard PID form, export the compensator to the MATLAB workspace, as described in “SISO Design Task Node Menu Bar” on page 13-4. Then use the `pid` or `pidstd` commands to convert the exported compensator to parallel or standard PID parameters, respectively.

References.

- [1] Åström, K. J. and Hägglund, T. *Advanced PID Control*, Research Triangle Park, NC: Instrumentation, Systems, and Automation Society, 2006.
- [2] Skogestad, S., “Simple analytic rules for model reduction and PID controller tuning.” *Journal of Process Control*, Vol. 13, No. 4, 2003, pp. 291–309.

Internal Model Control (IMC) Tuning

IMC design generates a full-order feedback controller that guarantees closed-loop stability when there is no model error. It also contains an integrator, which guarantees zero steady-state offset for plants without a free differentiator. You can use this tuning method for both stable and unstable plants.

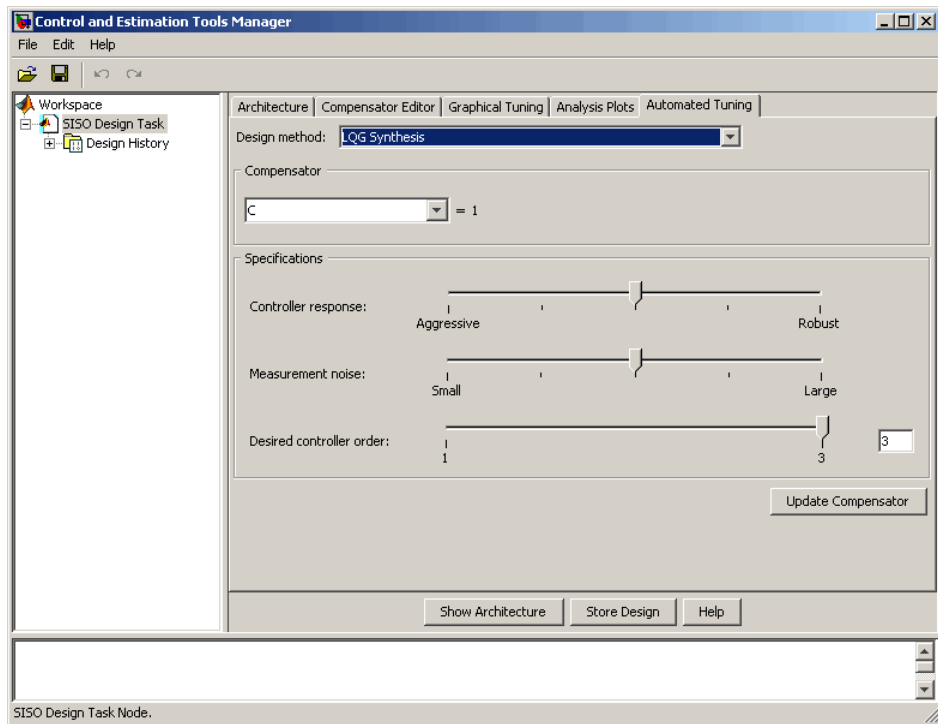


To design an IMC controller:

- 1** Specify a value in the **Dominant closed-loop time constant** field.
The initial value is set as 5% of the open-loop settling time. In general, increasing this value slows down the closed system and makes it more robust.
- 2** Specify a value in the **Desired controller order** field using the slider.
After you obtain a full-order feedback controller, you can try to reduce its order. You may lose performance and closed-loop stability if you reduce the order.
- 3** Click **Update Compensator**.

LQG Synthesis

LQG tracker design generates a full-order feedback controller that guarantees closed-loop stability. It also contains an integrator, which guarantees zero steady-state error for plants without a free differentiator.



To design an LQG controller:

1 Specify your preference for controller response using the **Controller response** slider.

- Move the slider to the left for aggressive control response.

This means that large overshoot is more heavily penalized so that the controller acts more aggressively. If you believe your model is accurate and that the manipulated variable has a large enough range, an aggressive controller is more desirable.

- Move the slider to the right for robust control response.

2 Specify your estimation of the level of measurement noise using the **Measurement noise** slider.

- Move the slider to the left for small measurement noise.

This means that you expect low noise from the process output measurement. Because this measurement is used by the Kalman estimator, process disturbances are picked up more accurately by the estimated states. In this case, the controller is freer from robustness considerations.

- Move the slider to the right for large measurement noise. This results in a controller that is more robust to measurement noise.

3 Specify your preference for controller order using the **Desired controller order** slider.

4 Click **Update Compensator**.

Loop Shaping

Loop shaping generates a stabilizing feedback controller to match as closely as possible to a desired loop shape. You can specify this loop shape as a bandwidth or an open loop frequency response. If you have Robust Control Toolbox software installed, you can use loop shaping for SISO systems. For more information see the section on H-Infinity Loop Shaping in the *Robust Control Toolbox User's Guide*.

To design a controller using loop shaping:

1 Select a tuning preference by clicking one of these option buttons:

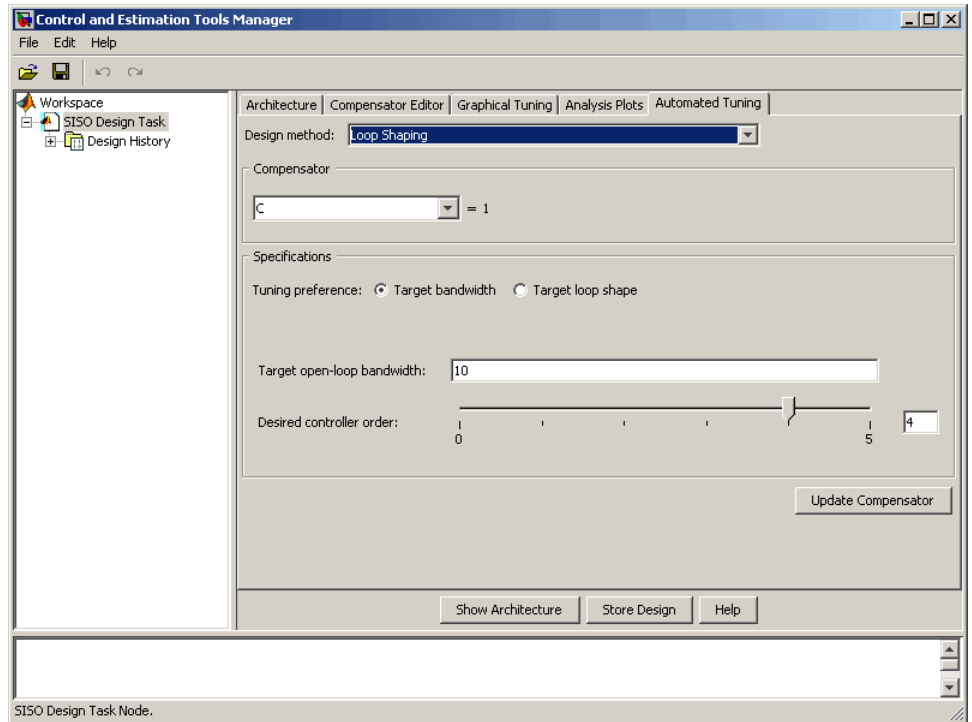
- **Target bandwidth** — Allows you to specify a target loop shape bandwidth (ω_b). This results in a loop shape of your specified bandwidth

over an integrator ($\frac{\omega_b}{s}$).

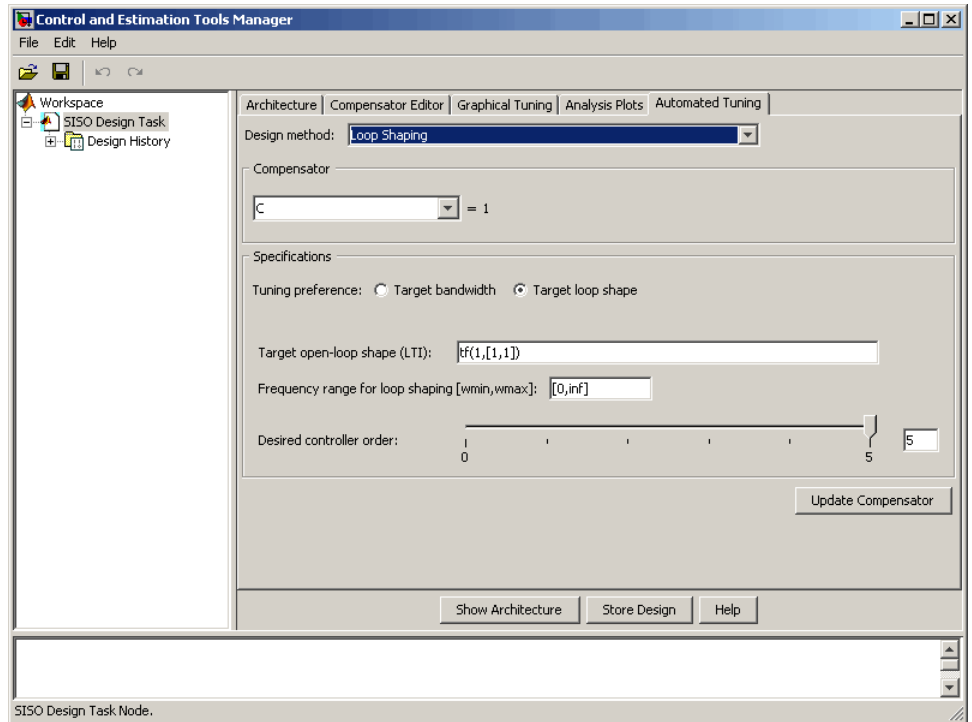
- **Target loop shape** — Allows you to specify the target open loop shape in one of the following representations: state-space, zero-pole-gain, or transfer functions.

2 Set the tuning options available for your selected tuning preference as follows:

- If you chose **Target bandwidth**, specify the desired **Target open-loop bandwidth** in the editable box.



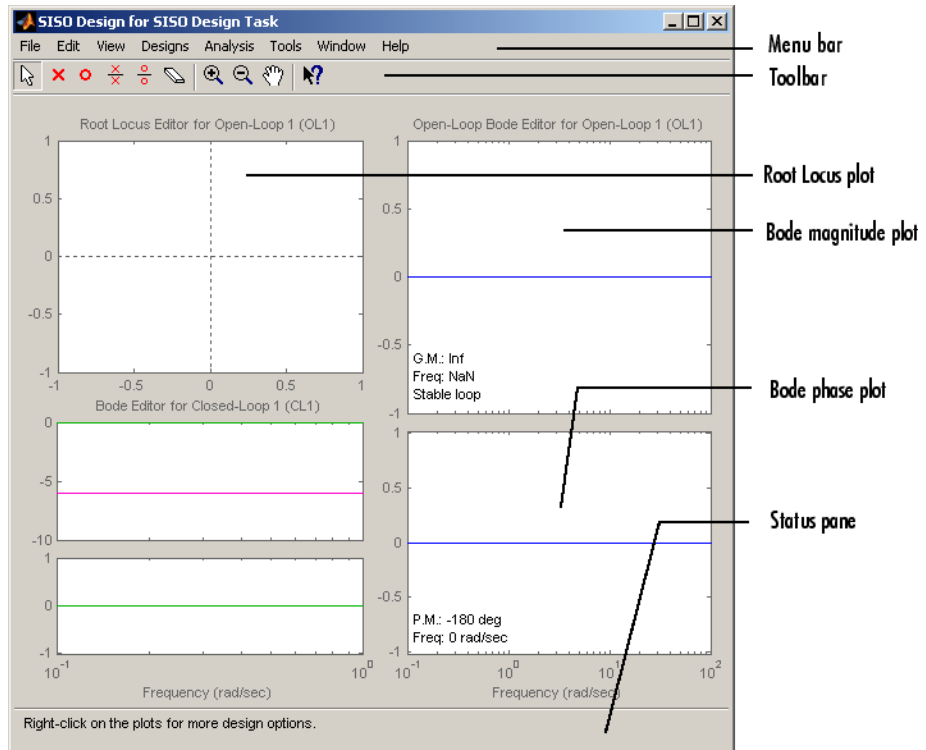
- If you chose **Target loop shape**, do the following:
 - Enter the desired **Target open-loop shape (LTI)**.
This can be a state-space representation, a zero-pole-gain representation, or a transfer function.
 - Enter the desired **Frequency range for loop shaping [wmin,wmax]**.



- 3** Specify your preference for controller order using the **Desired controller order** slider.
- 4** Click **Update Compensator**.

SISO Design Task Graphical Tuning Window

The following figure shows the Graphical Tuning window and introduces some terminology.



Graphical Tuning Window

Bode and Nichols plots in the graphical tuning window automatically display the following information:

- Gain margin and the -180 degree phase crossing frequency where it is measured
- Phase margin and the 0 dB gain crossing frequency where it is measured
- Whether the characteristic equation $1+L$ is stable (**Stable loop**) or unstable (**Unstable loop**). L is the open loop plotted in the figure.

For row or column arrays of LTI models, the plots display the characteristics of the nominal model only.

The following topics describe the Graphical Tuning window features:

- “Using the Graphical Tuning Window Menu Bar” on page 13-44
- “Using the Graphical Tuning Window Toolbar” on page 13-56
- “Using the Right-Click Menus in the Graphical Tuning Window” on page 13-57

Using the Graphical Tuning Window Menu Bar

In this section...

“Overview of the Graphical Tuning Window Menu Bar” on page 13-44

“File” on page 13-44

“Edit” on page 13-47

“View” on page 13-48

“Analysis” on page 13-49

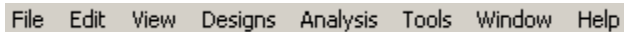
“Tools” on page 13-50

“Window” on page 13-54

“Help” on page 13-54

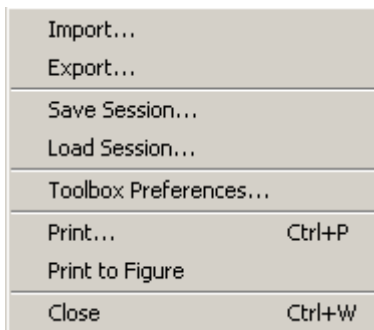
Overview of the Graphical Tuning Window Menu Bar

Several of the tasks you can do in the SISO Design Tool can be done from the menu bar, shown below.



File Edit View Designs Analysis Tools Window Help

File



Using the **File** menu, you can:

- Import and export models

- Save and reload sessions
- Set toolbox preferences
- Print and print to figure
- Close the Graphical Tuning Window

The following sections describe the **File** menu options in turn.

Import

Selecting **Import** opens the same System Data dialog box that clicking **System Data** on the **Architecture** pane does. See “Model Import” on page 13-14.

Export

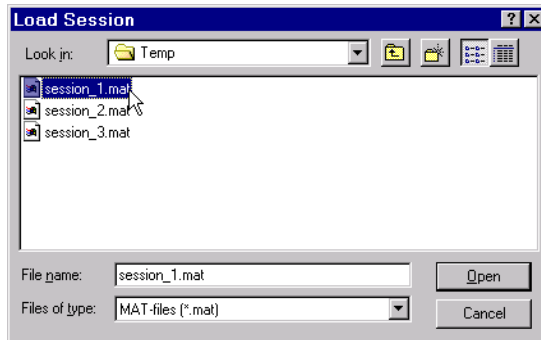
Selecting **Export** from the Graphical Tuning window **File** menu opens the same SISO Tool Export window that selecting **Export** from the SISO Design Task node **File** menu does. See Export.

Save Session

Selecting **Save Session** from the Graphical Tuning window **File** menu opens the same Save Projects window that selecting **Save** from the SISO Design Task node **File** menu does. See Save.

Load Session

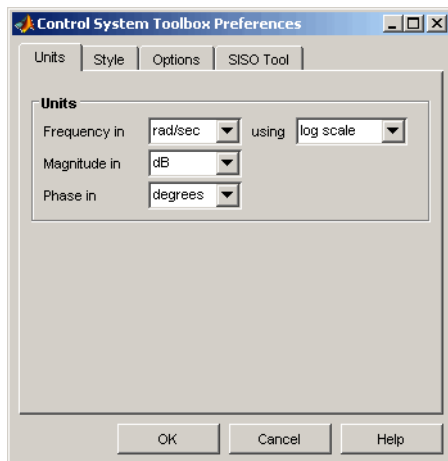
To load a saved SISO Design Tool session, select **Load Session** from the **File** menu. This opens the **Load Session** dialog.



Sessions are saved as MAT-files. Select the session you want to load from the list, and click **Open**. See “Save Session” on page 13-45 for information on saving **SISO Design Tool** sessions.

Toolbox Preferences

Select **Toolbox Preferences** from the **File** menu to open the **Control System Toolbox Preferences** window.



The Control System Toolbox Preferences Window

For a discussion of this window's features, see "Setting Toolbox Preferences" online in the Control System Toolbox documentation.

Print

Use **Print** to send a picture of the Graphical Tuning window to your printer.

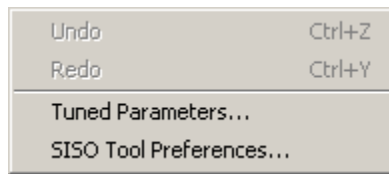
Print to Figure

Print to Figure opens a separate figure window containing the design views in your current Graphical Tuning window.

Close

Use **Close** to close the Graphical Tuning window.

Edit



Undo and Redo

Selecting **Undo** and **Redo** perform the same actions as selecting **Undo** and **Redo** from the **SISO Design Task Node Edit** menu. See “**Edit Menu Options**” on page 13-8.

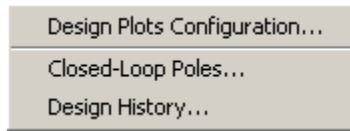
Tuned Parameters

Selecting **Tuned Parameters** opens the **SISO Tool Preferences** dialog box on the **Options** page.

SISO Tool Preferences

Selecting the **SISO Tool Preferences** option opens the same dialog box that selecting **SISO Tool Preferences** from the **Edit** menu on the **SISO Design Task Node**. See “**Edit Menu Options**” on page 13-8.

View

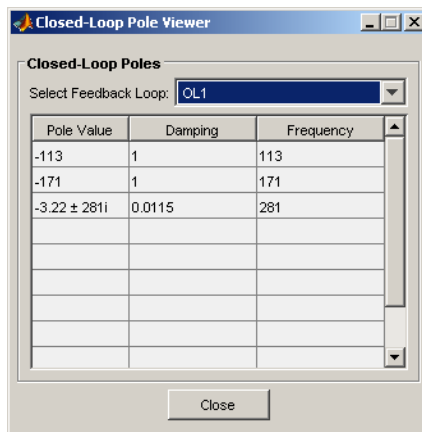


Design Plots Configuration

Select **Design Plots Configuration** to open the Graphical Tuning pane. See “Graphical Tuning” on page 13-19.

Closed-Loop Poles

Select **Closed-Loop Poles** from **View** to open the **Closed-Loop Pole Viewer**.

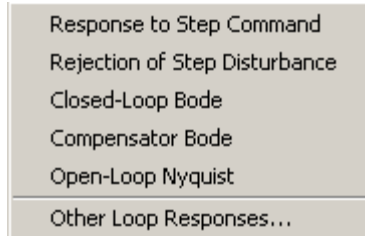


This window displays all the closed-loop pole values of the selected feedback loop and the associated damping and frequency values.

Design History

Select **Design History** from the **View** menu to open the **Design History** window, which displays all the actions you’ve performed during a design session. You can save the history to an ASCII flat text file by clicking **Save to Text File**.

Analysis



Common Response Plots

Each of the top group of items in the Analysis menu opens an LTI Viewer that is dynamically linked to your SISO Design Tool. You have the following response plot choices:

- **Response to Step Command** — The closed-loop step response of your system
- **Rejection of Step Disturbance** — The open-loop step response of your system
- **Closed-Loop Bode** — The closed-loop Bode diagram for your system
- **Compensator Bode** — The open-loop Bode diagram for your compensator
- **Open-Loop Nyquist** — The open-loop Nyquist plot for your system

When you make changes to the design via the Graphical Tuning window, the Compensator Editor pane, or the Automated Tuning pane, the response plots in the LTI Viewer automatically change to reflect the new design's responses.

Other Loop Responses

If you choose **Other Loop Responses**, the **Analysis Plots** pane opens. See "Analysis Plots" on page 13-23.

Tools

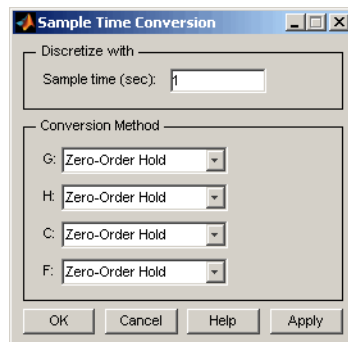
Continuous/Discrete Conversions...
 Draw Simulink Diagram...
 Automated Tuning...

Continuous/Discrete Conversions Using the Sample Time Conversion Dialog Box

- “Converting Continuous-Time Models to Discrete-Time Models” on page 13-50
- “Converting Discrete-Time Models to Continuous-Time” on page 13-51
- “Changing the Sample Time of a Discrete-Time Model” on page 13-52

Converting Continuous-Time Models to Discrete-Time Models. To convert a continuous-time model to a discrete-time model, perform these steps:

- 1 In the SISO Design Tool, select **Tools > Continuous/Discrete Conversions** to open the **Sample Time Conversion** window.



- 2 Specify a positive number for the sample time in the **Sample time (sec)** field.
- 3 Select a continuous-to-discrete conversion method for each component of your model. The components include the plant (**G**), the compensator (**C**), the prefilter (**F**), or the sensor (**H**). You can choose from the following conversion methods:

- Zero-order hold
- First-order hold
- Tustin
- Tustin with prewarping

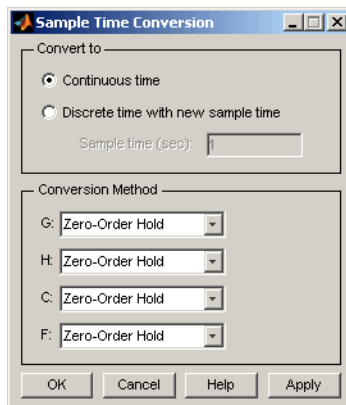
Note If you choose Tustin with prewarping, you must specify the critical frequency in radians per second.

- Matched pole/zero

For more information on each of these conversion methods, see “Supported Conversion Functions and Methods” on page 5-24 in the Control System Toolbox documentation.

Converting Discrete-Time Models to Continuous-Time. To convert a discrete-time model to a continuous-time model, perform these steps:

- 1 In the SISO Design Tool, select **Tools > Continuous/Discrete Conversions** to open the **Sample Time Conversion** window.



- 2 Select a discrete-to-continuous conversion method for each component of your model. The components include the plant (**G**), the compensator (**C**),

the prefilter (**F**), or the sensor (**H**). You can choose from the following conversion methods:

- Zero-order hold
- First-order hold
- Tustin
- Tustin with prewarping

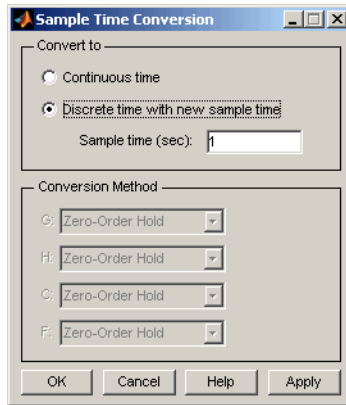
Note If you choose Tustin with prewarping, you must specify the critical frequency in radians per second.

- Matched pole/zero

For more information on each of these conversion methods, see “Supported Conversion Functions and Methods” on page 5-24 in the Control System Toolbox documentation.

Changing the Sample Time of a Discrete-Time Model. To change the sample time of (resample) a discrete system, perform these steps:

- 1** In the SISO Design Tool, select **Tools > Continuous/Discrete Conversions** to open the **Sample Time Conversion** window.
- 2** Click the **Discrete time with new sample time** option button.

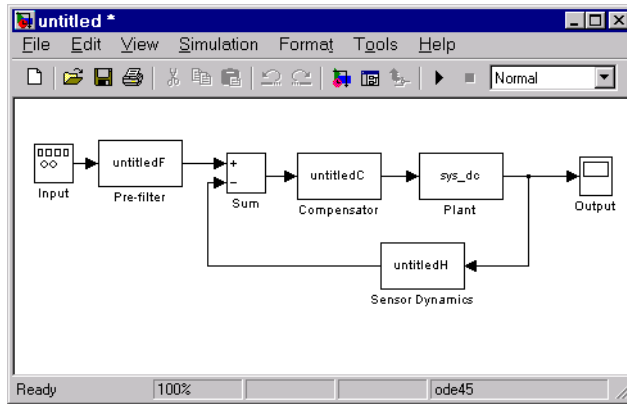


- 3 Specify a positive number for the sample time in the **Sample time (sec)** field.

Draw Simulink Diagram

Note You must have a license for Simulink to use this feature. If you do not have Simulink, this option does not appear under the **Tools** menu.

Select **Draw Simulink Diagram** from the Tools menu to draw a block diagram of your system (plant, compensator, prefilter, and sensor). The following diagram shows how the tool would render the DC motor example described in the *Control System Toolbox Getting Started Guide*.



Automated Tuning

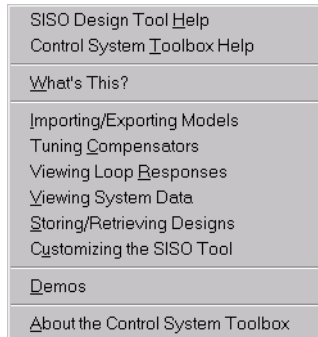
Select **Automated Tuning** from the Tools menu in the SISO Design Tool to open the **Automated Tuning** tab of the Control and Estimation Tools Manager. You can use this tab to perform automated tuning of compensators. For more information, see “Automated Tuning Design”.

Window

The **Window** menu item lists all of the windows open in the MATLAB technical computing environment. The first item is always the MATLAB Command Window. After that, the windows you have opened are listed in the order in which you opened them. Select any window from the list to make it the active window.

Help

Help brings you to various places in the Control System Toolbox help system. This figure shows the menu.



Each topic takes you to brief discussions of basic information about the SISO Design Tool and the Control System Toolbox software:

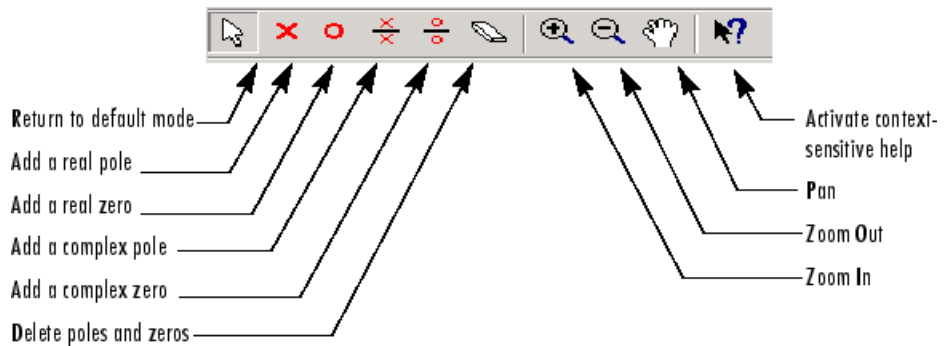
- **SISO Design Tool Help** — An overview of the SISO Design Tool
- **Control System Toolbox Help** — A roadmap for the Control System Toolbox help
- **What's This?** — Activates the "What's This?" cursor, which appears as a question mark. Click in various regions of the SISO Design Tool to see brief descriptions of the tool's features.
- **Importing/Exporting Models** — How to import models into the SISO Design Tool and how to export completed designs
- **Tuning Compensators** — Basic information about adjusting gains and adding dynamics to your prefilter (**F**) and compensator (**C**)
- **Viewing Loop Responses** — How to open an LTI Viewer containing loop responses for your system. Many response types are available.
- **Viewing System Data** — How to see information about your model
- **Storing/Retrieving Designs** — How to store and retrieve designed systems
- **Customizing the SISO Tool** — How to open the SISO Tool Preferences editor, which allows you to customize plot displays in the tool
- **Demos** — A link to Control System Toolbox demos
- **About the Control System Toolbox software** — The version number of your Control System Toolbox software

Using the Graphical Tuning Window Toolbar

The toolbar performs the following operations:

- Add and delete real and complex poles and zeros
- Zoom in and out
- Invoke the SISO Design Tool's context-sensitive help

This picture shows the toolbar.



Options Available from the Toolbar

You can use the tool tips feature to find out what a particular icon does. Just place your mouse over the icon in question, and you will see a brief description of what it does.

Once you've selected an icon, your mouse stays in that mode until you press the icon again.

You can reach all of these options from the right-click menus.

Using the Right-Click Menus in the Graphical Tuning Window

In this section...

“Overview of the Right-Click Menus” on page 13-57

“Add Pole/Zero” on page 13-58

“Delete Pole/Zero” on page 13-61

“Edit Compensator” on page 13-62

“Gain Target” on page 13-62

“Show” on page 13-62

“Multimodel Display” on page 13-63

“Design Requirements” on page 13-63

“Grid” on page 13-76

“Full View” on page 13-76

“Properties” on page 13-77

“Select Compensator” on page 13-78

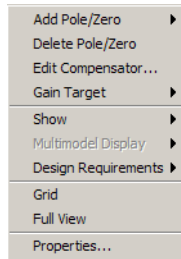
“Status Pane” on page 13-78

Overview of the Right-Click Menus

The Graphical Tuning window provides right-click menus for all the views available. These views include the root-locus, open-loop Bode diagrams, Nichols plot, and the closed-loop Bode diagrams. The menu items in each of these views are identical. The design requirements, however, differ, depending on which view you are accessing the menus from.

You can use the right-click menu to design a compensator by adding poles, zeros, lead, lag, and notch filters. In addition, you can use this menu to add grids and zoom in on selected regions. Also, you can open each view's **Property Editor** to customize units and other elements of the display.

Note Click items on the menu bar pictured below to get help contents.



Open-Loop Right-Click Menu

Note that if you have a closed-loop response, the Gain Target menu item is replaced by “Select Compensator” on page 13-78.

Add Pole/Zero

The **Add Pole/Zero** menu options give you the ability to add dynamics to your compensator design, including poles, zeros, lead and lag networks, and notch filters. The following pole/zero configurations are available:

- **Real Pole**
- **Complex Pole**
- **Integrator**
- **Real Zero**
- **Complex Zero**
- **Differentiator**
- **Lead**
- **Lag**
- **Notch**

In all but the integrator and differentiator, once you select the configuration, your cursor changes to an ‘x’. To add the item to your compensator design,

place the x at the desired location on the plot and left-click your mouse. You will see the root locus design automatically update to include the new compensator dynamics.

The notch filter has three adjustable parameters. For a discussion about how to add and adjust notch filters, see "Adding a Notch Filter" in the *Control System Toolbox Getting Started Guide*.

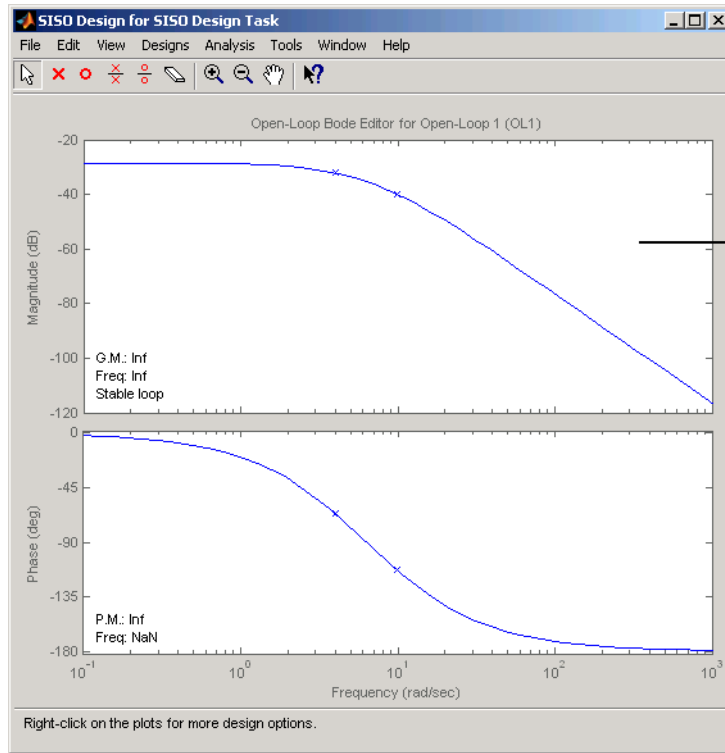
Note For systems with FRD plants, you cannot add or modify poles and zeros outside the plotted frequency range on Bode and Nichols plots. Instead, you can make such modifications using the Compensator Editor pane of the Control and Estimation Tools Manager. For more information, see "Compensator Editor" on page 13-18.

Example: Adding a Complex Pair of Poles

This example shows you how to add a complex pair of poles to the open-loop Bode diagram. First, type

```
load ltiexamples
sisotool('bode',sys_dc)
```

at the MATLAB prompt. This opens the SISO Design Tool with the DC motor example loaded and the open-loop Bode diagram displayed in the Graphical Tuning window.

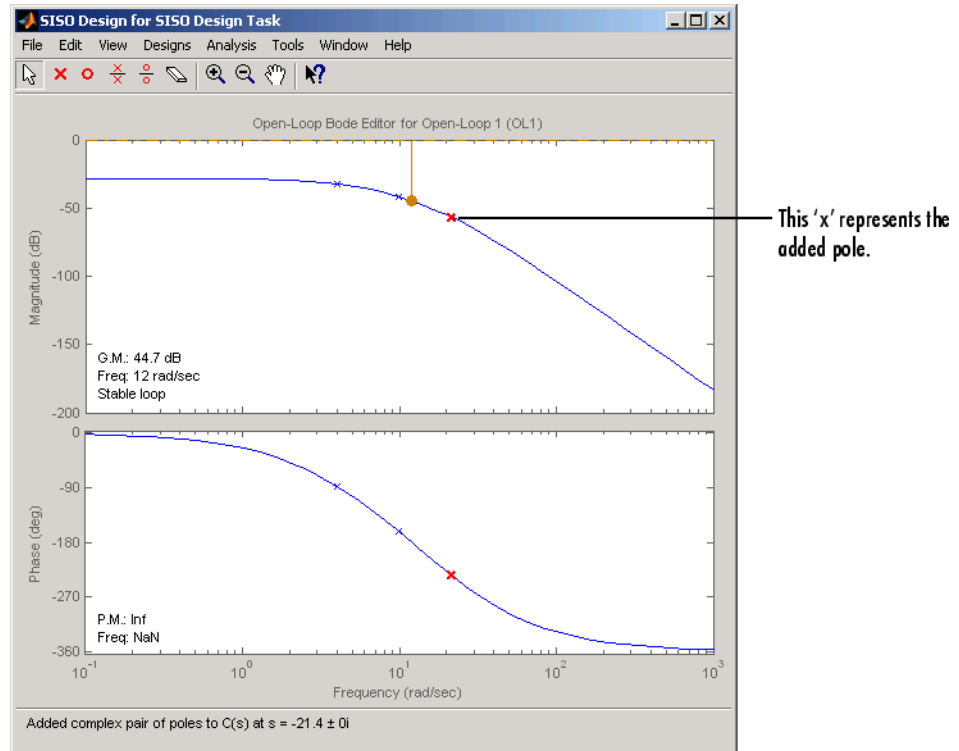


After selecting **Add Pole/Zero->Complex Pole** from the right-click menu, use the mouse cursor to specify the frequency of the complex pole pair.

To add a complex pair of poles:

- 1 Select **Add Pole/Zero->Complex Pole** from the right-click menu
- 2 Place the mouse cursor where you want the pole to be located
- 3 Left-click to add the pole

Your Graphical Tuning window should look similar to this.



In the case of Bode diagrams, when you place a complex pole, the default damping value is 1, which means you have a double real pole. To change the damping, grab the red 'x' by left-clicking on it and drag it upward with your mouse. You will see damping ratio change in the Status pane at the bottom of the SISO Design Tool.

Delete Pole/Zero

Select **Delete Pole/Zero** to delete poles and zeros from your compensator design. When you make this selection, your cursor changes to an eraser. Place the eraser over the pole or zero you want to delete and left-click your mouse.

Note the following:

- You can only delete compensator poles and zeros. Plant (**G** in the feedback structure pane) poles and zeros cannot be altered.
- If you delete one of a pair of poles or zeros, the other member of the pair is also removed.

Edit Compensator

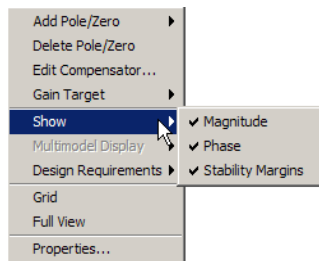
Edit Compensator opens the **Compensator Editor** pane in the SISO Design Task. You can use this pane to adjust the compensator gain and add or remove compensator poles and zeros from your compensator (**C**) or prefilter (**F**) design. See “Compensator Editor” on page 13-18 for a discussion of this pane.

Gain Target

This feature is intended for users of the Simulink Control Design software. It is nonfunctional in the Control System Toolbox software.

Show

Use **Show** to select/deselect the display of characteristics relevant to which view you are working with. This figure displays the Show submenu for the open-loop Bode diagram.

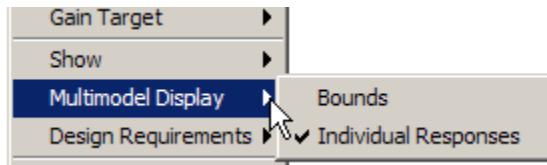


For this particular view, the options available are magnitude, phase, and stability margins. Selecting any of these toggles between showing and hiding the feature. A check next to the feature means that it is currently displayed on the Bode diagram plots. Although the characteristics are different for each view in the Graphical Tuning window, they all toggle on and off in the same manner.

Multimodel Display

This menu is enabled only when you import or open the SISO Design Tool with row or column arrays of LTI models.

Use **Multimodel Display** to view the responses of models in an LTI array as individual responses or as an envelope encompassing all responses on the Bode and Nichols plots.



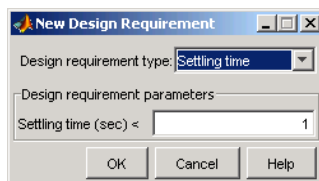
On the Root Locus plot, use this menu to show or hide the pole/zero locations of all models in the array, except the nominal one.

For more information on control design analysis for multiple models, see “Control Design Analysis of Multiple Models” in the Getting Started Guide.

Design Requirements

When designing compensators, it is common to have design specifications that call for specific settling times, damping ratios, and other characteristics. The Graphical Tuning window provides tools for design requirements that can help make the task of meeting design specifications easier.

The New Design Requirement dialog box lets you create design requirements by creating graphical representations for feasible and nonfeasible regions, automatically changes to reflect which design requirements are available for the view in which you are working. Select **Design Requirements > New** to open the New Design Requirement dialog box.



Since each view has a different set of design requirements, click the following links to go to the appropriate descriptions:

- “Design Requirements for the Root Locus” on page 13-64
- “Design Requirements for Open- and Closed-Loop Bode Diagrams” on page 13-67
- “Design Requirements for Open-Loop Nichols Plots” on page 13-71
- “LTI Viewer for SISO Design Task Design Requirements” on page 13-79

For row or column arrays of LTI models, the design requirements are for the nominal plant that you are designing the controller for. You can analyze the effects of this controller on the remaining models. See “Control Design Analysis of Multiple Models” in the Getting Started Guide.

Design Requirements for the Root Locus

For the root locus, you can use the following design requirements:

- “Settling Time” on page 13-64
- “Percent Overshoot” on page 13-65
- “Damping Ratio” on page 13-65
- “Natural Frequency” on page 13-65
- “Region Constraint” on page 13-65

Use the **Design requirement type** drop-down list to select a design requirement. In each case, to specify the design requirement, enter the value in the **Design requirement parameters** pane. You can select any or all of them, or have more than one of each.

Settling Time. If you specify a settling time in the continuous-time root locus, a vertical line appears on the root locus plot at the pole locations associated with the settling time value provided (using a first-order approximation). This vertical line is exact for a second order system and is only an approximation for higher order systems. In the discrete-time case, the design requirement boundary is a curved line.

Percent Overshoot. Specifying percent overshoot in the continuous-time root locus causes two rays, starting at the root locus origin, to appear. These rays are the locus of poles associated with the percent value (using a second-order approximation). In the discrete-time case, the design requirement appears as two curves originating at (1,0) and meeting on the real axis in the left-hand plane.

Note that the percent overshoot (p.o.) design requirement can be expressed in terms of the damping ratio, as in this equation:

$$p.o. = 100 \exp\left(-\frac{\pi\zeta}{\sqrt{1-\zeta^2}}\right)$$

where ζ is the damping ratio.

Damping Ratio. Specifying a damping ratio in the continuous-time root locus causes two rays, starting at the root locus origin, to appear. These rays are the locus of poles associated with the damping ratio. In the discrete-time case, the design requirement boundary appears as curved lines originating at (1,0) and meeting on the real axis in the left-hand plane.

Natural Frequency. If you specify a natural frequency lower bound, a semicircle centered around the root locus origin appears. If you specify a natural frequency upper bound, the inverse of this semicircle appears. The radius equals the natural frequency.

Region Constraint. Specifying a region constraint at given locations causes black lines and a yellow area to appear. The vertices of this free-form piecewise region are defined by the specified real and imaginary values.

Example: Adding Damping Ratio Design Requirements

This example adds a damping ratio design requirement of 0.707.

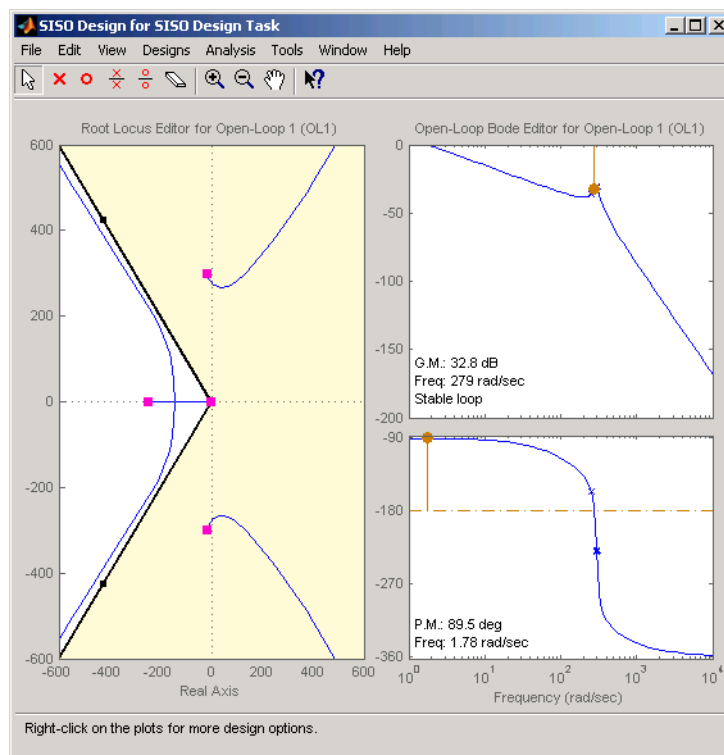
1 At the MATLAB prompt, type the following:

```
load ltiexamples
sisotool(sys_dc)
```

This opens the SISO Design Tool with the DC motor example imported.

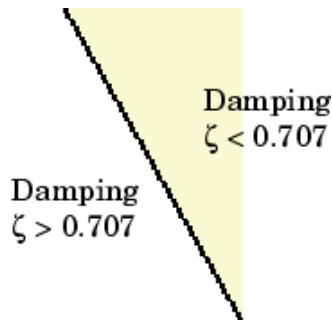
- 2 From the root locus right-click menu, select **Design Requirement** > **New** to open the New Design Requirement dialog box.
- 3 To add the design requirement, select **Damping Ratio** as the design requirement. Click **OK** to accept the default damping ratio of 0.707.

The Graphical Tuning window should now look similar to this figure.



Damping Ratio Requirements in the Root Locus

The two rays centered at (0,0) represent the damping ratio boundaries. The dark edge is the region boundary, and the shaded area outlines the exclusion region. This figure explains what this means for this design requirement.



You can, for example, use this design requirement to ensure that the closed-loop poles, represented by the red squares, have some minimum damping. Try adjusting the gain until the damping ratio of the closed-loop poles is 0.7.

Design Requirements for Open- and Closed-Loop Bode Diagrams

For both the open- and closed-loop Bode diagrams, you have the following options:

- “Upper Gain Limit” on page 13-67
- “Lower Gain Limit” on page 13-68
- “Gain and Phase Margin” on page 13-68

Specifying any of these design requirements causes lines to appear in the Bode magnitude curve. To specify an upper or lower gain limit, enter the frequency range, the magnitude limit, and/or the slope in decibels per decade, in the appropriate fields of the New design requirement dialog box. You can have as many gain limit design requirements as you like in your Bode magnitude plots.

Upper Gain Limit. You can specify one or multiple piecewise linear upper gain limits over a frequency range, which appear as straight lines on the Bode magnitude curve. You must select frequency limits, the upper gain limit in decibels, and the slope in dB/decade.

Lower Gain Limit. You can specify one or multiple lower gain limit in the same fashion as the upper gain limit.

Gain and Phase Margin. You can specify a lower bound for the gain, the phase margin, or both. The specified bounds appear in text on the Bode magnitude plot.

Example: Adding Upper Gain Limits

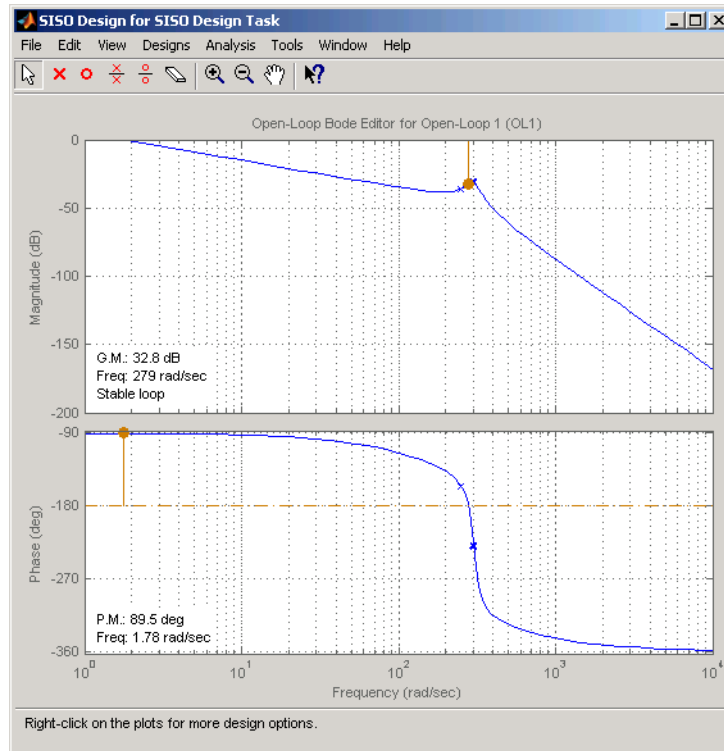
This example shows you how to add two upper gain limit requirements to the open-loop Bode diagram.

- 1** At the MATLAB prompt, type the following:

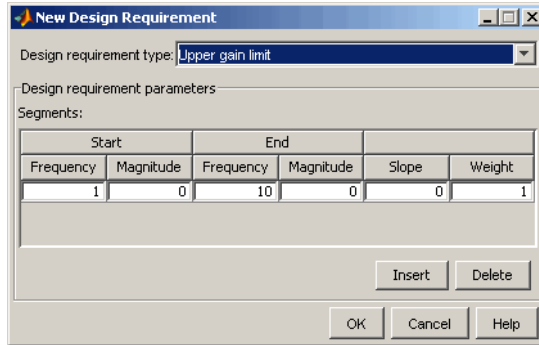
```
load ltiexamples  
sisotool('bode',Gservo)
```

This opens the SISO Design Tool with the servomechanism model loaded.

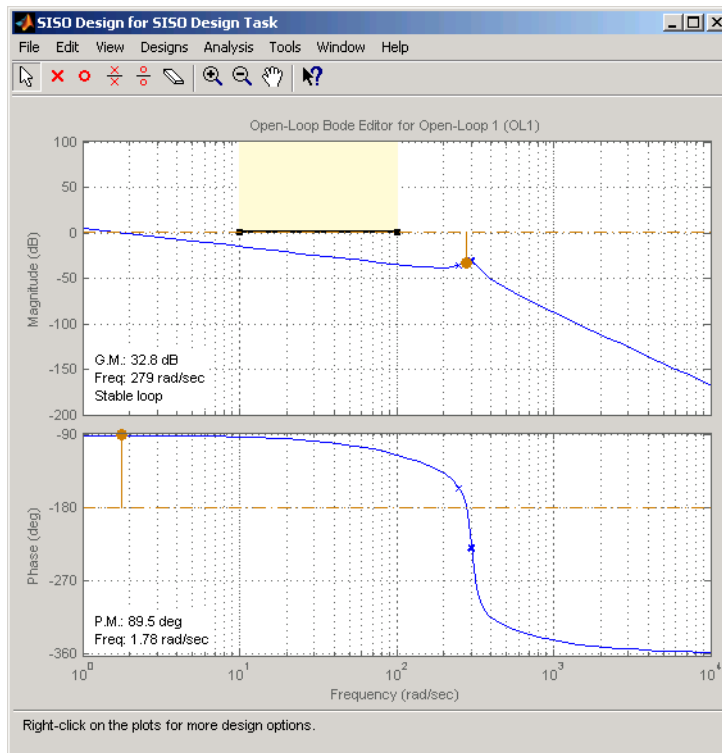
- 2** Use the right-click menu to add a grid.



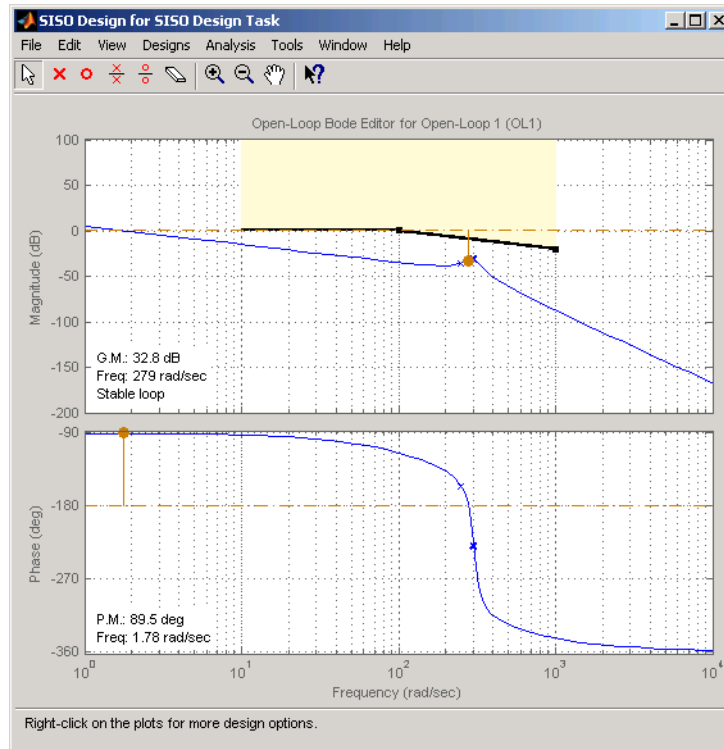
- 3 To add an upper gain limit requirement of 0 dB from 10 rad/sec to 100 rad/sec, open the New Design Requirement dialog box and select **Upper gain limit** from the pull-down menu. Fill in the dialog box fields as shown in the following figure.



Your Graphical Tuning window should now look like this (you may have to adjust some axis limits).



- 4 To constrain the roll off, open the New Design Requirement dialog box and add an upper gain limit from 100 rad/sec to 1000 rad/sec with a slope of -20 db/decade. This figure shows the result.



With these design requirements in place, you can see how much you can increase the compensator gain and still meet design specifications.

Note that you can change the design requirements by moving them with your mouse. See “Editing Design Requirements” on page 13-74 for more information.

Design Requirements for Open-Loop Nichols Plots

For open-loop Nichols plots, you have the following design requirement options:

- “Phase Margin” on page 13-72
- “Gain Margin” on page 13-72
- “Closed-Loop Peak Gain” on page 13-72
- “Gain-Phase Design Requirement” on page 13-72

Specifying any of these design requirements causes lines or curves to appear in the Nichols plot. In each case, to specify the design requirement, enter the value in the **Design requirement parameters** pane. You can select any or all of them, or have more than one of each.

Phase Margin. Specify a minimum phase margin at a given location. For example, you can require a minimum of 30 degrees at the -180 degree crossover. The phase margin specified should be a number greater than 0. The location must be a -180 plus a multiple of 360 degrees. If you enter an invalid location point, the closest valid location is selected.

Gain Margin. Specify a gain margin at a given location. For example, you can require a minimum of 20 dB at the -180 degree crossover. The location must be -180 plus a multiple of 360 degrees. If you enter an invalid location point, the closest valid location is selected.

Closed-Loop Peak Gain. Specify a peak closed-loop gain at a given location. The specified dB value can be positive or negative. The design requirement follows the curves of the Nichols plot grid, so it is recommended that you have the grid on when using this feature.

Gain-Phase Design Requirement. Specify both a gain and phase design requirement at a given location. The vertices of this free-form piecewise region are defined by the specified open-loop phase and open-loop gain values.

Example: Adding a Closed-Loop Peak Gain Design Requirement

This example shows how to add a closed-loop peak gain design requirement to the Nichols plot.

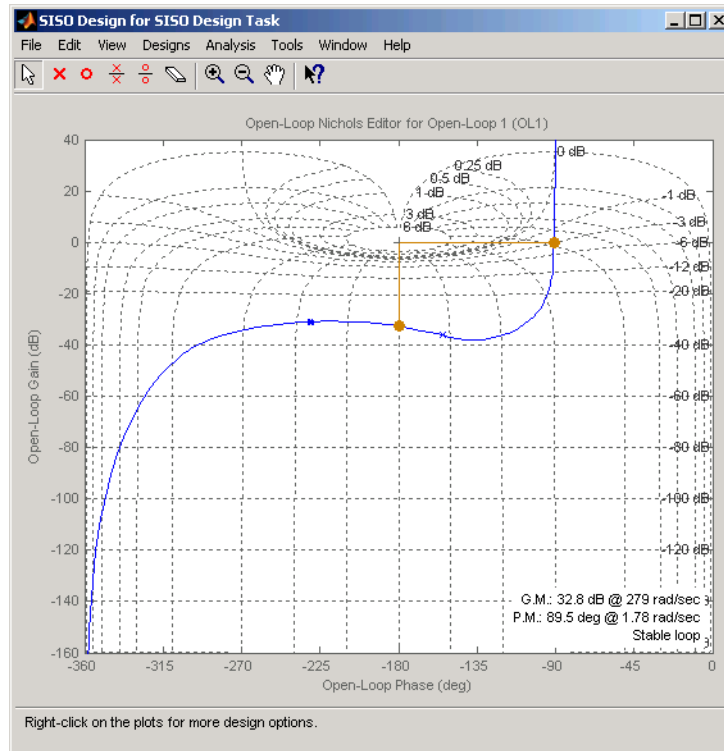
1 At the MATLAB prompt, type the following:

```
load ltiexamples
```

```
sisotool('nichols',Gservo)
```

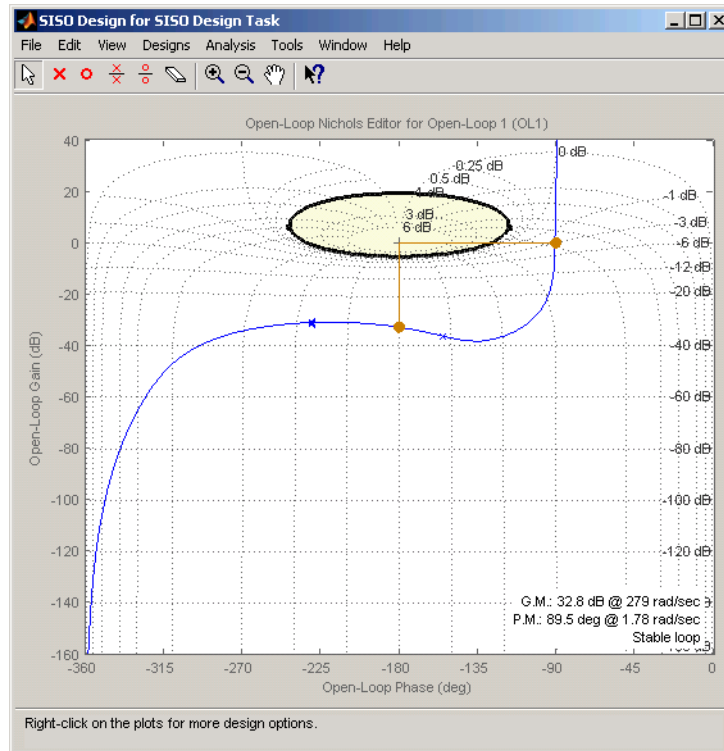
This opens the SISO Design Tool with Gservo imported as the plant.

- 2 Use the right-click menu to add a grid, as this figure shows.



- 3 To add closed-loop peak gain of 1 dB at -180 degrees, open the New Design Requirement dialog box and select **Closed-Loop Peak Gain** from the pull-down menu. Set the peak gain field to 1 dB.

The figure shows the resulting design requirement.



As long as the curve is outside of the gray region, the closed-loop gain is guaranteed to be less than 1 dB. Note that this is equivalent, up to second order, to specifying the peak overshoot in the time domain. In this case, a 1 dB closed-loop peak gain corresponds to an overshoot of 15%.

Editing Design Requirements

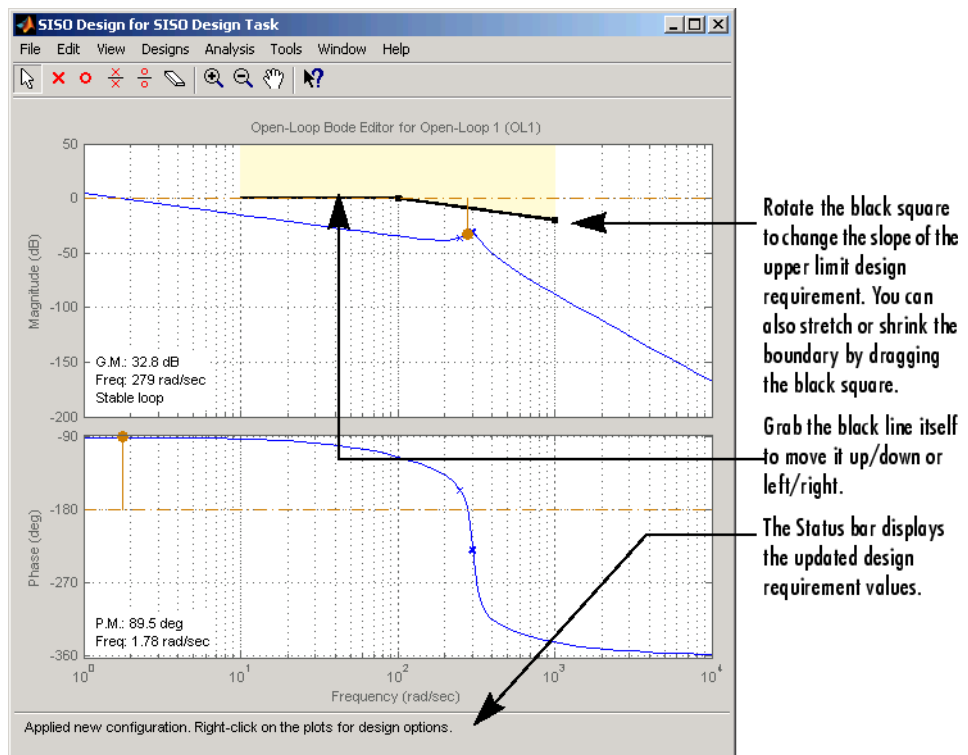
To edit an existing design requirement, left-click on the design requirement boundary to select it. Two black squares appear on the design requirement when it is selected. In general, there are two ways to adjust a design requirement:

- Click on the design requirement boundary and drag it. Generally, this does not change the shape of the boundary. That is, the adjustment is strictly a translation of the design requirement.

- Grab a black square and drag it. In this case, you can rotate, expand, and/or contract the design requirement.

For example, in Bode diagrams you can move an upper gain limit by clicking on it and moving it anywhere in the plot region. As long as you haven't grabbed a black square, the length and slope of the gain limit will not change as you move the line. On the other hand, you can change the slope of the upper gain limit by grabbing one of the black squares and rotating the line. In all cases, the Status pane at the bottom of the Graphical Tuning window displays the design requirement values as they change.

This figure shows the process of editing an upper gain limit in the open-loop Bode diagram.



An alternative way to adjust a design requirement is to select **Design Requirements->Edit** from the right-click menu. The **Edit Design Requirement** window opens.



To adjust a design requirement, select the boundary by clicking on it and change the values in the fields of the Design requirement parameters pane. If you have additional design requirement in, for example, the Bode diagram, you can edit them directly from this window by selecting **Open-Loop Bode** from the **Editor** menu.

Deleting Design Requirements

To delete a design requirement, place your cursor directly over the design requirement yellow region. Right-click to open a menu containing **Edit** and **Delete**. Select **Delete** from the menu list; this eliminates the design requirement. You can also delete design requirements by left-clicking on a design requirement boundary and then pressing the **BackSpace** or **Delete** key on your keyboard.

Finally, you can delete design requirements by selecting **Undo Add Design Requirement** from the **Edit** menu, or pressing **Ctrl+Z** if adding design requirements was the last action you took.

Grid

Grid adds a grid to the selected plot.

Full View

Selecting **Full View** causes the plot to scale limits so that the entire curve is visible.

Properties

Properties opens the **Property Editor**, which is a GUI for customizing root locus, Bode diagrams, and Nichols plots inside the Graphical Tuning window. The Property Editor automatically reconfigures as you select among the different plots open.

This picture shows the open window for the root locus.



You can use this window to change titles and axis labels, reset axes limits, add grid lines, and change the aspect ratio of the plot. Note that you can also activate this menu by double-clicking anywhere in the root locus away from the curve.

There are only three panes in the Property Editor: Labels, Limits, and Options. The configuration of each page differs, depending on whether you're working with the root-locus, Bode diagrams, or the open-loop Nichols plot. Click the **Help** button on the Property Editor you have open to view information specific to that editor, or click on the links below:

- [Root locus](#)
- [Bode diagram](#)
- [Nichols plot](#)

Select Compensator

This option allows you to select which compensator to edit for closed-loop Bode response.

Status Pane

The Status pane is located at the bottom of the Graphical Tuning window. It displays the most recent action you have performed, occasionally provides advice on how to use the window, and tracks key parameters when moving objects in the design views.

LTI Viewer for SISO Design Task Design Requirements

In this section...

“Overview of LTI Viewer Design Requirements” on page 13-79

“Available Design Requirements in the LTI Viewer” on page 13-79

“Example: Time Domain Requirement” on page 13-80

Overview of LTI Viewer Design Requirements

You can use the LTI Viewer for SISO Design Tasks to specify both time and frequency domain requirements in analysis plots. Adding and editing design requirements is similar to those illustrated in the Graphical Tuning window.

Note To add design requirements, you must open the LTI Viewer from the SISO Design Task in the Control and Estimation Tools Manager. Design requirements are not available from an LTI Viewer that is opened using the `ltiview` command.

For row or column arrays of LTI models, the design requirements are for the nominal model that you are designing the controller for. You can analyze the effects of this controller on the remaining models. See “Control Design Analysis of Multiple Models” in the Getting Started Guide.

Available Design Requirements in the LTI Viewer

The design requirements for Bode, Root Locus, and Nichols plots can be applied to both graphical tuning windows and the LTI viewer. See “Design Requirements” on page 13-63 for information on graphical tuning design requirements.

You can also specify the following design requirements for both step and impulse response plots:

- **Upper time response bounds** — Creates an upper amplitude bound for a specified time duration.

- **Lower time response bounds** — Creates a lower amplitude bound for a specified time duration.

If you are using a step response plot, you can also specify the following design requirement:

- **Step response bounds** — Creates a group of upper and lower time response bounds, in the shape of a step response envelope, to encompass your specified design requirement parameters.

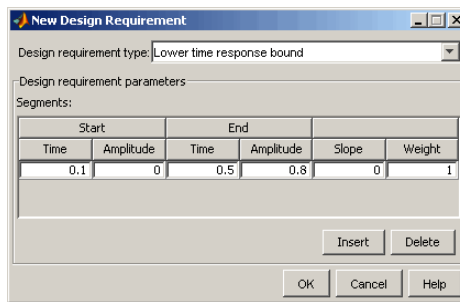
Example: Time Domain Requirement

This example shows you how to create a lower bound time response design requirement.

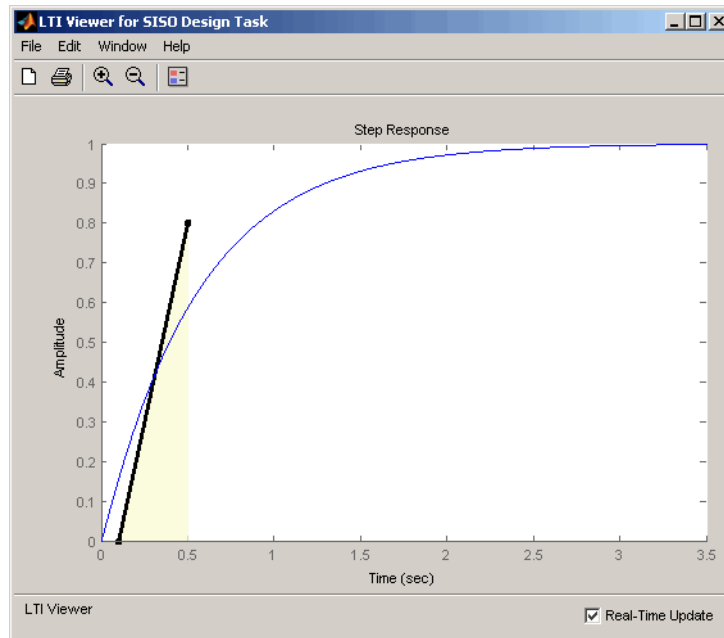
- 1 At the MATLAB prompt, type the following:

```
load ltiexamples
sisotool(Gservo)
```

- 2 From the **Analysis Plot** pane, select step response. See “Analysis Plots” on page 13-23 if you are unfamiliar with this task.
- 3 Select Design Requirements->New from the LTI Viewer right-click menu.
- 4 Select Lower time response bound from the Design requirements menu.
- 5 Set Time from 0.1 to 0.5 s.
- 6 Set Amplitude from 0 to 0.8. Your New Design Requirement window should look like this.



- 7 Click OK. This adds the design requirement. Your step response should look like this.



LTI Viewer

- “Basic LTI Viewer Tasks” on page 14-2
- “Using the Right-Click Menu in the LTI Viewer” on page 14-4
- “Importing, Exporting, and Deleting Models in the LTI Viewer” on page 14-12
- “Selecting Response Types” on page 14-16
- “Analyzing MIMO Models” on page 14-20
- “Customizing the LTI Viewer” on page 14-25

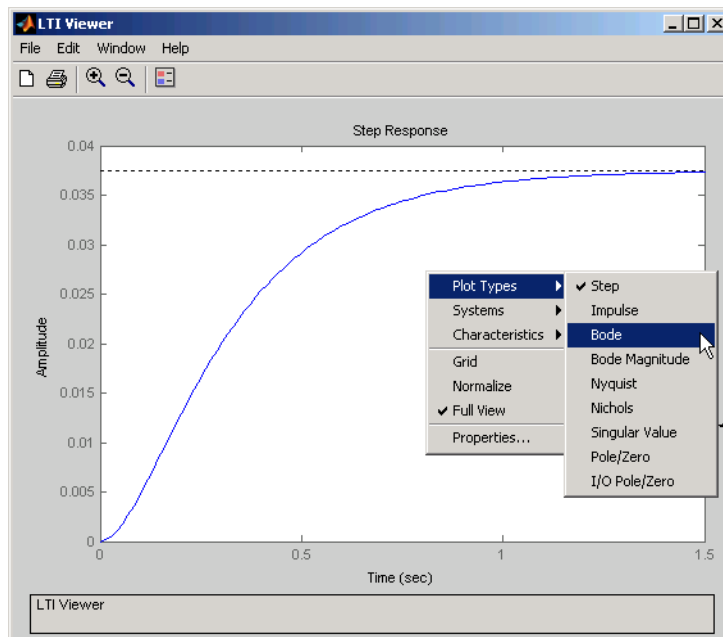
Basic LTI Viewer Tasks

LTI Viewer is a graphical user interface (GUI) that simplifies the analysis of linear, time-invariant systems. You use the LTI Viewer to view and compare the response plots of SISO and MIMO systems, or of several linear models at the same time. You can generate time and frequency response plots to inspect key response parameters, such as rise time, maximum overshoot, and stability margins.

The easiest way to work with the LTI Viewer is to use the right-click menus. For example, type

```
load ltiexamples
ltiview(sys_dc)
```

at the MATLAB prompt. The default plot is a step response.



The LTI Viewer can display up to seven different plot types simultaneously, including step, impulse, Bode (magnitude and phase or magnitude only), Nyquist, Nichols, sigma, pole/zero, and I/O pole/zero.

See `ltiview` for help on the function that opens an LTI Viewer. For examples of how to use the LTI Viewer, see Analyzing Models in the *Control System Toolbox Getting Started Guide*.

Using the Right-Click Menu in the LTI Viewer

| In this section... |
|--|
| “Overview of the Right-Click Menu” on page 14-4 |
| “Setting Characteristics of Response Plots” on page 14-4 |
| “Adding Design Requirements” on page 14-9 |

Overview of the Right-Click Menu

The quickest way to manipulate views in the LTI Viewer is use the right-click menu. You can access several LTI Viewer controls and options, including:

- **Plot Type** — Changes the plot type
- **Systems** — Selects or deselects any of the models loaded in the LTI Viewer
- **Characteristics** — Displays key response characteristics and parameters
- **Grid** — Adds grids to your plot
- **Properties** — Opens the **Property Editor**, where you can customize plot attributes
- **Design Requirements** — Opens the New Design Requirement window for adding step response design requirements to your plot (available only for LTI Viewers linked to the Graphical Tuning window of the SISO Design Tool)

In addition to right-click menus, all response plots include data markers. These allow you to scan the plot data, identify key data, and determine the source system for a given plot.

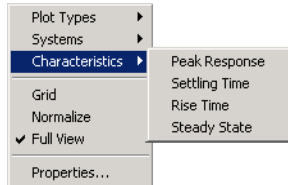
Setting Characteristics of Response Plots

The **Characteristics** menu changes for each plot response type. Characteristics refers to response plot information, such as peak response, or, in some cases, rise time and settling time.

The next sections describe the menu items for each of the eight plot types.

Step Response

Step plots the model's response to a step input.



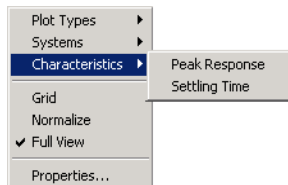
You can display the following information in the step response:

- **Peak Response** — The largest deviation from the steady-state value of the step response
- **Settling Time** — The time required for the step response to decline and stay at 5% of its final value
- **Rise Time** — The time required for the step response to rise from 10% to 90% of its final value
- **Steady-State** — The final value for the step response

Note You can change the definitions of settling time and rise time using the **Characteristics** pane of the Control System Toolbox editor, the LTI Viewer editor, or the Property editor.

Impulse Response

Impulse Response plots the model's response to an impulse.

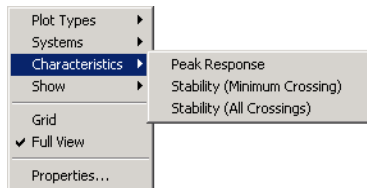


The LTI Viewer can display the following information in the impulse response:

- **Peak Response** — The maximum positive deviation from the steady-state value of the impulse response
- **Settling Time** — The time required for the step response to decline and stay at 5% of its final value

Bode Diagram

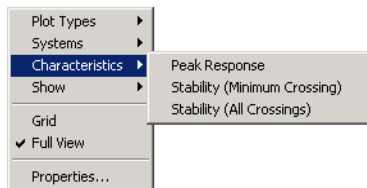
Bode plots the open-loop Bode phase and magnitude diagrams for the model.



The LTI Viewer can display the following information in the Bode diagram:

- **Peak Response** — The maximum value of the Bode magnitude plot over the specified region
- **Stability Margins (Minimum Crossing)** — The minimum phase and gain margins. The gain margin is defined to the gain (in dB) when the phase first crosses -180° . The phase margin is the distance, in degrees, of the phase from -180° when the gain magnitude is 0 dB.
- **Stability Margins (All Crossings)** — Display all stability margins

Bode Magnitude

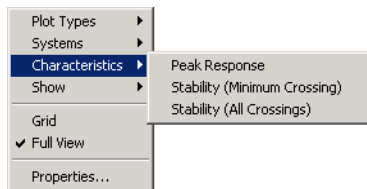


Bode Magnitude plots the Bode magnitude diagram for the model.

The LTI Viewer can display the following information in the Bode magnitude diagram:

- **Peak Response**, which is the maximum value of the Bode magnitude in decibels (dB), over the specified range of the diagram.
- **Stability (Minimum Crossing)** — The minimum gain margins. The gain margin is defined to the gain (in dB) when the phase first crosses -180° .
- **Stability (All Crossings)** — Display all gain stability margins

Nyquist Diagrams



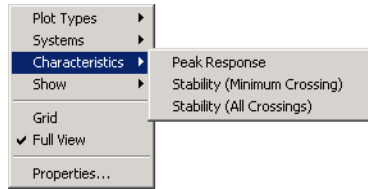
Nyquist plots the Nyquist diagram for the model.

The LTI Viewer can display the following types of information in the Nyquist diagram:

- **Peak Response** — The maximum value of the Nyquist diagram over the specified region
- **Stability (Minimum Crossing)** — The minimum gain and phase margins for the Nyquist diagram. The gain margin is the distance from the origin to the phase crossover of the Nyquist curve. The phase crossover is where the curve meets the real axis. The phase margin is the angle subtended by the real axis and the gain crossover on the circle of radius 1.
- **Stability (All Crossings)** — Display all gain stability margins

Nichols Charts

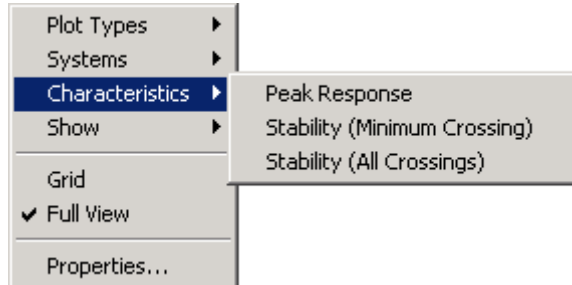
Nichols plots the Nichols Chart for the model.



The LTI Viewer can display the following types of information in the Nichols chart:

- **Peak Response** — The maximum value of the Nichols chart in the plotted region.
- **Stability (Minimum Crossing)** — The minimum gain and phase margins for the Nichols chart.
- **Stability (All Crossings)** — Display all gain stability margins

Singular Values



Singular Values plots the singular values for the model.

The LTI Viewer can display the **Peak Response**, which is the largest magnitude of the Singular Values curve over the plotted region.

Pole/Zero and I/O Pole/Zero

Pole/Zero plots the poles and zeros of the model with 'x' for poles and 'o' for zeros. I/O Pole/Zero plots the poles and zeros of I/O pairs.

There are no **Characteristics** available for pole-zero plots.

Adding Design Requirements

If you open an LTI Viewer for the Graphical Tuning window, you have plots linked to your compensator design. In this environment, the LTI Viewer provides access to design requirements, a set of graphical tools for creating constraints in your design plots.

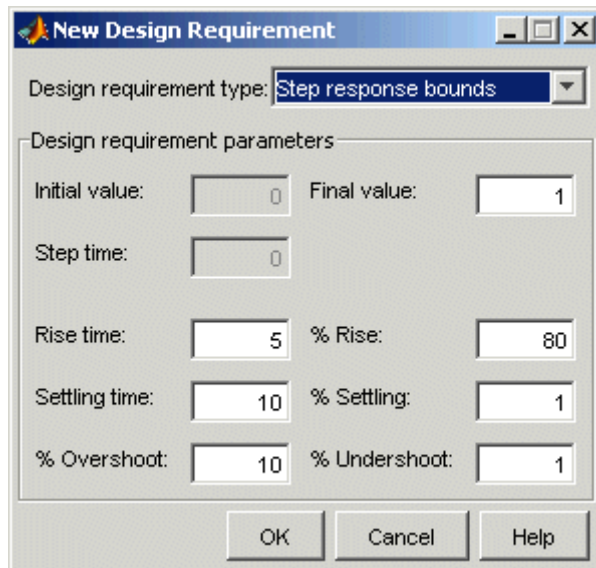
In addition to all the design requirements available in the Graphical Tuning window, the LTI Viewer has the step response design requirements described in “Choosing Step Response Specifications” on page 14-9.

For more information on adding design requirements to LTI Viewer plots, see “LTI Viewer for SISO Design Task Design Requirements” on page 13-79.

Note Design requirements are not available from an LTI Viewer that is opened using the `ltiview` command.

Choosing Step Response Specifications

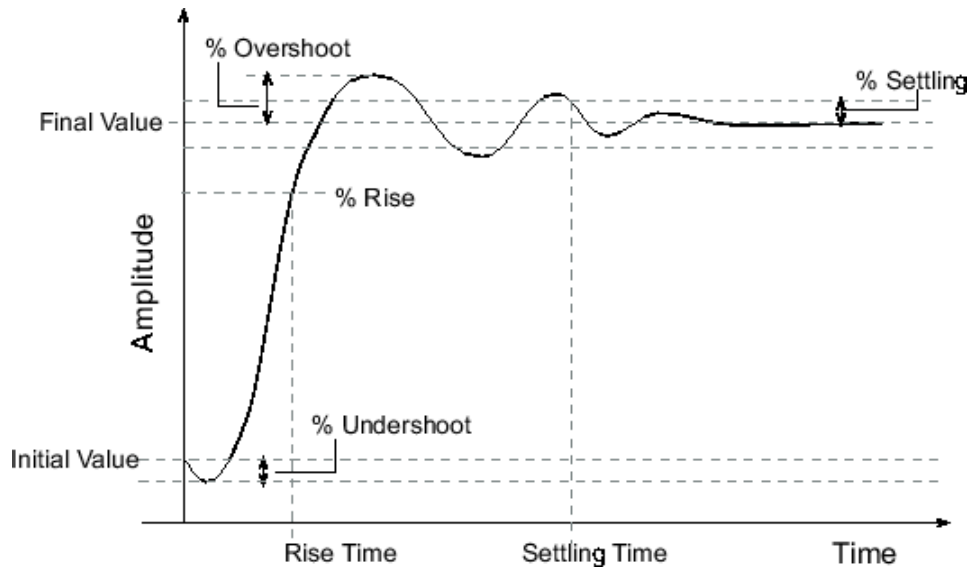
To specify step response characteristics select **Design Requirements > New** in the right-click menu. This action opens the **New Design Requirements** editor. Select **step response bounds** from the **Design requirement type** pull down menu to display the step response specifications as shown below.



The top three options specify the details of the step input:

- **Initial value:** input level before the step occurs. This option is grayed out because LTI systems always have initial value equal to 0.
- **Step time:** time at which the step takes place. This option is grayed out since LTI systems always have an initial time equal to 0.
- **Final value:** input level after the step occurs

The remaining options specify the characteristics of the response signal. Each of the step response characteristics is described in the figure below.



- **Rise time:** The time taken for the response signal to reach a specified percentage of the step's range. The step's range is the difference between the final and initial values.
- **% Rise:** The percentage used in the rise time.
- **Settling time:** The time taken until the response signal settles within a specified region around the final value. This settling region is defined as the final step value plus or minus the specified percentage of the final value.
- **% Settling:** The percentage used in the settling time.
- **% Overshoot:** The amount by which the response signal can exceed the final value. This amount is specified as a percentage of the step's range. The step's range is the difference between the final and initial values.
- **% Undershoot:** The amount by which the response signal can undershoot the initial value. This amount is specified as a percentage of the step's range. The step's range is the difference between the final and initial values.

Enter values for the response specifications in the **Design Requirements** editor, based on the requirements of your model, and then click **OK**. The constraint edges will now reflect the constraints specified.

Importing, Exporting, and Deleting Models in the LTI Viewer

In this section...

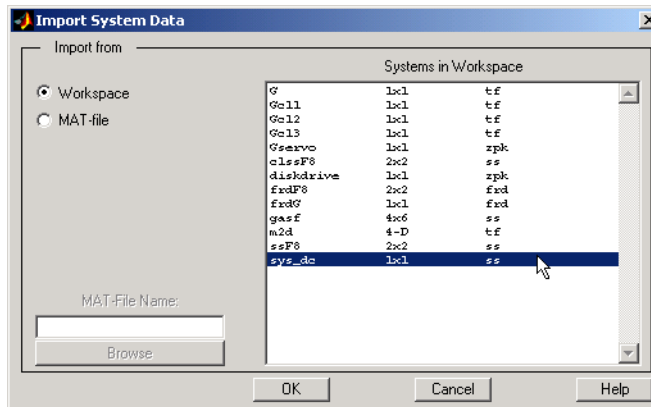
“Importing Models” on page 14-12

“Exporting Models” on page 14-13

“Deleting Models” on page 14-14

Importing Models

To import models into the LTI Viewer, select **Import** under the **Edit** menu. This opens the **LTI Browser**, shown below.



Use the **LTI Browser** to import LTI models into or from the LTI Viewer workspace.

To import a model:

- Click on the desired model in the LTI Browser List. To perform multiple selections:
 - Hold the Control key and click on nonadjacent models.
 - Hold the Shift key while clicking to select multiple adjacent models.

- Click the **OK** or **Apply** Button

Note that the **LTI Browser** lists only the LTI models in the MATLAB workspace.

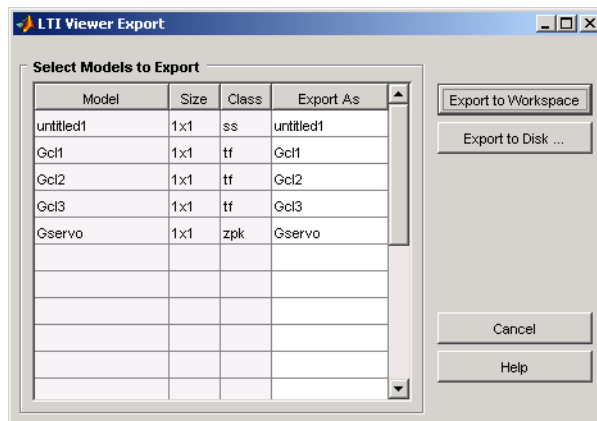
Alternatively, you can directly import a model into the LTI Viewer using the `ltiview` function, as in

```
ltiview({'step', 'bode'}, modelname)
```

See the `ltiview` function for more information.

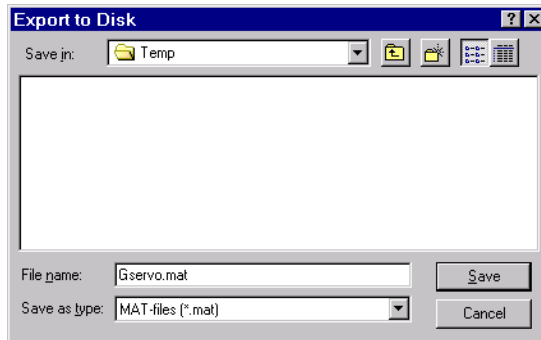
Exporting Models

Use **Export** in the **File** menu to open the **LTI Viewer Export** window, shown below.



The **LTI Viewer Export** window lists all the models with responses currently displayed in your LTI Viewer. You can export models back to the MATLAB workspace or to disk.

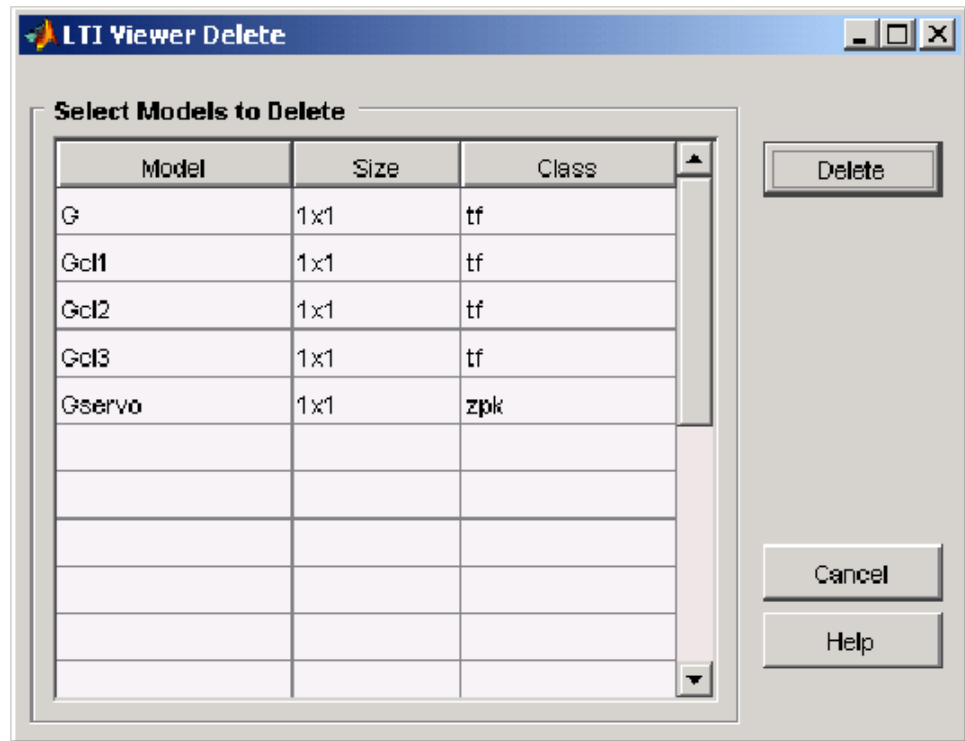
To export single or multiple models, follow the steps described in the importing models section above. If you choose **Export to Disk**, this window opens.



Choose a name for your model(s) and click **Save**. Your models are stored in a MAT-file.

Deleting Models

Select **Edit->Delete Systems** to open the **LTI Viewer Delete** window.



To delete a model:

- Click on the desired model in the Model list. To perform multiple selections:
 - 1 Click and drag over several variables in the list.
 - 2 Hold the Control key and click on individual variables.
 - 3 Hold the Shift key while clicking, to select a range.

Click the **Delete** button.

Selecting Response Types

In this section...

“Methods for Selecting Response Types” on page 14-16

“Right Click Menu: Plot Type” on page 14-16

“Plot Configurations Window” on page 14-16

“Line Styles Editor” on page 14-18

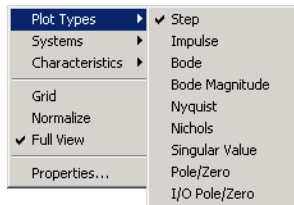
Methods for Selecting Response Types

There are two methods for selecting response plots in the LTI Viewer:

- Selecting **Plot Type** from the right-click menus
- Opening the **Plot Configurations** window

Right Click Menu: Plot Type

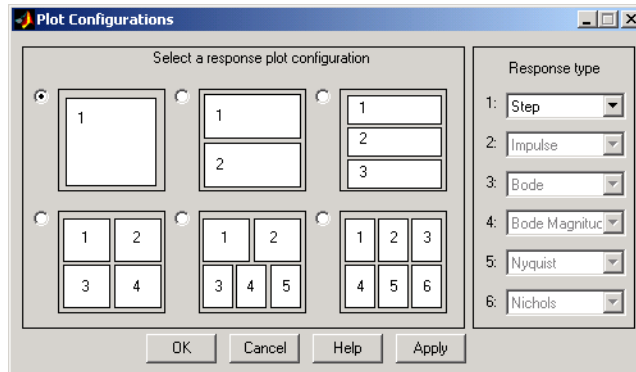
If you have a plot open in the LTI Viewer, you can switch to any other response plot available by selecting **Plot Type** from the right click menu.



To change the response plot, select the new plot type from the **Plot Type** submenu. The LTI Viewer automatically displays the new response plot.

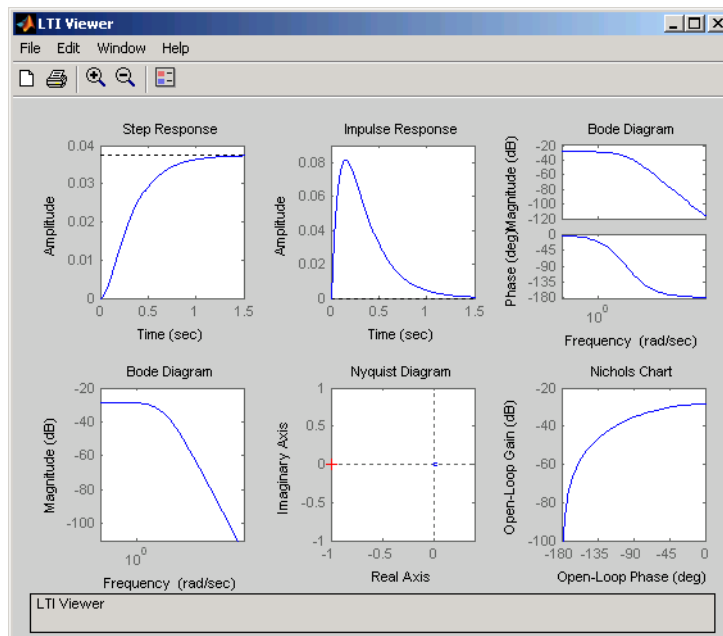
Plot Configurations Window

The Plot Type feature of the right-click menu works on existing plots, but you can also add plots to an LTI Viewer by using the **Plot Configurations** window. By default, the LTI Viewer opens with a closed-loop step response. To reconfigure an open viewer, select **Plot Configuration** in the **Edit** menu.



Use the radio buttons to select the number of plots you want displayed in your LTI Viewer. For each plot, select a response type from the menus located on the right-hand side of the window.

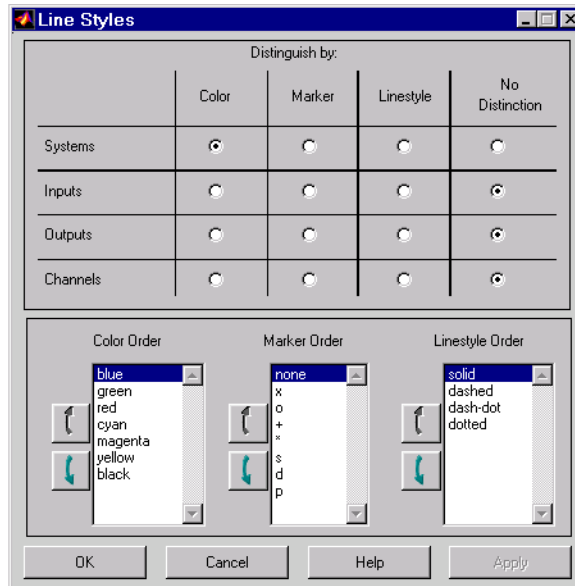
It's possible to configure a single LTI Viewer to contain up to six response plots.



Available response plots include: step, impulse, Bode (magnitude and phase, or magnitude only), Nyquist, Nichols, sigma, pole/zero maps, and I/O pole/zero maps.

Line Styles Editor

Select **Edit-> Line Styles** to open the **Line Styles** editor.



The **Line Styles** editor is particularly useful when you have multiple systems imported. You can use it change line colors, add and rearrange markers, and alter line styles (solid, dashed, and so on).

You can use the **Linestyle Preferences** window to customize the appearance of the response plots by specifying:

- The line property used to distinguish different systems, inputs, or outputs
- The order in which these line properties are applied

Each LTI Viewer has its own **Linestyle Preferences** window.

Setting Preferences

You can use the "Distinguish by" matrix (the top half of the window) to specify the line property that will vary throughout the response plots. You can group multiple plot curves by systems, inputs, outputs, or channels (individual input/output relationships). Note that the Line Styles editor uses radio buttons, which means that you can only assign one property setting for each grouping (system, input, etc.).

Ordering Properties

The **Order** field allows you to change the default property order used when applying the different line properties. You can reorder the colors, markers, and linestyles (e.g., solid or dashed).

To change any of the property orders, click the up or down arrow button to the left of the associated property list to move the selected property up or down in the list.

Analyzing MIMO Models

| In this section... |
|---|
| “Overview of Analyzing MIMO Models” on page 14-20 |
| “Array Selector” on page 14-21 |
| “I/O Grouping for MIMO Models” on page 14-23 |
| “Selecting I/O Pairs” on page 14-24 |

Overview of Analyzing MIMO Models

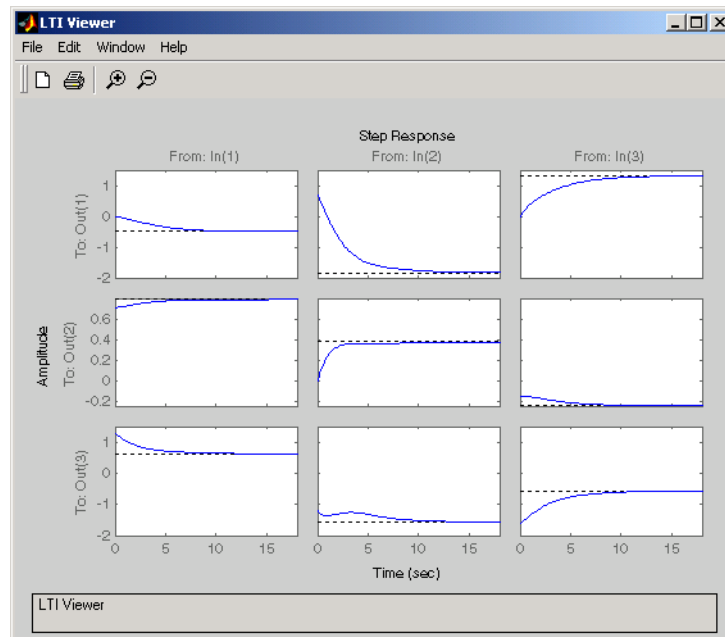
If you import a MIMO system, or an LTI array containing multiple linear models, you can use special features of the right-click menu to group the response plots by input/output (I/O) pairs or select individual plots for display. For example, if you randomly generate a 3-input, 3-output MIMO system,

```
sys_mimo=rss(3,3,3);
```

and open an LTI Viewer,

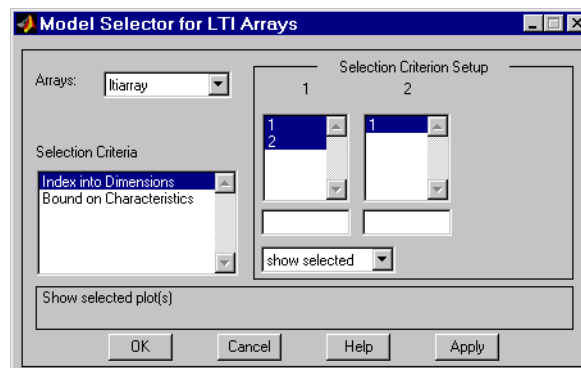
```
ltiview(sys_mimo);
```

the default is an unwrapped set of 9 plots, one from each input to each output.



Array Selector

If you import an LTI array into your LTI Viewer, **Array Selector** appears as an option in the right-click menu. Selecting this option opens the **Model Selector for LTI Arrays**, shown below.



You can use this window to include or exclude models within the LTI array using various criteria. The following subsections discuss the features in turn.

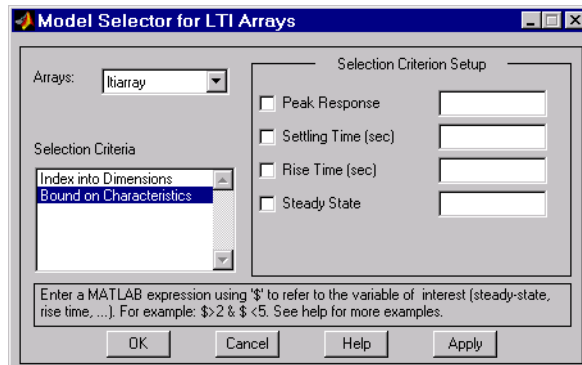
Arrays

Select which LTI array for applying model selection options by using the Arrays pull-down list.

Selection Criteria

There are two selection criteria. The default, **Index into Dimensions**, allows you to include or exclude specified indices of the LTI Array. Select systems from the **Selection Criteria Setup** and specify whether to show or hide the systems using the pull-down menu below the Setup lists.

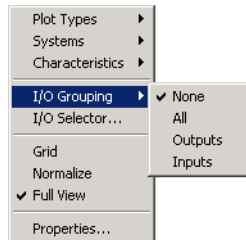
The second criterion is **Bound on Characteristics**. Selecting this options causes the Model Selector to reconfigure. The reconfigured window is shown below.



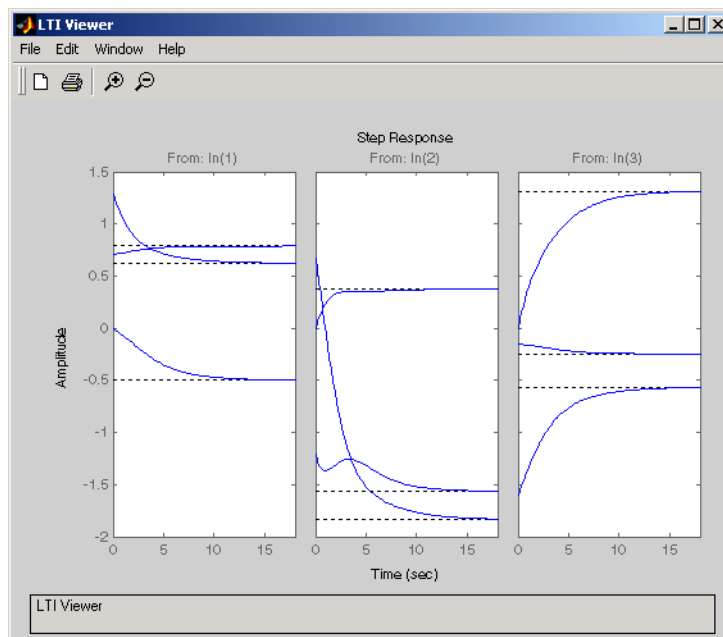
Use this option to select systems for inclusion or exclusion in your LTI Viewer based on their time response characteristics. The panel directly above the buttons describes how to set the inclusion or exclusion criteria based on which selection criteria you select from the reconfigured **Selection Criteria Setup** panel.

I/O Grouping for MIMO Models

You can group this by inputs, by outputs, or both by selecting **I/O Grouping** and then **Inputs**, **Outputs**, or **All**, respectively, from the right-click menu.



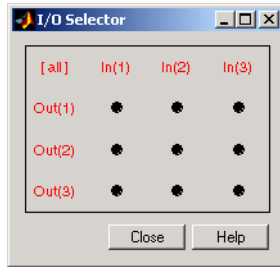
For example, if you select **Outputs**, the LTI Viewer reconfigures into 3 plots, one for each input.



Selecting **None** returns to the default configuration, where all I/O pairs are displayed individually.

Selecting I/O Pairs

Another way to organize MIMO system information is to choose **I/O Selector** from the right-click menu, which opens the **I/O Selector** window.



This window automatically configures to the number of I/O pairs in your MIMO system. You can select:

- Any individual plot (only one at a time) by clicking on a button
- Any row or column by clicking on $Y(*)$ or $U(*)$
- All of the plots by clicking [all]

Using these options, you can inspect individual I/O pairs, or look at particular I/O channels in detail.

Customizing the LTI Viewer

In this section...

“Overview of Customizing the LTI Viewer” on page 14-25

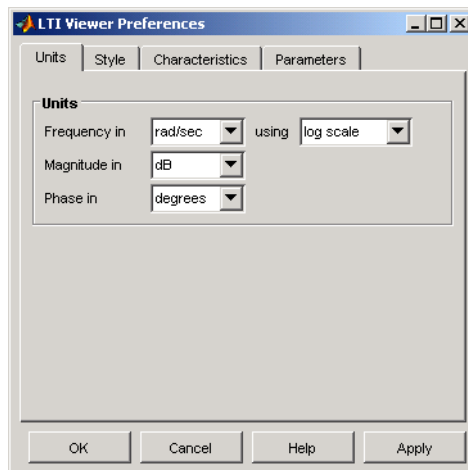
“LTI Viewer Preferences Editor” on page 14-25

Overview of Customizing the LTI Viewer

The LTI Viewer has a tool preferences editor, which allows you to set default characteristics for specific instances of LTI Viewers. If you open a new instance of either, each defaults to the characteristics specified in the **Toolbox Preferences** editor.

LTI Viewer Preferences Editor

Select **Viewer Preferences** in the **Edit** menu of the LTI Viewer to open the **LTI Viewer Preferences** editor. This figure shows the editor open to its first pane.



The **LTI Viewer Preferences** editor contains four panes:

- Units--Convert between various units, including rad/sec and Hertz

- Style--Customize grids, fonts, and colors
- Characteristics--Specify response plot characteristics, such as settling time tolerance
- Parameters--Set time and frequency ranges, stop times, and time step size

If you want to customize the settings for all instances of LTI Viewers, see the Toolbox Preferences editor.

A

- addition of LTI models 5-15
 - scalar 5-16
- adjoint.. *See* pertransposition
- append 5-20 to 5-21
- array selector for LTI Viewer 14-21
- arrays. *See* Model arrays 2-58

C

- classical control 11-2 11-19
- closed loop.. *See* feedback
- concatenation, model
 - horizontal 5-21
 - state-space model order, effects on 5-13
 - vertical 5-21
- connection
 - feedback 11-11
 - parallel 5-16 11-54
 - series 5-17 11-14
- continuous-time 6-2
- conversion, model
 - automatic 1-24
 - between model types 1-24 5-4
 - continuous to discrete (c2d) 5-25
 - discrete to continuous (d2c) 5-25
 - with negative real poles 5-26
 - FRD model, to 1-24
 - resampling 5-37
 - SS model, to 1-24
 - state-space, to 1-25
 - TF model, to 1-24
 - ZPK model, to 1-24
- covariance
 - error 11-56 11-61
- creating model arrays 2-58
- customizing plots 10-2
- customizing subplots 10-18

D

- d2d 5-37
- delays
 - discretization 5-27
 - Pade approximation 2-30
- deletion
 - parts of LTI models 5-13
- denominator
 - specification 1-11
 - value 5-7
- descriptor systems.. *See* state-space models, descriptor
- design
 - classical 11-2 11-19
 - Kalman estimator 11-35 11-49 11-57
 - LQG 11-30
 - regulators 11-30
 - robustness 11-27
 - root locus 11-8 11-23
- desired responses
 - step responses 14-9
- discrete-time models 6-2
 - control design 11-19
 - Kalman estimator 11-50
 - resampling 5-37
 - upsampling 5-37
- discretization 5-25 11-20
 - delay systems 5-27
 - first-order hold 5-27
 - matched poles/zeros 5-35
 - Tustin method 5-32
 - zero-order hold 5-25
- dual.. *See* transposition

E

- error covariance 11-56 11-61
- extraction
 - LTI models, in 5-8

F

feedback 11-11
filtering.. *See* Kalman estimator
first-order hold (FOH) 5-27
 with delays 5-27
FRD (frequency response data) objects 1-14
 conversion to 1-24
 frequencies
 indexing by 5-11
 referencing by 5-11
frequency response 1-15

G

gain margins 11-27
Graphical design window 13-2
Graphical Tuning Preferences Editor 9-9
group. *See* I/O groups 5-12

I

I/O

concatenation 5-20
dimensions 6-2
groups
 referencing models by group name 5-12
names
 conflicts, naming 4-16 5-4
 referencing models by 5-12
 relation 5-8
inheritance 5-3
input
 number of inputs 6-2
InputGroup 4-16 5-4
 conflicts, naming 4-16 5-4
 See also I/O groups
InputName 4-16 5-4
 conflicts, naming 4-16 5-4
 See also I/O names
inversion

model 5-18

K

Kalman
 filtering 11-49
Kalman estimator
 continuous 11-35
 discrete 11-50
 time-varying 11-57

L

LQG (linear quadratic-gaussian) method
 continuous LQ regulator 11-35
 cost function 11-35
 design 11-30 11-45
 LQ-optimal gain 11-35
 regulator 11-30
LTI arrays
 conversion, model.. *See* conversion
LTI models
 addition 5-15
 scalar 5-16
 building 5-20
 characteristics 6-2
 concatenation
 effects on model order 5-13
 horizontal 5-21
 vertical 5-21
 continuous 6-2
 conversion 1-24 5-4 5-26
 continuous/discrete 5-25
 See also conversion, model
 discrete 6-2
 discretization, matched poles/zeros 5-35
 empty 6-2
 functions, analysis 6-4
 I/O group or channel name, referencing
 by 5-12

- interconnection functions 5-20
- inversion 5-18
- model data, accessing 5-6
- modifying 5-8
- multiplication 5-17
- operations 5-1 5-5
 - precedence rules 5-3
 - See also* operations
- proper transfer function 6-2
- resizing 5-13
- subsystem, modifying 5-13
- subtraction 5-16
- type 6-2
- LTI properties
 - I/O groups.. *See* I/O, groups
 - I/O names.. *See* I/O, names
 - inheritance 5-3
 - sample time 4-16 5-4
 - variable property 4-16 5-4
- LTI Viewer
 - array selector 14-21
 - configuring plots 14-16
 - customization 14-25
 - I/O grouping 14-23
 - importing/exporting models 14-12
 - MIMO models 14-20
 - overview 14-2
 - right-click menu 14-4
 - selecting I/O pairs 14-24
 - selecting response types 14-16
 - SISO Design Task 13-2
- LTI Viewer Preferences Editor 9-3

M

- map, I/O 5-8
- margins, gain and phase 11-27
- MIMO 5-21
- model arrays
 - creating model arrays 2-58

- Model arrays 2-58
- model building 5-20
 - feedback connection 11-11
 - parallel connection 5-16 11-54
 - series connection 5-17 11-14
- model dynamics, function list 6-4
- modeling.. *See* model building
- multiplication 5-17
 - scalar 5-18

N

- numerator
 - specification 1-11
 - value 5-7

O

- operations on LTI models
 - addition 5-15
 - append 5-21
 - arithmetic 5-15
 - concatenation 5-13 5-21
 - extracting a subsystem 5-5
 - inversion 5-18
 - multiplication 5-17
 - pertransposition 5-19
 - precedence 5-3
 - resizing 5-13
 - subsystem, extraction 5-8
 - subtraction 5-16
 - transposition 5-18
- output
 - number of outputs 6-2
- OutputGroup
 - group names, conflicts 4-16 5-4
 - See also* I/O, groups 4-16 5-4
- OutputName
 - conflicts, naming 4-16 5-4

P

- Pade approximation (pade) 2-30
- parallel connection 5-16 11-54
- pertransposition 5-19
- phase margins 11-27
- plot customization 10-2
- Plot Tools 10-19
- precedence rules 5-3
- proper transfer function 6-2
- properties
 - sample time 4-16 5-4
 - variable 4-16 5-4
- Property Editor 10-3

R

- realizations 6-7
- regulation 11-30
- resampling 5-37
- response, I/O 5-8
- robustness 11-27
- root locus 11-23
 - design 11-8 11-23
 - See also* Root Locus Design GUI

S

- sample time 4-16 5-4
 - accessing 5-6
 - resampling 5-37
 - upsampling 5-37
- scaling 12-2
- series connection 5-17 11-14
- SISO 6-2
- SISO Design Task in the Control and Estimation Tools Manager 13-2
- SISO Design Tool 13-2
 - customizing plots 10-48
 - root locus right-click menu 13-57

- SS models 5-19
- state 1-13 to 1-14
- state-space models
 - conversion to 1-24
 - See also* conversion 1-24
 - descriptor 1-14 5-6
 - matrices 1-13
 - model data 1-13
 - quick data retrieval 5-6
 - realizations 6-7
 - scaling 12-2
 - specification 1-13
- step responses
 - specifications 14-9
- subplot customization 10-18
- subsystem 5-5 5-8
- subsystem operations on LTI models
 - subsystem, modifying 5-13
- subtraction 5-16

T

- tfdata
 - output, form of 5-6
- time-varying Kalman filter 11-57
- Toolbox Preferences Editor 8-2
- transfer functions
 - conversion to 1-24
 - denominator 1-11
 - MIMO 5-21
 - numerator 1-11
 - quick data retrieval 5-6
 - specification 1-11
 - variable property 4-16 5-4
- transposition 5-18
- triangle approximation 5-27
- Tustin approximation 5-32
 - with frequency prewarping 5-33

V

variable property 4-16 5-4

Z

zero-order hold (ZOH) 5-25 11-20
with delays 5-27

zero-pole-gain (ZPK) models

conversion to 1-24

MIMO 5-21

quick data retrieval 5-6

zpkdata

output, form of 5-6