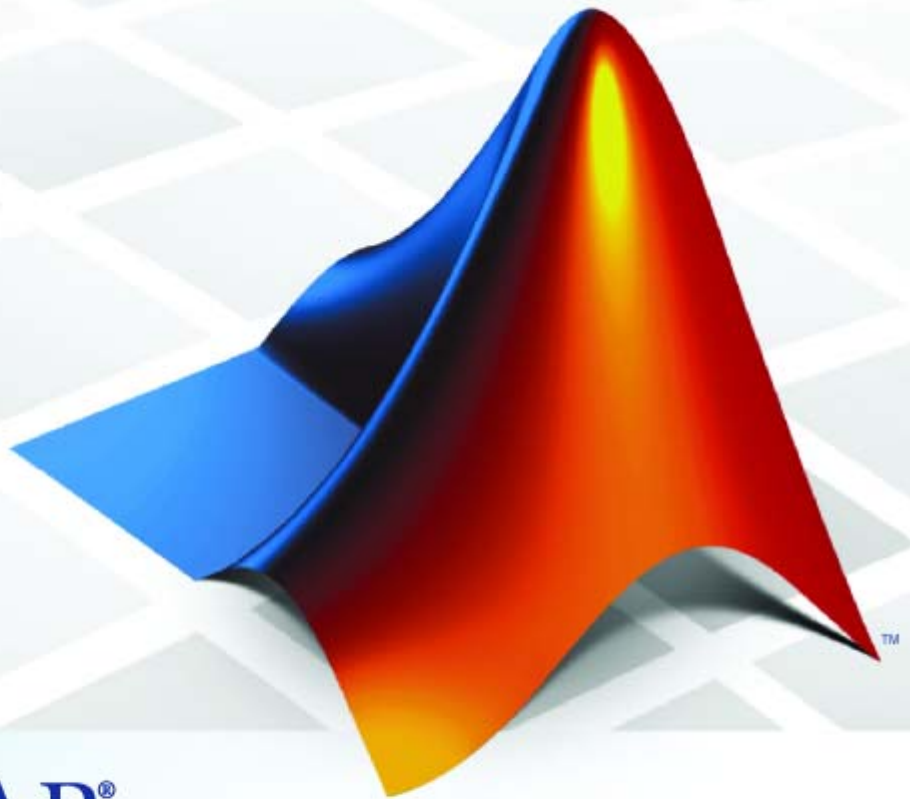


Simulink® Control Design™ 3

User's Guide



MATLAB®
& **SIMULINK®**

How to Contact The MathWorks



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

Web
Newsgroup
Technical Support



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

Product enhancement suggestions
Bug reports
Documentation error reports
Order status, license renewals, passcodes
Sales, pricing, and general information



508-647-7000 (Phone)



508-647-7001 (Fax)



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

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

Simulink® Control Design™ User's Guide

© COPYRIGHT 2004–2009 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

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

Revision History

June 2004	Online only	New for Version 1.0 (Release 14)
October 2004	Online only	Revised for Version 1.1 (Release 14SP1)
March 2005	Online only	Revised for Version 1.2 (Release 14SP2)
September 2005	Online only	Revised for Version 1.3 (Release 14SP3)
March 2006	Online only	Revised for Version 2.0 (Release 2006a)
September 2006	Online only	Revised for Version 2.0.1 (Release 2006b)
March 2007	Online only	Revised for Version 2.1 (Release 2007a)
September 2007	Online only	Revised for Version 2.2 (Release 2007b)
March 2008	Online only	Revised for Version 2.3 (Release 2008a)
October 2008	Online only	Revised for Version 2.4 (Release 2008b)
March 2009	Online only	Revised for Version 2.5 (Release 2009a)
September 2009	Online only	Revised for Version 3.0 (Release 2009b)

Working with Simulink® Control Design Projects

1

Creating or Opening a Simulink Model	1-2
Details of Possible Models	1-2
Example Model: The Magnetic Ball System	1-2
Beginning a Project	1-7
Creating a Simulink® Control Design Project	1-7
Creating an Operating Points Task	1-8
Creating a Linearization Task	1-8
Creating a Simulink Compensator Design Task	1-10
Saving Projects	1-11
Opening Previously Saved Projects	1-12
Exporting Results	1-13
Exporting Linearization Results	1-13
Exporting Compensator Designs	1-14
Exporting Operating Points	1-15
Exporting and Restoring Linearization I/O Settings	1-15

Operating Point Analysis Using the GUI

2

What Are Operating Points?	2-2
Definition of an Operating Point	2-2
Equilibrium Operating Points	2-2
Simulink Model Operating Points	2-3

Why Are Operating Points Important?	2-6
Impact of Operating Points	2-6
Choosing an Operating Point for Accurate Linearization ..	2-7
Ways to Create Operating Points	2-9
Creating Operating Points	2-10
Simulink® Control Design Default Operating Point	2-10
Computing Operating Points from Specifications	2-10
Specifying Operating Points from Known Values	2-15
Extracting Operating Points From Simulation	2-18
Computing Equilibrium Operating Points	2-22
Working with Operating Points	2-23
Copying Operating Points	2-23
Exporting Operating Points	2-24
Saving Operating Points	2-25
Importing Operating Points	2-25
Importing Initial Values	2-26
Constraining Outputs	2-27
Changing Optimization Settings	2-27
Recommendations for Computing Operating Points ..	2-31
How to Create Accurate Operating Points	2-31
Impact of Blocks on the Simulink Model Operating Point	2-31
Computing Operating Points for SimMechanics Models ..	2-36
Choosing Initial Values for Computing Operating Points	2-37
Computing Operating Points for Blocks with Special Behavior	2-38

Operating Point Analysis Using the Command Line

3

Overview	3-2
-----------------------	------------

Example: Water-Tank System	3-3
Water-Tank System	3-3
Model Equations	3-4
Creating or Opening a Simulink Model	3-5
Computing Operating Points from Specifications	3-7
Workflow for Computing Operating Points from Specifications	3-7
Creating an Operating Point Specification Object	3-7
Configuring the Operating Point Specification Object	3-8
Computing the Complete Operating Point	3-10
Alternative Method for Specifying Initial Guesses	3-11
Adding Output Constraints to Specifications	3-12
Specifying Completely Known Operating Points	3-14
Workflow for Specifying Completely Known Operating Points	3-14
Creating an Operating Point Object	3-14
Changing Operating Point Values	3-15
Extracting Values from Simulation	3-16
Using Structures and Vectors of Operating Point Values	3-17

Exact Linearization Using the GUI

4

What Is Linearization?	4-2
Linearization Background	4-2
Analytic Representations of Linear Models	4-3
Ways to Linearize Models	4-7
Steps for Linearizing Models Using the GUI	4-8

Choosing Linearization Settings and Algorithms	4-9
How to Choose Linearization Settings and Algorithms ...	4-9
Options for Linearization Algorithm Method	4-11
Block-by-Block Analytic Linearization	4-12
Numerical-Perturbation Linearization	4-24
Changing State Ordering in the Linearized Model	4-34
Configuring the Linearization of Specific Blocks and	
Subsystems	4-36
Ways to Configure Blocks and Subsystems For	
Linearization	4-36
Controlling the Analytic Linearization of Individual	
Blocks	4-36
Specifying the Linearization of Blocks and Subsystems ...	4-37
Controlling the Block Perturbation Linearization of	
Individual Blocks	4-40
Selecting Inputs and Outputs for the Linearized	
Model	4-45
What are Linearization Points?	4-45
Inserting Linearization Points	4-46
Removing Linearization Points	4-48
Performing Open-Loop Analysis	4-49
Inspecting Analysis I/Os	4-52
Linearizing a Block	4-55
Linearizing the Model	4-57
Linearizing at an Operating Point	4-57
Creating Other Types of Linear Models	4-64
Linearizing Discrete-Time and Multirate Models	4-65
Viewing Linearization Results	4-66
Using the LTI Viewer	4-66
Displaying Linearization Results	4-66
Highlighting Blocks in the Linearization	4-68
Inspecting the Linearization Results Block by Block	4-69
Comparing the Results of Multiple Linearizations	4-71
Validating Exact Linearization Results	4-76
Ways to Validate Exact Linearization Results	4-76

Creating Input Signals for Validation	4-76
Frequency-Domain Validation	4-77
Time-Domain Validation	4-80
Troubleshooting Exact Linearization Results	4-82
Diagnosing Blocks	4-82
Troubleshooting Your Model at the Subsystem and Block Level	4-86
Troubleshooting Linearization Settings	4-87
Troubleshooting Models with Events-Based Subsystems ..	4-88
Troubleshooting Your Operating Point	4-88

Exact Linearization Using the Command Line

5

Steps for Linearizing Models Using the Command Line	5-2
Configuring the Linearization for Specific Blocks and Subsystems	5-3
Selecting Inputs and Outputs for the Linearized Model	5-4
Workflow for Selecting Inputs and Outputs	5-4
Choosing and Storing Linearization Points	5-4
Extracting Linearization Points from a Model	5-7
Editing an I/O Object	5-8
Open-Loop Analysis Using Functions	5-10
Linearizing the Model Using Functions	5-11
Linearizing the Model	5-11
Linearizing Discrete-Time and Multirate Models	5-13
Computing Multiple Linearizations for Large Models	5-13
Analyzing the Results Using Functions	5-14
Options for Analyzing the Results	5-14
Using the LTI Viewer	5-14
Saving Your Work	5-16

Restoring Linearization I/O Settings	5-16
--	------

Frequency Response Estimation of Simulink Models

6

About Frequency Response Estimation	6-2
What Is a Frequency Response Model?	6-2
Overview of Frequency Response Estimation in Simulink	6-3
Creating Input Signals for Estimation	6-7
Supported Input Signals	6-7
Creating the Sinestream Input Signal	6-7
Creating the Chirp Input Signal	6-14
Estimating Frequency Response	6-17
Analyzing Estimated Frequency Response	6-20
Opening the Simulation Results Viewer	6-20
Interpreting the Frequency Response Analysis Plots	6-21
Analyzing the Simulated Output and FFT at Specific Frequencies	6-23
Annotating Data on Plots Using Data Tips	6-24
Displaying Frequency Response of MIMO Systems	6-25
Troubleshooting a Frequency Response Estimation ..	6-26
Time Response Not at Steady State	6-26
FFT Contains Large Harmonics At Frequencies Other Than The Input Signal Frequency	6-32
Time Response Grows Without Bound	6-34
Time Response Is Discontinuous or Zero	6-35
Estimating Models With Noise	6-39
Generating Data for Modeling Noise	6-39
Example—Estimating a Model With Noise Using Signal Processing Toolbox	6-40

Example—Estimating State-Space Model Using System Identification Toolbox	6-41
--	------

Designing Compensators

7

Compensator Design Using Simulink	7-2
Automatic PID Tuning	7-4
About Automatic PID Tuning	7-4
Tuning One-Degree-of-Freedom PID Controllers	7-4
Tuning Two Degree-of-Freedom PID Controllers	7-14
Design and Analysis of Control Systems	7-16
Compensator Design Process Overview	7-16
Beginning a Compensator Design Task	7-16
Selecting Blocks to Tune	7-18
Selecting Closed-Loop Responses to Design	7-21
Selecting an Operating Point	7-23
Creating a SISO Design Task	7-26
Completing the Design	7-37

Function Reference

8

Linearization Analysis I/Os	8-1
Operating Points	8-2
Linearization	8-3
Frequency Response Estimation	8-3

Functions — Alphabetical List

9

Block Reference

10

Examples

A

Linearization Example Using the Graphical Interface	A-2
Linearization Example Using Functions	A-2
Frequency Response Estimation	A-2

Index

Working with Simulink Control Design Projects

- “Creating or Opening a Simulink Model” on page 1-2
- “Beginning a Project” on page 1-7
- “Saving Projects” on page 1-11
- “Opening Previously Saved Projects” on page 1-12
- “Exporting Results” on page 1-13

Creating or Opening a Simulink Model

In this section...
“Details of Possible Models” on page 1-2
“Example Model: The Magnetic Ball System” on page 1-2

Details of Possible Models

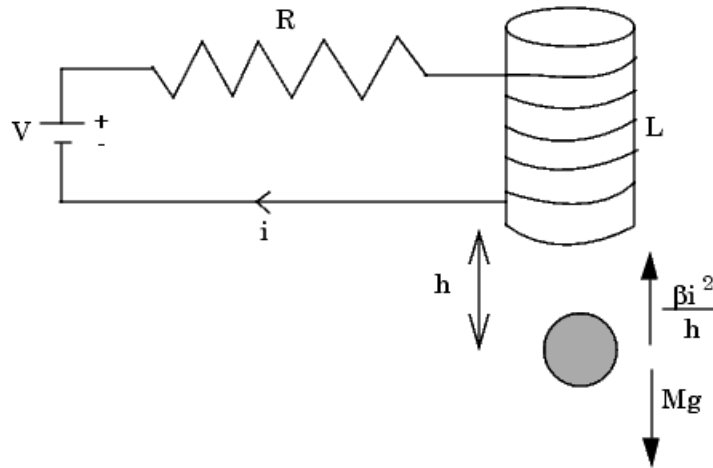
The first step in the linearization or compensator design process is to create or open a Simulink® model of your system. The model can have any number of inputs and outputs (including none), and any number of states. The model can include user-defined blocks or S-functions. Your model can involve multiple compensators in addition to the plant, multiple feedback loops, and any number of subsystems.

Example Model: The Magnetic Ball System

This section introduces an example model, the magnetic ball system, that the remaining sections and chapters use to illustrate the process of linearizing a model or designing a compensator.

Magnetic Ball System

The electronic circuit in the following figure consists of a voltage source, a resistor, and an inductor in the form of a tightly wound coil. An iron ball beneath the inductor experiences a gravitational force as well as an induced magnetic force (from the inductor) that opposes the gravitational force.



Model Equations

A differential equation for the force balance on the ball is given by

$$M \frac{d^2 h}{dt^2} = Mg - \frac{\beta i^2}{h}$$

where M is the mass of the ball, h is the height (position) of the ball, g is the acceleration due to gravity, i is the current, and β is a constant related to the magnetic force experienced by the ball. This equation describes the height, h , of the ball due to the unbalanced forces acting upon it.

The current in the circuit also varies with time and is given by the following differential equation

$$L \frac{di}{dt} = V - iR$$

where L is the inductance of the coil, V is the voltage in the circuit, and R is the resistance of the circuit.

The system of equations has three states:

$$h, \frac{dh}{dt}, i$$

The system also has one input (V), and one output (h). It is a nonlinear system due to the term in the equation involving the square of i and the inverse of h .

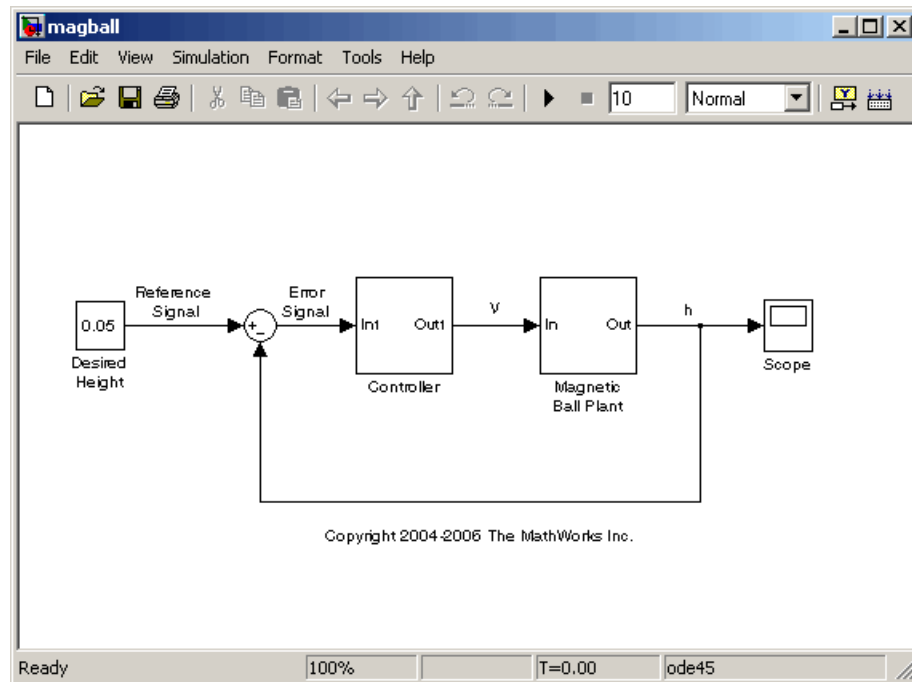
Due to its nonlinearity, you cannot analyze this system using methods for linear-time-invariant (LTI) systems such as step response plots, bode diagrams, and root-locus plots. However, you can linearize the model using the Simulink® Control Design™ software to approximate the nonlinear system as an LTI system. Linearization also occurs automatically when designing a compensator. This linearized system can then use the LTI Viewer for display and analysis and the SISO Design Tool for compensator design. Refer to for a discussion of the uses of linearized models and “What Is Linearization?” on page 4-2 for a discussion of the linearization process.

Opening the Model

To open the model for the magnetic ball example, type

```
magball
```

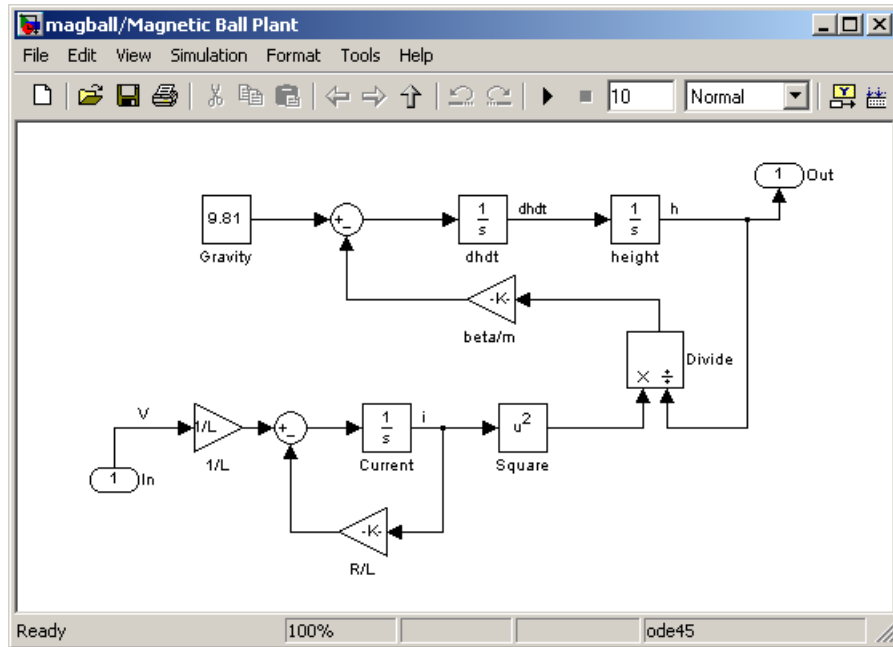
at the MATLAB® prompt. The magnetic ball system opens in the Simulink model viewer as shown in this figure.



The magball model consists of

- The magnetic ball system itself, within the subsystem labeled Magnetic Ball Plant.
- A Controller subsystem that controls the height of the ball by balancing the forces acting on it.
- A reference signal that sets the desired height of the ball.
- A Scope block that displays the height of the ball as a function of time.

Double-click a block to view its contents. The Controller block contains a zero-pole-gain model. The Magnetic Ball Plant block is shown in this figure.



The input to the Magnetic Ball Plant system, which is also the output of the Controller subsystem, is the voltage, V . The output is the height of the ball, h . The system contains three states within the three integrators: height, dh/dt , and Current.

Values of the parameters are given as $M=0.1$ kg, $g=9.81$ m/s², $R=2$ Ohm, $L=0.02$ H, and $\beta=0.001$.

Beginning a Project

In this section...

“Creating a Simulink® Control Design Project” on page 1-7

“Creating an Operating Points Task” on page 1-8

“Creating a Linearization Task” on page 1-8

“Creating a Simulink Compensator Design Task” on page 1-10

Creating a Simulink Control Design Project

With Simulink Control Design software you can create operating points, linearize, and design compensators for Simulink models. You perform all these tasks in a graphical environment called the Control and Estimation Tools Manager. The tasks are contained within a Control and Estimation Tools Manager *project*. Each project is associated with a single Simulink model and in addition to Simulink Control Design tasks, it can include tasks from other products such as the Simulink® Design Optimization™ product, the Control System Toolbox™ product, and the Model Predictive Control Toolbox™ product.

To open a new Simulink Control Design project:

- 1** Select **Start > Simulink > Simulink Control Design > Linearization Task** or select **Start > Simulink > Simulink Control Design > Simulink Compensator Design Task**.
- 2** Enter a project name, select a model to analyze, and choose the tasks you want to perform. Click **OK** to close the dialog box and open the new project.

Alternatively, you can create a new project from a Simulink model window. Within the model window select **Tools > Control Design > Linear Analysis** to open a project containing a linearization task, or select **Tools > Control Design > Control Design** to open a project containing a compensator design task.

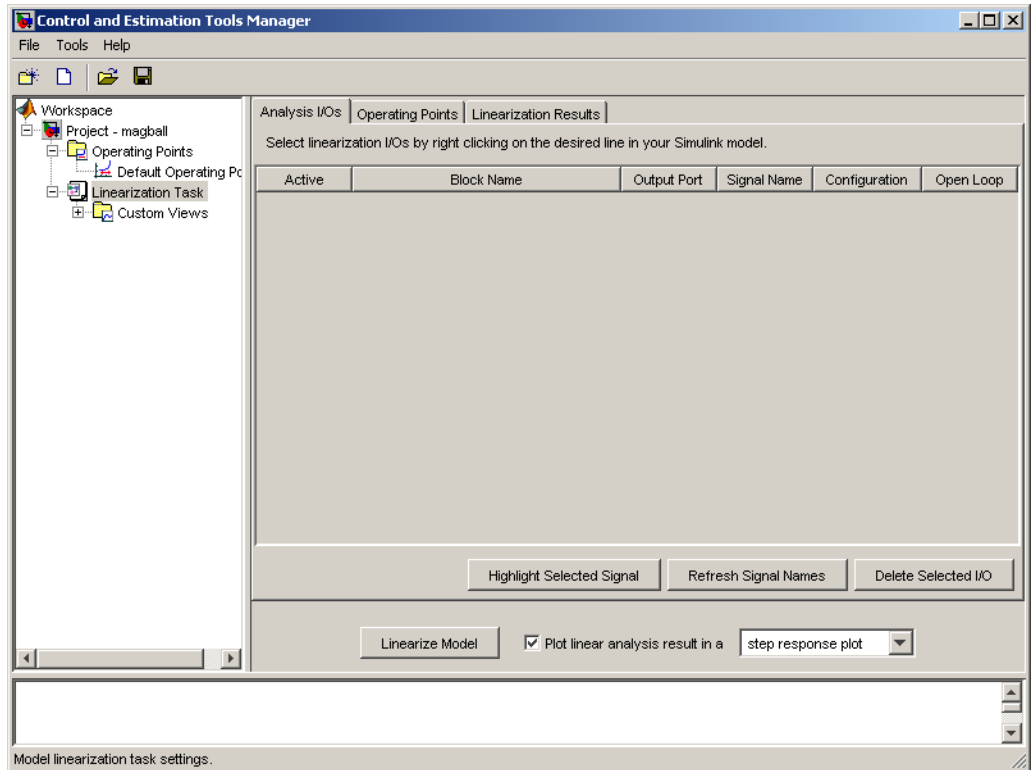
Creating an Operating Points Task

An **Operating Points** node in the Control and Estimation Tools Manager is automatically created when you begin a **Linearization Task** or a **Simulink Compensator Design Task**. You can use the **Operating Points** node to create operating points for a Simulink model.

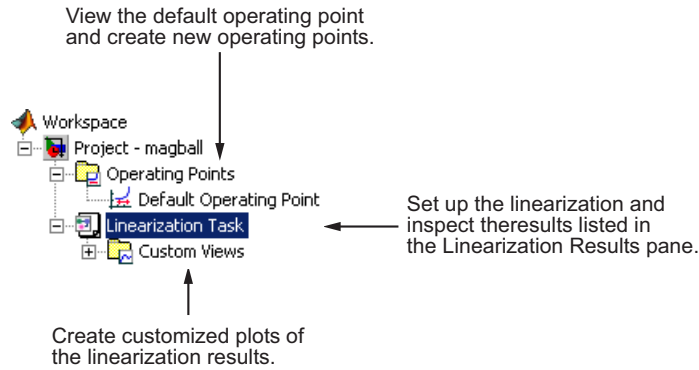
Creating a Linearization Task

To create a linearization task in the Control and Estimation Tools Manager, use one of the methods in “Creating a Simulink® Control Design Project” on page 1-7 to open a new project for your model, and choose a linearization task for this project. To add a linearization task to an existing project, select **File > New > Task** in the Control and Estimation Tools Manager window to open the New Task dialog box. Select **Linearization Task** and the project that you want to open the task within, and then click **OK**.

To open a new project within the Control and Estimation Tools Manager for linearization of the magball model, select **Tools > Control Design > Linear Analysis** from the magball window. The Control and Estimation Tools Manager opens and creates a new linearization task, as shown in the following figure.



The left pane of the Control and Estimation Tools Manager shows the project tree, which contains all your current projects. At this stage you should have just one project, **Project - magball**. Select a node within the tree to display its contents in the pane on the right.



- For information on the **Operating Points** node or the **Operating Points** pane within the **Linearization Task** node, refer to “Creating Operating Points” on page 2-10.
- For information on the **Analysis I/Os** pane within the **Linearization Task** node, refer to “Selecting Inputs and Outputs for the Linearized Model” on page 4-45.
- For information on the **Linearization Results** pane within the **Linearization Task** node and inspecting linearization results, refer to “Viewing Linearization Results” on page 4-66.
- For information on **Custom Views**, refer to “Viewing Linearization Results” on page 4-66.

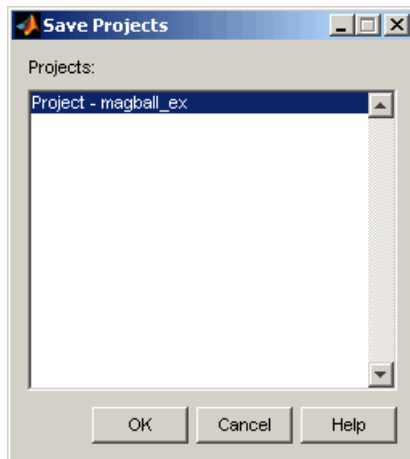
Creating a Simulink Compensator Design Task

To create a Simulink Compensator Design Task in the Control and Estimation Tools Manager, use one of the methods in “Beginning a Project” on page 1-7 to open a new project for your model, and choose a compensator design task for this project. To add a compensator design task to an existing project, select **File > New > Task** in the Control and Estimation Tools Manager window to open the New Task dialog box. Select **Simulink Compensator Design Task** and the project that you want to open the task within, and then click **OK**.

Saving Projects

A Control and Estimation Tools Manager project can consist of multiple tasks such as linearization, compensator design, and operating points tasks as well as Simulink Design Optimization tasks and Model Predictive Control Toolbox tasks. Each task contains data, objects, and results for the analysis of a particular model.

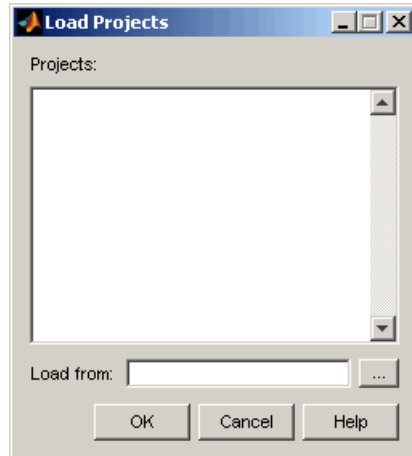
- 1 To save a project as a MAT-file, select **File > Save** from the Control and Estimation Tools Manager window.



- 2 In the Save Projects dialog box, select one or more projects you want to save. You can save multiple projects in one file. Click **OK**, and browse to the folder where you want to save the project. Enter the project name, and click **Save**.

Opening Previously Saved Projects

- 1 To open previously saved projects, select **File > Load** from the Control and Estimation Tools Manager window. This opens the Load Projects dialog box.



- 2 Choose a project-file by either browsing for the directory and file, or typing the full path and filename in the **Load from** field. Project files are always MAT-files. After you specify the file, the projects contained in this file appear in the list. Select one or more projects in the list, and then click **OK**. When a file contains multiple projects, you can choose to load them all or just a few.

Exporting Results

In this section...

“Exporting Linearization Results” on page 1-13

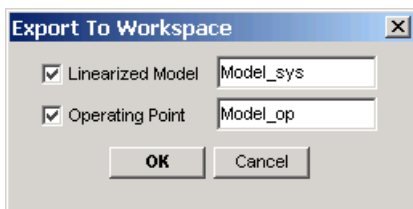
“Exporting Compensator Designs” on page 1-14

“Exporting Operating Points” on page 1-15

“Exporting and Restoring Linearization I/O Settings” on page 1-15

Exporting Linearization Results

To export linearization results and the corresponding operating points, right-click the results node, **Model**, under the **Linearization Task** node and select **Export** from the menu. In the Export To Workspace dialog box, choose new names for the linearized model and operating point, or accept the defaults, and then click **OK**.



The MATLAB workspace now contains two new objects, `Model_op` and `Model_sys`. To see this, type

```
who
```

at the MATLAB prompt. This returns

```
Your variables are:
```

```
L           Model_sys  beta      m
Model_op    R           g
```

Alternatively, you can export the results to the MATLAB workspace by selecting **File > Export** from the LTI Viewer window or by clicking the

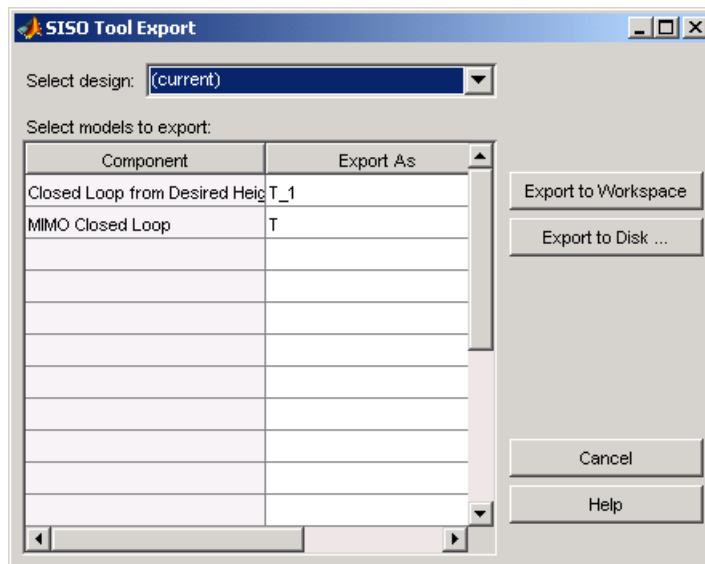
Export to Workspace button at the bottom of the **Linearization Summary** pane within the **Model** node.

By right-clicking the results node, **Model**, you can also delete results.

Exporting Compensator Designs

To export a compensator design to the MATLAB workspace:

- 1 Select **File > Export** from the SISO Design Tool window.
- 2 In the SISO Tool Export dialog box, use the **Select design** list to choose the design you want to export.
- 3 In the list, select the compensators to export, and then click either **Export to Workspace** or **Export to Disk**.



Exporting Operating Points

After creating operating points, you can use the Export To Workspace dialog box to export them for use outside of the Control and Estimation Tools Manager. You can use the exported operating point to perform analysis at the MATLAB command line or to initialize a model for simulation.

- 1** Under **Select destination workspace**, select either
 - **Base workspace** to export the operating point to the MATLAB workspace where you can use it with Simulink Control Design command-line functions
 - **Model workspace** to export the operating point to the Model workspace where you can save it with the model for future use.
- 2** Enter a name for the exported operating point.
- 3** Select **Use the operating point to initialize model** when you want to use the operating point values as initial conditions for the states and inputs in the model. The initial values are automatically set in the **Data Import/Export** pane of the Configuration Parameters dialog box. Simulink software uses these initial conditions when simulating the model.

Exporting and Restoring Linearization I/O Settings

To export linearization I/O settings to the MATLAB workspace, use the `getlinio` function. You can save these settings using the `save` function. Use them in a later session by reloading them with the `load` function. Upload them to the model diagram with the `setlinio` function.

For more information, see the function reference pages for `getlinio` and `setlinio`.

Operating Point Analysis Using the GUI

- “What Are Operating Points?” on page 2-2
- “Why Are Operating Points Important?” on page 2-6
- “Ways to Create Operating Points” on page 2-9
- “Creating Operating Points” on page 2-10
- “Working with Operating Points” on page 2-23
- “Recommendations for Computing Operating Points” on page 2-31

What Are Operating Points?

In this section...
“Definition of an Operating Point” on page 2-2
“Equilibrium Operating Points” on page 2-2
“Simulink Model Operating Points” on page 2-3

Definition of an Operating Point

The *operating point* of a dynamic system defines its overall *state* at a given time. For example, in a model of a car engine, variables such as engine speed, throttle angle, engine temperature, and surrounding atmospheric conditions typically describe the operating point. It is important to specify the operating point accurately because it affects the system’s behavior. For example, the behavior of a car engine can vary greatly when it operates at high or low elevations.

Equilibrium Operating Points

An *equilibrium operating point* remains steady and constant with time; all states in the model are at equilibrium. It is also known as a *steady state* or *trimmed operating point*. For example, a car operating on cruise control on a flat road maintains a constant speed. Its operating point is steady, or at equilibrium.

A hanging pendulum provides an example of a stable equilibrium operating point. When the pendulum hangs straight down, its position does not change with time because it is at an equilibrium position. When its position deviates slightly from this position, it always returns to the equilibrium; small changes in the operating point do not cause the system to leave the region of good approximation around the equilibrium value.

A pendulum that points upward provides an example of an unstable equilibrium operating point. As long as the pendulum points *exactly* upward, it is steady at this equilibrium state. However, when the pendulum deviates slightly from this state, it swings downward and the operating point leaves the region around the equilibrium value.

Simulink Model Operating Points

This section includes the following topics:

- “Operating Point Versus Simulink Full-Model Operating Point” on page 2-3
- “Example of a Simulink Model Operating Point” on page 2-4

Operating Point Versus Simulink Full-Model Operating Point

The Simulink full model operating point includes information from all of the blocks in a Simulink model. When you use the Simulink Control Design GUI or the MATLAB command line to create operating points for a model, you are actually creating an *operating point object* (what is referred to as the operating point). The operating point is a subset of the Simulink full-model operating point.

The following table shows the different types of blocks that make up a full-model operating point and which types are included in the operating point.

Makeup of Simulink Full-Model Operating Point

Block Types	Example of Blocks	Included in Operating Point?
Blocks with double value states and root level inport blocks with double data type	Integrator, State Space, Transfer Function, Inport	Yes
Root level inport blocks with nondouble or complex data type	Inport	No
Blocks with internal state representation that impacts block outputs	Backlash, Memory, Stateflow	No
Source blocks with outputs specified by block dialog parameters	Constant, Step	No

The operating point includes only the block information most commonly redefined by users. This information provides you with three places to make changes to the parameters of these common blocks in your model:

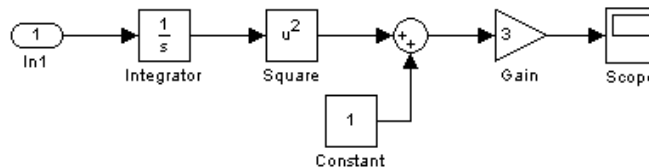
- Directly in the Simulink Control Design Control and Estimation Tools Manager GUI
- At the MATLAB command line
- In the Simulink model

Note If you want to make changes to any block not included in the operating point, you must make the changes directly in the Simulink model.

Example of a Simulink Model Operating Point

The following figure shows a simple Simulink model that has one block with state (the integrator block) and therefore one state, x_1 . This state depends on the initial conditions set in the integrator block. The value of this state and the input from the inport block define the operating point in the Simulink Control Design software.

The square block output is derived from just the initial condition of the integrator block while the gain block has its output derived from both the initial condition of the integrator block and output of the constant block. The derivative dx/dt of the integrator block state is defined by the output level of the root inport block.



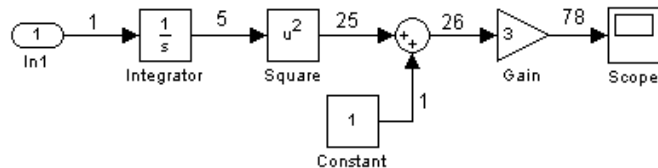
The state, x_1 , defines the signal levels at the input and output of every block in the model by propagating through the blocks in the model as follows.

- 1 The integrator block initial condition of $x_0 = 5$ sets the state $x_1 = 5$.

- 2 The state, x_1 , propagates in the direction of the arrows through the blocks in the model, defining input and output signals on each block, as described in the following table.

Block	Input Signal Level	Block Operation	Output Signal Level
Square	5	squares	25
Sum	25 from square, 1 from constant	sums	26
Gain	26	multiplies by 3	78

The following figure shows these input and output signal levels for each block.



Why Are Operating Points Important?

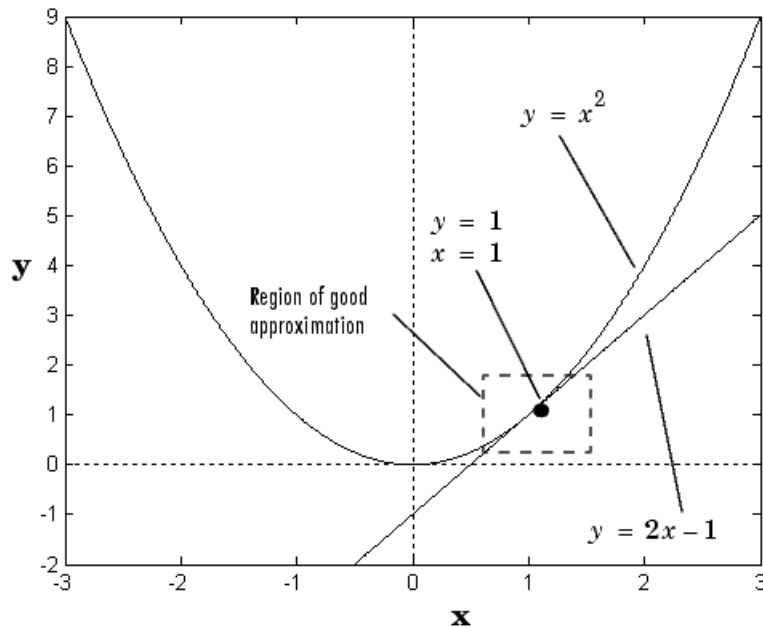
In this section...
“Impact of Operating Points” on page 2-6
“Choosing an Operating Point for Accurate Linearization” on page 2-7

Impact of Operating Points

A linearized model is an approximation that is valid in a region around the operating point of the system where the linearization took place. Near the operating point the approximation is good, while far away it may be poor. A linearized model of a car being operated at 3000 ft. is very accurate at elevations close to 3000 ft. but less accurate as the car travels higher or lower.

Example of a Linear Approximation of a Nonlinear Function

The following figure shows a nonlinear function, $y = x^2$, and a linear function, $y = 2x - 1$. The linear function is an approximation to the nonlinear function about the operating point $x=1, y=1$. Near this operating point, the approximation is good. Away from this operating point, the approximation is poor. The precise boundaries of this region are often somewhat arbitrary. The following figure shows a possible region of good approximation for the linearization of $y = x^2$.



Choosing an Operating Point for Accurate Linearization

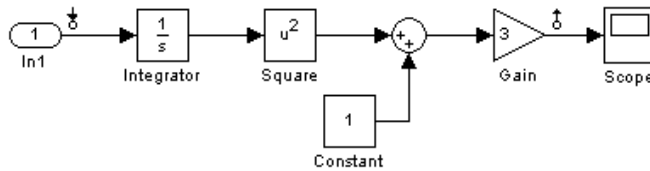
Your choice of operating point is important when you:

- Linearize a Simulink model. The choice of operating point determines the accuracy of the linear approximation.
- Designing compensators with Simulink Control Design software. A Compensator Design Task uses linearization when analyzing a Simulink model.

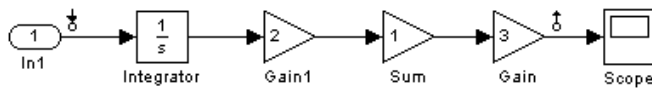
Tip Choose an operating point that is very close to the expected operating values of the system. One option is to use an equilibrium operating point, described in “Equilibrium Operating Points” on page 2-2.

Example of Linearization Results About Two Different Operating Points

A model can have two entirely different linearizations when the linearization is performed about different operating points. The following model can be linearized using the Simulink Control Design software.



The linearization result for this model is shown in the following figure.



When this linearization is performed about two different operating points, two different linearization results occur as shown in the following table.

Operating Point	Linearization Result
Initial Condition = 5, State x1 = 5	30/s
Initial Condition = 0, State x1 = 0	0

Note The operating point consists of values for *all* the states in the model, although only those states between the linearization points are linearized. The whole model contributes to the operating point values of the states, inputs, and outputs of the portion of the model you are linearizing.

Ways to Create Operating Points

You can compute operating points using any technique either:

- Interactively in the GUI
- Programmatically at the command line using MATLAB code

Tip You can automatically generate MATLAB code from your GUI configuration.

Use the following table to choose a technique for creating operating points.

When you know...	Compute operating points from...
Some values and constraints for inputs and states	Specifications <ul style="list-style-type: none"> • GUI • Command line
Exact values for all inputs and states	Known Values <ul style="list-style-type: none"> • GUI • Command line
The time or event in your model to extract an operating point during simulation	Simulation <ul style="list-style-type: none"> • GUI • Command line

Creating Operating Points

In this section...
“Simulink® Control Design Default Operating Point” on page 2-10
“Computing Operating Points from Specifications” on page 2-10
“Specifying Operating Points from Known Values” on page 2-15
“Extracting Operating Points From Simulation” on page 2-18
“Computing Equilibrium Operating Points” on page 2-22

Simulink Control Design Default Operating Point

A default operating point is automatically created and stored in a node in the project tree every time you open the Control and Estimation Tools Manager unless you indicate otherwise. The default operating point consists of a snapshot of initial condition variables in the Simulink model. Initial condition variables are a special set of variables that can be set in the Simulink model. These variable include initial conditions for blocks with state, such as the integrator or state space blocks, and output values of root level inport blocks. For more information on setting these variables, see “Importing and Exporting Data” in the Simulink User’s Guide.

By clicking on the **Default Operating Point** node, you can view these values and modify them for use in a linearization. The output levels for blocks like a constant block remain defined in the model.

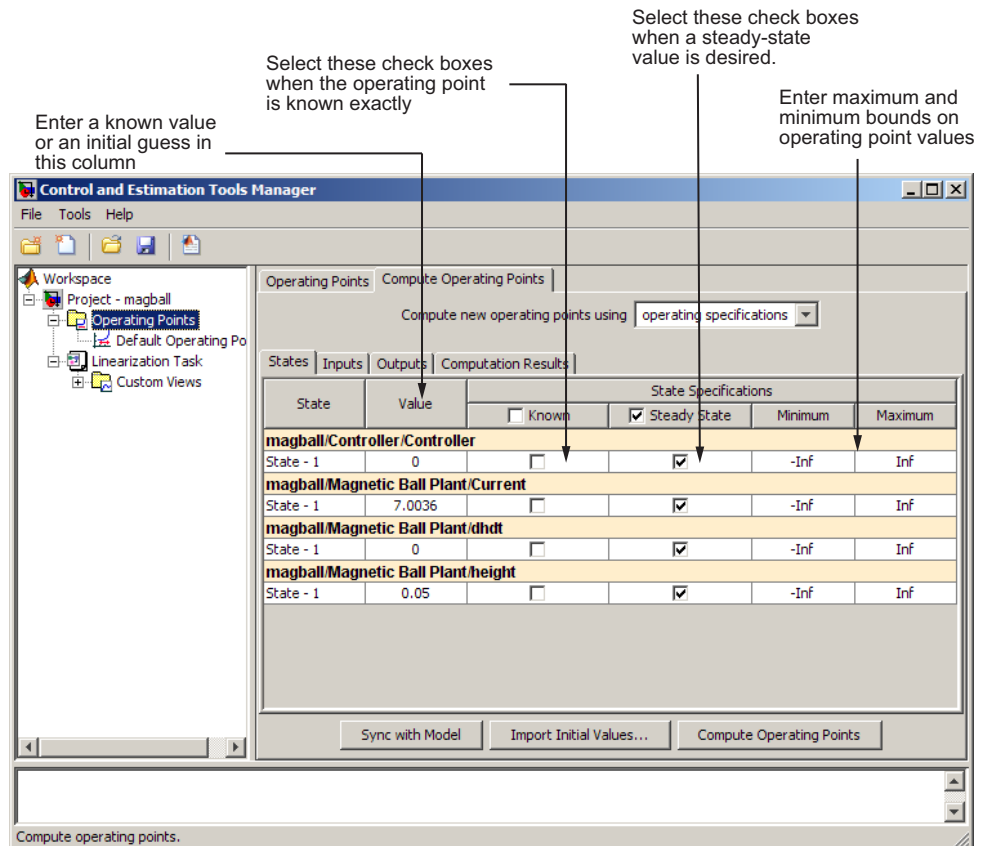
Computing Operating Points from Specifications

You can specify target values or constraints on a subset of the model’s inputs, outputs, and states. The software uses numerical optimization methods to determine the full operating point based on this partial specification.

This section continues the magball example from “Creating a Linearization Task” on page 1-8. At this stage in the example, a linearization task has already been created for the model.

You compute the operating point from specifications when you only know partial or implicit information. Typical operating point specifications search for steady state or equilibrium operating points.

- 1 Create a new operating point by either:
 - Selecting the **Operating Points** node and then clicking the **Compute Operating Points** tab
 - Clicking the **New Operating Point** button on the **Operating Points** pane of the **Linearization Task** node
- 2 From the **Compute new operating points using** list, select operating specifications. The Control and Estimation Tools Manager window should now resemble the following figure.



- Enter operating point specifications in the table, such as any known values or constraints on signal values. Switch between states, inputs, and outputs using the tabs on the left.

A suitable set of specifications for the magball model is shown in the following figure. This model does not contain any root-level input or output ports and, as a result, the **Inputs** and **Outputs** panes are empty. You can still constrain the output signal of any block by adding an **Output Constraint** linearization point to the model. See “Constraining Outputs” on page 2-27 for information about adding an output constraint to the specifications for an operating point.

The height of the ball is known exactly (same as the reference signal height).

The ball height is 0.05 (same as the reference signal value). For a steady ball, use initial guess of 0 for dhdt. For the remaining states, choose arbitrary initial guesses.

A steady-state operating point is desired.

Use a minimum value of zero for the current (by convention current is positive when flowing clockwise around the circuit).

State	Value	State Specifications			
		<input type="checkbox"/> Known	<input checked="" type="checkbox"/> Steady State	Minimum	Maximum
magball/Controller/Controller					
State - 1	0	<input type="checkbox"/>	<input checked="" type="checkbox"/>	-Inf	Inf
magball/Magnetic Ball Plant/Current					
State - 1	7.0036	<input type="checkbox"/>	<input checked="" type="checkbox"/>	0	Inf
magball/Magnetic Ball Plant/dhdt					
State - 1	0	<input type="checkbox"/>	<input checked="" type="checkbox"/>	-Inf	Inf
magball/Magnetic Ball Plant/height					
State - 1	0.05	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	-Inf	Inf

Compute operating points.

When you add states, inputs, or outputs to the model, or remove them from the model, click the **Sync with Model** button to update the operating point table to reflect these changes.

- 4 Click **Compute Operating Points**. The Simulink Control Design software finds an operating point that closely matches the specifications and adds the new operating point, labeled **Operating Point**, to the **Operating Points** node. Some specifications, even values specified as **Known**, may not be met exactly. Select the operating point in the project tree to view its contents and assess the results.

Actual dx values are small, because a steady-state value was found.


State	Desired Value	Actual Value	Desired dx	Actual dx
magball/Controller/Controller				
State - 1	[-Inf , Inf]	-14.0071	0	0
magball/Magnetic Ball Plant/Current				
State - 1	[0 , Inf]	7.0036	0	2.9672e-011
magball/Magnetic Ball Plant/dhdt				
State - 1	[-Inf , Inf]	0	0	-1.7453e-010
magball/Magnetic Ball Plant/height				
State - 1	0.05	0.05	0	0

- Computing operating point for the model: magball.
 - An operating point called "Operating Point" has been added to the node Operating Points.

Operating points for a model.

For information on options that you can set when finding operating points from partial specifications, see “Changing Optimization Settings” on page 2-27.

Note Inputs and outputs for the operating point are not the same as the linearization input and output points, or analysis I/Os, used to define the inputs and outputs of the linearized model. Instead, operating-point inputs and outputs define the full operating point of a Simulink model, along with any additional, user-defined output constraints.

Tip To automatically generate MATLAB code that computes operating points as specified in the Control and Estimation Tools Manager, click  or select **File > Generate MATLAB Code**.

Specifying Operating Points from Known Values

You can completely specify all inputs and states in the operating point.

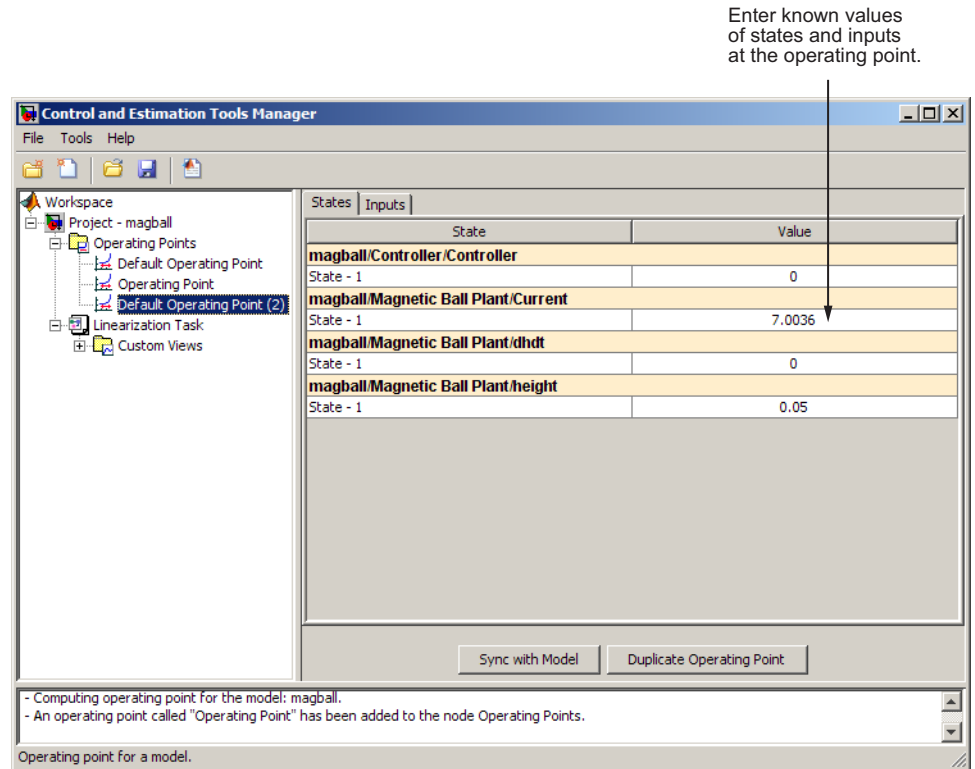
This section continues the magball example from “Computing Operating Points from Specifications” on page 2-10. At this stage in the example, a linearization task has already been created for the model, and a steady state operating point has been computed from specifications.

When you know the values of *all* states and inputs at the operating point, you can create a new operating point in the Control and Estimation Tools Manager and manually edit the operating point values:

- 1** Select the **Operating Points** node in the left pane and then click the **Operating Points** tab in the right pane.
- 2** Click the **New** button in the bottom-right corner to create a new operating point under the **Operating Points** node. This new operating point is labeled **Default Operating Point (2)**.

3 View the details of **Default Operating Point (2)** (shown in the following figure) by either:

- Selecting it under the **Operating Points** node in the project tree
- Selecting it in the **Operating Points** pane of the **Linearization Task** node, and then clicking **View**



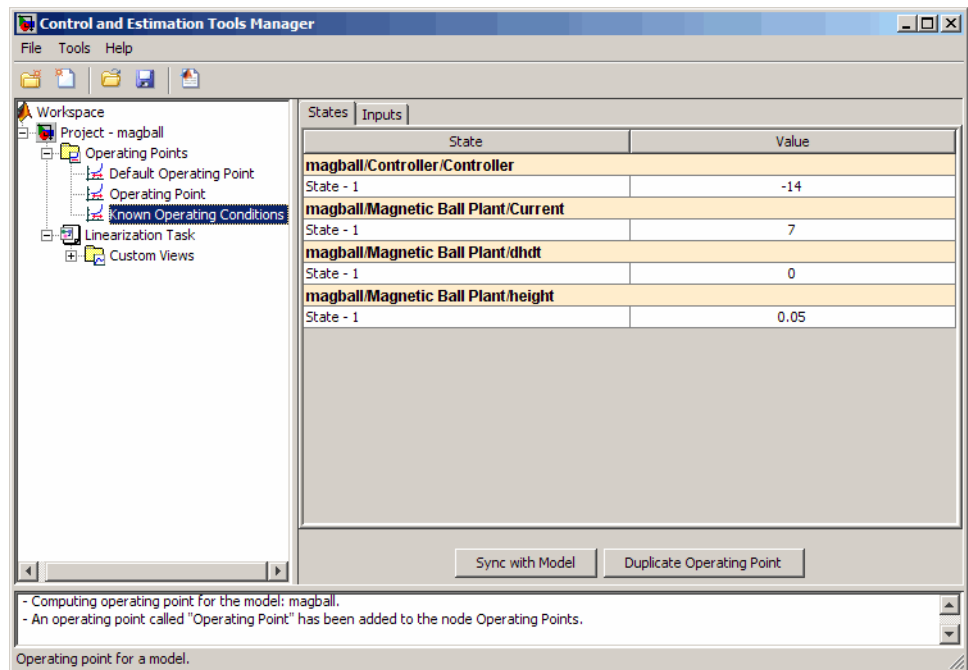
4 Edit the operating point by entering new values in the table. Switch between state and input values using the tabs on the left.


Change the value of **State-1** of **Controller** to -14 and the value of **Current** to 7. The model does not contain any root-level input ports, and, as a result, the **Inputs** pane is empty.

When you add states, inputs, or outputs to the model, or remove them from the model, click the **Sync with Model** button to update the operating point table to reflect these changes.

- 5 Rename the operating point by right-clicking **Default Operating Point (2)** under the **Operating Points** node, selecting **Rename**, and entering a new name in the dialog box.

For example, label this operating condition **Known Operating Conditions**. The pane should now resemble the following figure.



Tip To automatically generate MATLAB code that computes operating points as specified in the Control and Estimation Tools Manager, click  or select **File > Generate MATLAB Code**.

Extracting Operating Points From Simulation

You can create an operating point from a simulation of your model at the following simulation points:

- Specified simulation times, such as when the simulation reaches a steady state solution (see “Creating Operating Points at Specified Simulations Times” on page 2-18 example in this section).
- Events during a specified simulation interval (see “Creating Operating Points at Simulation Events” on page 2-21)

Creating Operating Points at Specified Simulations Times

You can extract an operating point at specified times during a simulation of the model.

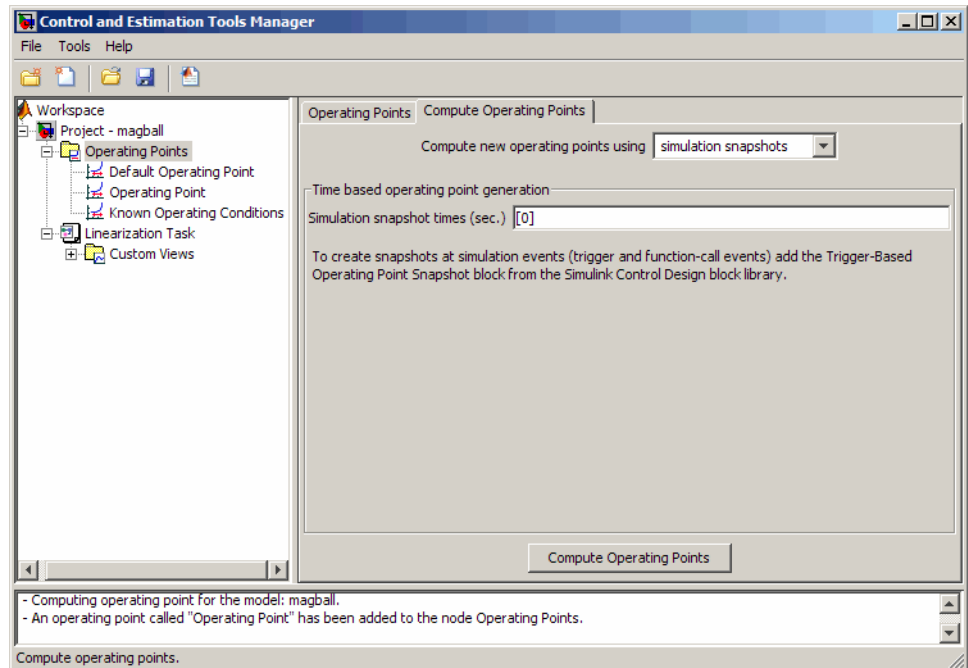
This section continues the magball example from “Specifying Operating Points from Known Values” on page 2-15. At this stage in the example, a linearization task has already been created for the model, a steady state operating point has been computed from specifications, and a completely known operating point has been specified.

To create operating points at specified simulation times:

1 Open the **Compute Operating Points** tab by either:

- Selecting the **Operating Points** node in the project tree, and then clicking the **Compute Operating Points** tab
- Clicking the **New Operating Point** button on the **Operating Points** pane of the **Linearization Task** node

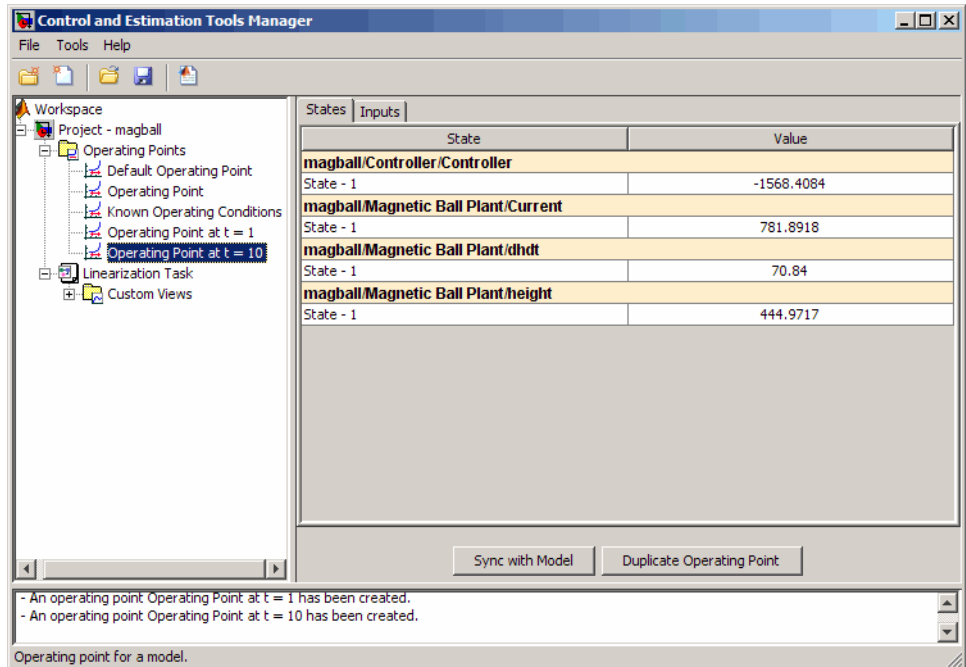
- From the **Compute new operating points using** list, select **simulation snapshots**. The window should now resemble the following figure.




- Enter a vector of times in the **Simulation snapshot times (sec)** field. Enter [1, 10] to compute operating points at $t=1$ and $t=10$.
- Click **Compute Operating Points**. The Simulink Control Design software simulates the model, extracts operating points, labeled **Operating Point**

at $t=1$ and **Operating Point at $t=10$** , and adds them to the **Operating Points** node in the project tree.

To view the contents of the operating point you created, select the operating point in the project tree as shown in this figure.



Note When you add states, inputs, or outputs to the model, or remove them from the model, click the **Sync with Model** button to update the operating point table to reflect these changes.

Tip To automatically generate MATLAB code that computes operating points as specified in the Control and Estimation Tools Manager, click  or select **File > Generate MATLAB Code**.

Creating Operating Points at Simulation Events

You can create an operating point from a simulation of your model at one or more of the following simulation events:

- Trigger-based events
- Function-call events

For more information about modeling events in Simulink models, see “Creating Conditional Subsystems” in the *Simulink User’s Guide*.

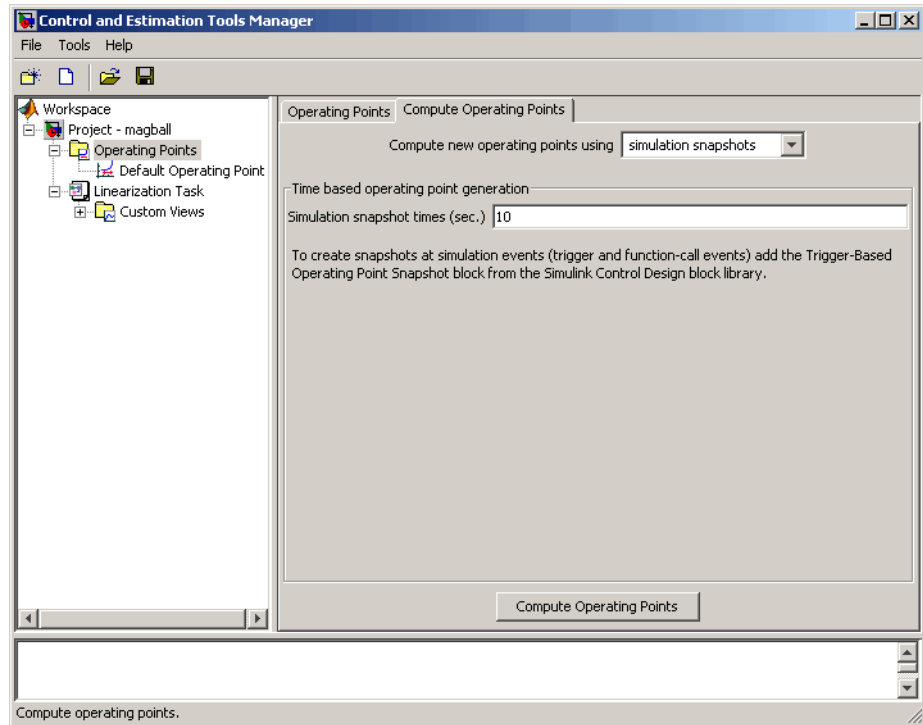
The Simulink Control Design software creates operating points at all simulation events within a specified simulation time.

To create operating points at one or more simulation events:

- 1** Add a Trigger-Based Operating Point Snapshot block to your model. This block is in the Simulink Control Design block library.

The model in the Trigger-Based Operating Point Snapshot demo shows the use of this block.

- 2** Select the **Compute Operating Points** tab in the **Operating Points** node.
- 3** From the **Compute new operating points using** list, select simulation snapshots.
- 4** Enter a scalar value that specifies the simulation end time in the **Simulation snapshot times (sec.)** field, shown in the following figure.



- 5 Click **Compute Operating Points**. The software simulates the model, extracts operating points, and adds them to the **Operating Points** node in the project tree. Select an operating point to view its contents and assess the results.

Computing Equilibrium Operating Points

You can use the software to compute equilibrium operating points. Follow the basic instructions in “Computing Operating Points from Specifications” on page 2-10. When you enter specifications in the **States** pane, select the **Steady State** check box at the top of the table. Selecting this check box causes the algorithm to look for an operating point in which all states are at equilibrium, or steady state.

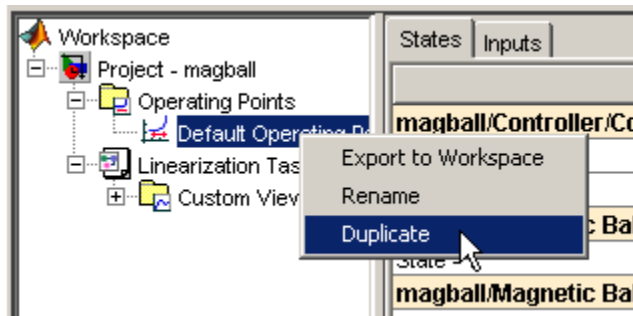
Working with Operating Points

In this section...

- “Copying Operating Points” on page 2-23
- “Exporting Operating Points” on page 2-24
- “Saving Operating Points” on page 2-25
- “Importing Operating Points” on page 2-25
- “Importing Initial Values” on page 2-26
- “Constraining Outputs” on page 2-27
- “Changing Optimization Settings” on page 2-27

Copying Operating Points

In some situations you might want to create and edit a copy of an operating point. To create a copy of an operating point, right-click the operating point in the tree on the left, and select **Duplicate** from the right-click menu, as shown in the following figure.

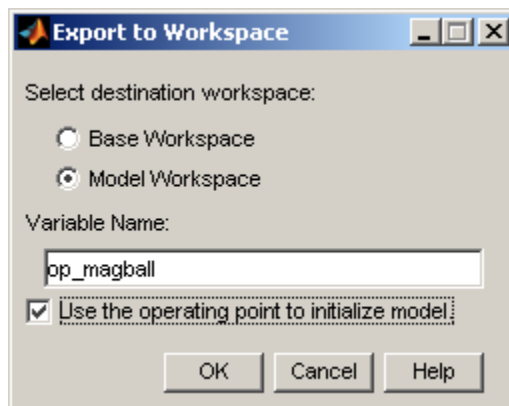


The new operating point appears beneath the original one in the tree. Click the new operating point to display its contents in the pane on the right. To change state or input values in the duplicated operating point, edit the values in the right pane. To change the name of the new operating point, right-click the operating point in the tree, select **Rename** from the right-click menu, and then enter a new name for the operating point.

Note that you cannot copy operating points that were computed from specifications. These operating points contain information related to the success of the optimization which would not be meaningful when the operating point values were changed.

Exporting Operating Points

After creating operating points using the Simulink Control Design software, you can export them from the Control and Estimation Tools Manager to the MATLAB workspace or the model workspace. You can use an exported operating point to perform analysis at the MATLAB command line or to initialize a Simulink model for simulation. To export an operating point, right-click the operating point under **Operating Points** in the pane on the left and select **Export to Workspace**. This opens the Export to Workspace dialog box, as shown below:



1 Click either

- **Base Workspace** to export the operating point to the MATLAB workspace where you can use it with Simulink Control Design command-line functions
- **Model Workspace** to export the operating point to the Model workspace where you can save it with the model for future use.

2 Enter a name for the exported operating point.

- 3 Select **Use the operating point to initialize model** when you want to use the operating point values as initial conditions for the states and inputs in the model. The initial values are automatically set in the **Data Import/Export** pane of the Configuration Parameters dialog box and Simulink uses these initial conditions when simulating the model.

Saving Operating Points

After you have exported the operating point to the MATLAB workspace, you can save it in a MAT-file for later use. To save the operating point `Operating_Point` in a file named `magball_operating_points.mat`, enter the following command:

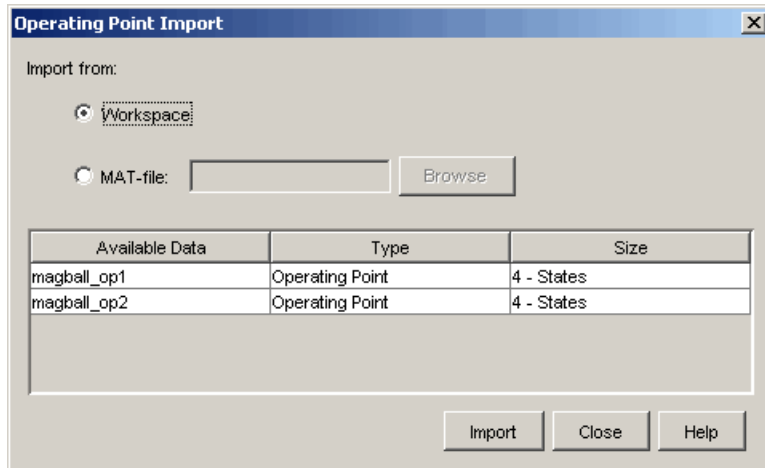
```
save magball_operating_point Operating_Point
```

Importing Operating Points

This section continues the example from “Example Model: The Magnetic Ball System” on page 1-2. At this stage in the example, a linearization project has already been created for the model, and linearization points have been inserted, and operating points have been created from specifications, known values, and simulation.

To import operating points from the MATLAB workspace or from a MAT-file.

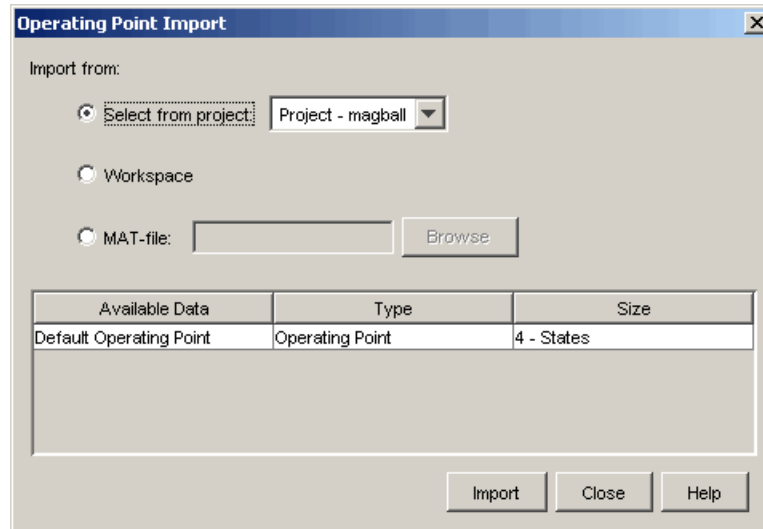
- 1 To import a new operating point, select the **Operating Points** node in the project tree and then select the **Operating Points** tab on the right. Click the **Import** button at the bottom of the pane. This displays the Operating Point Import dialog box.



- 2 Click **Workspace** or **MAT-file** as the location to import the operating point from, select an operating point from the list below, and then click **Import**. For this example, two operating points are loaded into the MATLAB workspace when you open the magball model.

Importing Initial Values

When you want populate the **Value** column of the operating point specifications by importing initial or known values from another operating point, a Simulink states structure, or a vector of values, click the **Import Initial Values** button at the bottom of the window. The Operating Point Import dialog box opens, as shown below.



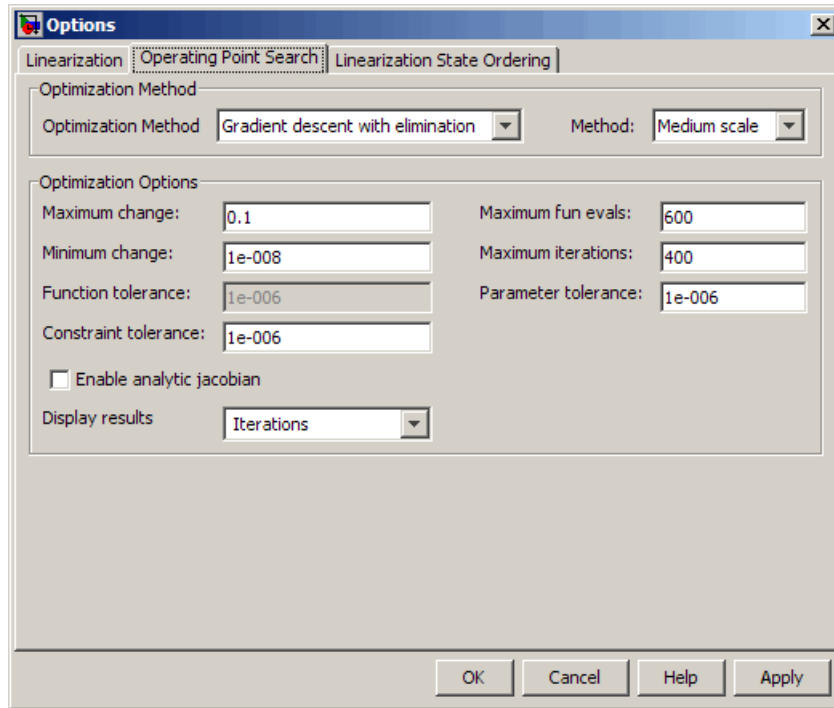
Select where to import the initial values from (a project, the workspace, or a file), then select the operating point from the list of available operating points below (or in the case of MAT-files, browse for a file). Click **Import** to import the initial values from the selected operating point into the **Value** column of the operating point specifications.

Constraining Outputs

Operating specifications often include constraints on the values of specific signals in the model. To constrain output signals when determining operating points from specifications, add an output constraint annotation to the model by right-clicking the signal line and choosing **Output Constraint** from the menu. This adds a small T to the signal line. Then, within the **Outputs** pane of the **Compute Operating Points** pane, select the **Known** check box and enter desired values as well as minimum and maximum values for this signal.

Changing Optimization Settings

To change the settings used when determining operating points by optimization, select **Tools > Options** and then click the **Operating Point Search** tab. This opens the Options dialog box.



To get help on each option or setting in the Options dialog box, right-click an option's label and select **What's This?**.



Additionally, you can refer to the Optimization Toolbox™ documentation and the `linoptions` reference page for more information about these settings. If you do not have the Optimization Toolbox documentation you can find it at

<http://www.mathworks.com/access/helpdesk/help/toolbox/optim/optim.shtml>

The methods Gradient descent with elimination, Simplex search, and Nonlinear least squares refer to the optimization methods `fmincon`, `fminsearch`, and `lsqnonlin` respectively. The method Gradient descent

refers to the optimization method `graddescent`, described in the `linoptions` reference page. The reference page for the Optimization Toolbox function `optimset` contains documentation for the following operating point search settings (the corresponding `optimset` parameter values are given in parentheses):

Operating Point Search Option	Parameter in <code>optimset</code>
Large Scale	<code>LargeScale</code> set to 'on'
Medium Scale	<code>LargeScale</code> set to 'off'
Maximum change	<code>DiffMaxChange</code>
Minimum change	<code>DiffMinChange</code>
Function tolerance	<code>TolFun</code>
Constraint tolerance	<code>TolCon</code>
Maximum fun evals	<code>MaxFunEvals</code>
Maximum iterations	<code>MaxIter</code>
Parameter tolerance	<code>TolX</code>
Enable analytic jacobian	Jacobian. When this option is selected, the Jacobian is computed at each iteration by linearizing the model about the current operating point. This option does not work with models that contain references to other models using the <code>Model</code> block or with models from products based on the Simscape™ platform that are in Trimming mode.
Display results	Display information contained in the output variable of the optimization functions, such as number of iterations, stepsize, etc.

Recommendations for Computing Operating Points

In this section...

“How to Create Accurate Operating Points” on page 2-31

“Impact of Blocks on the Simulink Model Operating Point” on page 2-31

“Computing Operating Points for SimMechanics Models” on page 2-36

“Choosing Initial Values for Computing Operating Points” on page 2-37

“Computing Operating Points for Blocks with Special Behavior” on page 2-38

How to Create Accurate Operating Points

Particular Simulink blocks and modeling situations can sometimes cause difficulties with computing operating points (trimming). However, by understanding what it means to trim a Simulink model and by using the correct modeling techniques, you can create accurate operating points for use in further analysis and design.

This section consists of examples that highlight modeling situations that can lead to problems when computing operating points, with recommendations for ways to avoid these situations.

Impact of Blocks on the Simulink Model Operating Point

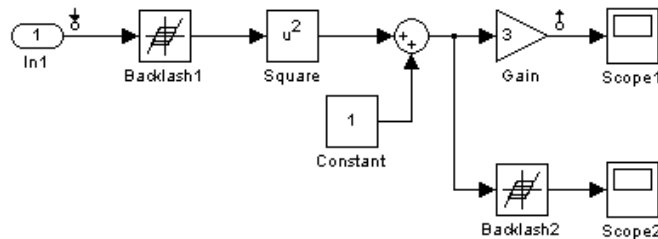
The full operating point in a Simulink model is specified in a number of ways by the blocks in the model:

- Integrator, State Space, and Transfer Function blocks have their outputs defined by double-valued discrete states.
- Source blocks such as Constant or Step blocks have their output specified by their block dialog parameters.
- Blocks such as Backlash, Memory, and Stateflow blocks have an internal state representation that impacts block outputs.

It is important to understand the impact of the blocks on the full operating point of your Simulink model. In particular, blocks with internal state representation can have a profound impact when you search for operating points or linearize a Simulink model. For more information on which blocks' states are included in an operating point versus a full model operating point, see "Simulink Model Operating Points" on page 2-3.

Example of the Impact of Blocks with Internal States

The following simple Simulink model shows the impact of blocks with internal states on the full operating point of a Simulink model. Each Backlash block has internal states that are initialized by the Initial output block dialog parameter.



The operating point for this model in the Simulink Control Design software *does not* include the backlash block states that exist in the model. See the following table for a comparison.

	States	Inputs
Full model operating point	2	1
Operating point	0	1

In this case, the value specified for the root level input is not propagated through the full model. However, the initial output for the Backlash1 block is propagated through the model.

When you linearize this model, the linearization is performed around the full model operating point, which includes the two states. For the input and

output points specified in this model, the second backlash block is not in the linearization path and thus its state does not impact the linearization result.

Types of Blocks with Internal States

Blocks with internal states that cannot be seen by the operating point object include:

- Action Subsystem blocks which are not enabled
- Backlash block
- Embedded MATLAB Function block with persistent data
- Transport Delay and Variable Transport Delay blocks
- Memory block
- Rate Transition block
- Stateflow[®] blocks
- S-Function blocks with states not registered as Continuous or Double Value Discrete

Finding Blocks with Internal States in Your Model

To determine when your model contains any of these blocks with internal states, run the following command:

```
sldiagnostics('modelName', 'CountBlocks')
```

This command returns a list of all the blocks in the model and the number of occurrences of each.

Working with Models Containing Blocks with Internal States

The following techniques provide strategies for working with models containing blocks with internal states:

- Block specific techniques
- Removing, replacing blocks, or both
- Linearizing at steady state using linearization snapshots

Block specific techniques exist for accurately computing operating points and linearizing models that contain the following blocks with internal states:

- “Memory Blocks” on page 2-34
- “Transport Delay and Variable Transport Delay Blocks” on page 2-36
- “Backlash Block” on page 2-36

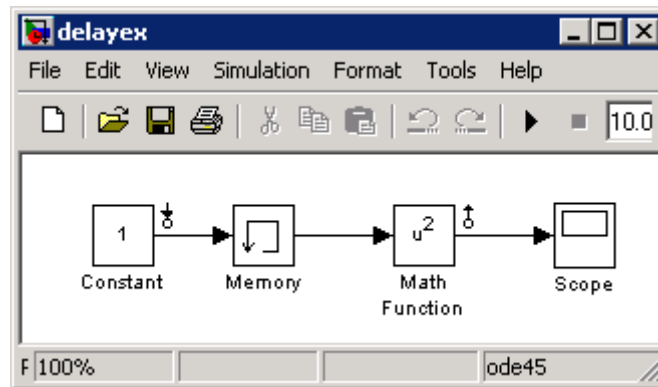
For other blocks with internal states, you should consider their impact on the analysis tools in the Simulink Control Design software in the following ways:

- When searching for an operating point you should determine if the output of the block impacts any of the state derivatives or desired output levels.
- When linearizing a model you should ascertain the effects on the model operating point. In particular, you should determine the effect on blocks between linearization input and output points.

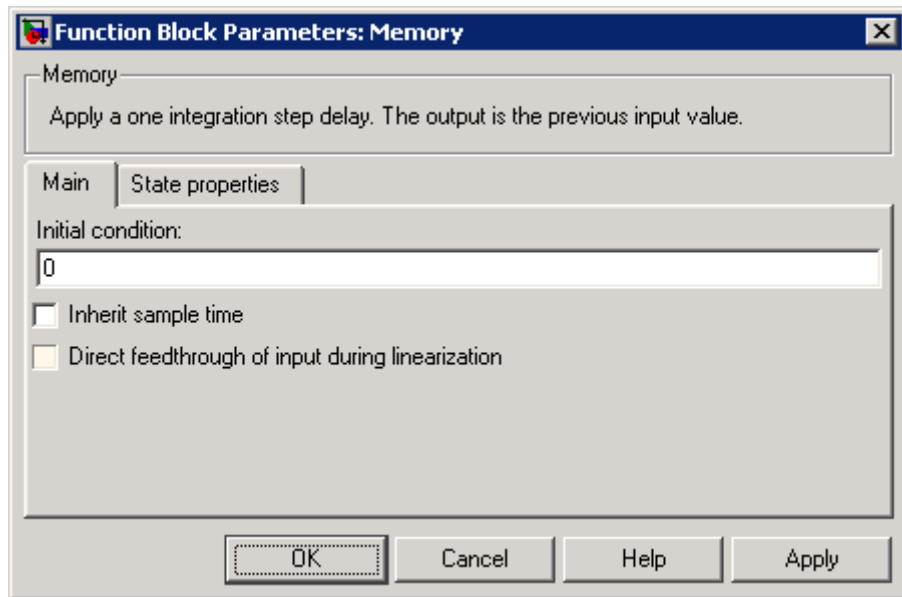
If the block does have impact, consider replacing it using a configurable subsystem when searching for an operating point and linearizing.

In many cases, performing a linearization using linearization snapshots avoids the challenges associated with blocks with internal states. You can linearize your model at steady state using linearization snapshots as described in “Linearizing at Specified Simulation Times” on page 4-61 and “Linearizing at Simulation Events” on page 4-63.

Memory Blocks. When you have Memory blocks in your model, you can configure the block to use a steady state output value when using the Simulink Control Design software. The model `delayex.mdl`, shown below, illustrates this issue.



In this model the Memory block is configured in the block dialog to have an initial output of 0 but is driven by a Constant block with an output of 1. This causes the output signal of the block to be 0 in the operating point. However, in the steady-state operating point for this model, the output of the Memory block is 1. When searching for an operating point or when linearizing a model at a steady state condition, select the **Direct feedthrough of input during linearization** option in the block dialog. This will force the output of the Memory block to be the same as the input during operating point searches or linearization.



Transport Delay and Variable Transport Delay Blocks. When you have Transport Delay or Variable Transport Delay blocks in your model, you can properly configure the initial outputs of these blocks so that operating point searches or linearization uses the correct output value at steady state condition. The discussion in “Memory Blocks” on page 2-34 applies to configuring the initial outputs of the Transport Delay and Variable Transport Delay blocks.

Backlash Block. The initial output and the output at the steady-state operating point of the Backlash block do not always match. There is no way to force the output of the Backlash block to be the same as the input during operating point searches or linearization. Extra care should be taken when working with a model containing Backlash blocks.

Computing Operating Points for SimMechanics Models

When computing operating points (trimming) for a SimMechanics™ model, you first need to put it in trimming mode. To do this:

- 1 Locate and open the machine environment (Env) block for the system.
- 2 From the **Parameters** pane, set **Analysis mode** to Trimming. Click **OK** to close the block dialog box. This will create an output port in the model that contains constraints related to errors in the system that must be set to zero for a steady state operating point.
- 3 To set these constraints to zero within a project for the model in the Control and Estimation Tools Manager, select **Operating Points** in the pane on the left, and then select **Compute Operating Points > Outputs**. Within this pane, set all constraints to 0.

At this point you can enter other design specifications on the states and inputs, and then compute an operating point for your model. After you have finished computing operating points for the SimMechanics model, make sure that you reset the **Analysis mode** to Forward dynamics in the Env block dialog box.

Choosing Initial Values for Computing Operating Points

When you compute an operating point from design specifications (trimming), it is often important to begin with a set of state and input values that are close to the actual steady state operating point values that you are trying to compute. To do this you can simulate the model for a specified period of time and then take a *snapshot* of the state and input values at that time. You can do this using either the Control and Estimation Tools Manager (see “Extracting Operating Points From Simulation” on page 2-18 for more information) or using the `findop` function (see “Extracting Values from Simulation” on page 3-16 for more information).

You can then use the values from the simulation snapshot as initial values for an operating point that you compute from specifications using optimization methods. To initialize the operating point specifications using these snapshot values, click the **Import Initial Values** button in the **Compute Operating Points** pane of the Control and Estimation Tools Manager, or use the `initopspec` function. For more information, see “Importing Operating Points” on page 2-25.

Computing Operating Points for Blocks with Special Behavior

Blocks such as Memory, Transport Delay, and Variable Transport Delay have states that cannot be optimized when computing operating points from specifications. In addition they do not have direct feedthrough as the input to the block at the current time does not determine the output of the block at the current time. This can cause problems when you determine operating points from specifications or create linearized models. To avoid these problems, select the **Direct feedthrough of input during linearization** option in the Block Parameters dialog box for the block in question (such as a Memory block) when determining operating points from specifications or linearizing models. This forces the input to *feed through* to the output, as if the system were operating at steady-state, and removes the problems associated with the states that cannot be used to compute operating points.

Operating Point Analysis Using the Command Line

- “Overview” on page 3-2
- “Example: Water-Tank System” on page 3-3
- “Creating or Opening a Simulink Model” on page 3-5
- “Computing Operating Points from Specifications” on page 3-7
- “Specifying Completely Known Operating Points” on page 3-14
- “Extracting Values from Simulation” on page 3-16
- “Using Structures and Vectors of Operating Point Values” on page 3-17

Overview

This section describes how to specify operating points for a model using functions in the MATLAB command window. Use the functions when you want to create M-files to automate the linearization process, or when you want to use an operating point to initialize a Simulink model. For a description of how to use the graphical interface for this task, see Chapter 2, “Operating Point Analysis Using the GUI”.

Before linearizing the model, you must choose an operating point to linearize the system about. This is often a steady-state value. Refer to “Why Are Operating Points Important?” on page 2-6 for more information on the role of operating points in linearization.

Use the Simulink Control Design functions for any of the following methods of specifying the operating point:

- You do not know all the input and state values, but you can characterize the operating point indirectly by specifying operating point values and constraints for specific signals and variables in the model (implicit specification).
- You know the operating point explicitly, i.e., you know the values of all inputs and states in the model.
- You want to simulate the model and extract the operating point at a given time.

Note The operating point consists of values for *all* the states in the model although only those states between the linearization points will be linearized. This is because the whole model contributes to the operating point values of the states/inputs/outputs of the portion of the model you are linearizing.

Example: Water-Tank System

In this section...

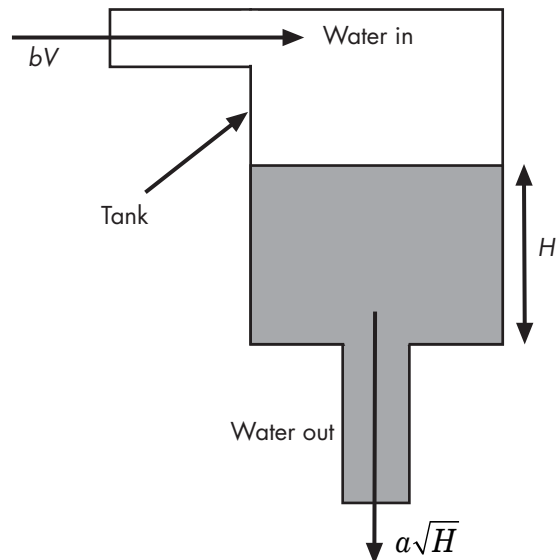
“Water-Tank System” on page 3-3

“Model Equations” on page 3-4

Water-Tank System

This section introduces an example that continues throughout the remaining sections of this chapter. By following this example, you will learn the process of linearizing a model using Simulink Control Design functions.

Water enters a tank from the top and leaves through an orifice in its base. The rate that water enters is proportional to the voltage, V , applied to the pump. The rate that water leaves is proportional to the square root of the height of water in the tank.



Schematic Diagram for the Water-Tank System

Model Equations

This section describes the model equations for the example started in the previous section “Example: Water-Tank System” on page 3-3.

A differential equation for the height of water in the tank, H , is given by

$$\frac{d}{dt} Vol = A \frac{dH}{dt} = bV - a\sqrt{H}$$

where Vol is the volume of water in the tank, A is the cross-sectional area of the tank, b is a constant related to the flow rate into the tank, and a is a constant related to the flow rate out of the tank. The equation describes the height of water, H , as a function of time, due to the difference between flow rates into and out of the tank.

The equation contains one state, H , one input, V , and one output, H . It is nonlinear due to its dependence on the square-root of H . Linearizing the model simplifies the analysis of this model. For information on the linearization process, see Chapter 4, “Exact Linearization Using the GUI”.

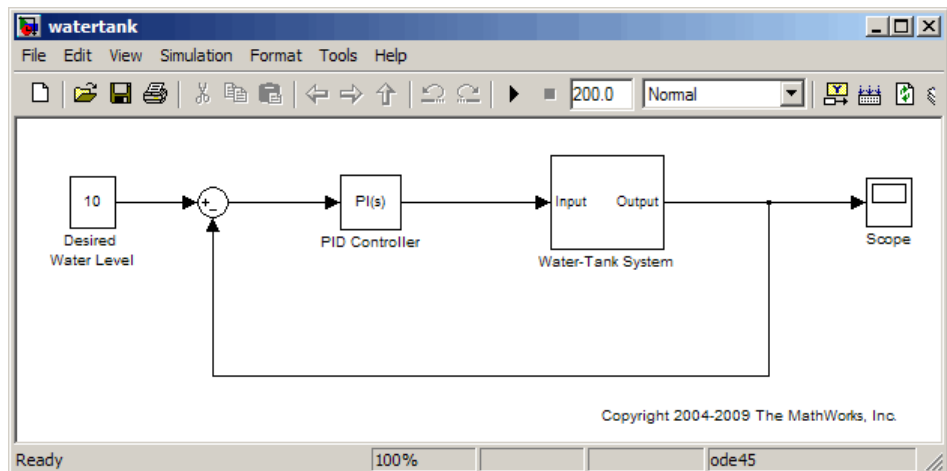
Creating or Opening a Simulink Model

To begin linearization using functions, you must first create or open a Simulink model of your system. The model can have any number of inputs and outputs (including none) and any number of states. The model can include user-defined blocks or S-functions. Your model can involve a plant only, a plant with a feedback loop and controller, or any number of subsystems.

To continue with the water-tank example, type

```
watertank
```

at the MATLAB prompt. This opens a Simulink model containing the water-tank system as shown in this figure.

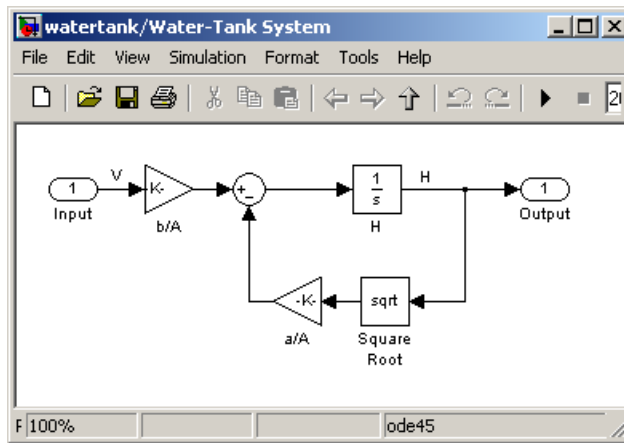


Simulink® Model of the Water-Tank System

The watertank model consists of

- The water-tank system itself
- A PID Controller to control the height of water in the tank by varying the voltage applied to the pump
- A reference signal that sets the desired water level
- A Scope block that displays the height of water as a function of time

Double-click a block to view its contents. The Water-Tank System block is shown in this figure.



Water-Tank System Block

The input to the Water-Tank System block, which is also the output of the PID Controller block, is the voltage, V . The output is the height of water, H . The system contains just one state (within the integrator), H . Values of the parameters are given as $a=2 \text{ cm}^{2.5}/\text{s}$, $A=20 \text{ cm}^2$, $b=5 \text{ cm}^3/(\text{s} \cdot \text{V})$.

Computing Operating Points from Specifications

In this section...

“Workflow for Computing Operating Points from Specifications” on page 3-7

“Creating an Operating Point Specification Object” on page 3-7

“Configuring the Operating Point Specification Object” on page 3-8

“Computing the Complete Operating Point” on page 3-10

“Alternative Method for Specifying Initial Guesses” on page 3-11

“Adding Output Constraints to Specifications” on page 3-12

Workflow for Computing Operating Points from Specifications

This section continues the example from “Example: Water-Tank System” on page 3-3. At this stage in the example, linearization point objects have been created in the MATLAB workspace. See “Selecting Inputs and Outputs for the Linearized Model” on page 5-4 for more information on creating linearization point objects.

To determine the operating points from specifications:

- 1** Create an operating point specification object. See “Creating an Operating Point Specification Object” on page 3-7.
- 2** Configure the object to store the specifications such as any constraints or known information about the operating point. See “Configuring the Operating Point Specification Object” on page 3-8.
- 3** Use the `findop` function to find the operating point values by optimization. See “Computing the Complete Operating Point” on page 3-10.

Creating an Operating Point Specification Object

When you know only some values exactly, or you know constraints on the values in the operating point, use the function `operspec` to create an operating point specification object for your model.

For example, to create an operating point specification object for the watertank model, type

```
watertank_spec =operspec('watertank')
```

MATLAB software displays

```
Operating Specification for the Model watertank.  
(Time-Varying Components Evaluated at time t=0)
```

```
States:
```

```
-----
```

```
(1.) watertank/PID Controller/Integrator  
    spec: dx = 0, initial guess:           0  
(2.) watertank/Water-Tank System/H  
    spec: dx = 0, initial guess:           1
```

```
Inputs: None
```

```
-----
```

```
Outputs: None
```

```
-----
```

Configuring the Operating Point Specification Object

The operating point specification object contains objects for all the states, inputs, and outputs in the model. By typing the object's name at the command line you can see a formatted display of key object properties. Alternatively, to list all the properties for a particular object, use the `get` function. For example

```
get(watertank_spec.States(1))
```

returns

```
Block: 'watertank/PID Controller/Integrator'  
StateName: ''  
x: 0  
Nx: 1  
Ts: [0 0]  
SampleType: 'CSTATE'  
inReferencedModel: 0
```

```

        Known: 0
    SteadyState: 1
            Min: -Inf
            Max: Inf
    Description: ''

```

Edit these properties to provide specifications for the operating point. For example:

- To set the second state to a known value (such as the desired height of water), first change `Known` to 1.

```
watertank_spec.States(2).Known=1
```

Next, provide the known value.

```
watertank_spec.States(2).x=10
```

- To find a steady-state value for the first state, set `SteadyState` to 1.

```
watertank_spec.States(1).SteadyState=1
```

- To provide an initial guess of 2 for this steady-state value, first make sure that `Known` is set to 0 for this state.

```
watertank_spec.States(1).Known=0
```

Then, provide the initial guess.

```
watertank_spec.States(1).x=2
```

- To set a lower bound of 0 on this state,

```
watertank_spec.States(1).Min=0
```

Optimization settings used with the `findop` function can be configured with the `linoptions` function.

Computing the Complete Operating Point

The operating point specification object, `watertank_spec`, now contains specifications for the operating point. Use this information to find the complete operating point using the `findop` command. Type

```
[watertank_op,op_report]=findop('watertank',watertank_spec)
```

This returns the optimized operating point. The optimized values of the states are contained in the `x` property, or `u` property for inputs.

```
Operating Point for the Model watertank.  
(Time-Varying Components Evaluated at time t=0)
```

```
States:
```

```
-----
```

```
(1.) watertank/PID Controller/Integrator
```

```
    x: 1.26
```

```
(2.) watertank/Water-Tank System/H
```

```
    x: 10
```

```
Inputs: None
```

```
-----
```

The operating point search report, `op_report`, is also generated. The `x` or `u` values give the state or input values. The `dx` values give the time derivatives of each state, with desired `dx` values in parentheses. The fact that the `dx` values are both zero indicates that the operating point is at steady state.

```
Operating Report for the Model watertank.  
(Time-Varying Components Evaluated at time t=0)
```

```
Operating point specifications were successfully met.
```

```
States:
```

```
-----
```

```
(1.) watertank/PID Controller/Integrator
```

```
    x:          1.26    dx:          0 (0)
```

```
(2.) watertank/Water-Tank System/H
```

```
    x:          10    dx:          0 (0)
```

Inputs: None

Outputs: None

Alternative Method for Specifying Initial Guesses

In some cases you might want to use a previously created operating point to specify initial guesses in another operating point specification object. For example, after extracting an operating point from a simulation, as in “Extracting Values from Simulation” on page 3-16, you can use this operating point as a starting point for finding a more accurate steady state value using `findop`. You can do this with the `initopspec` function.

For example, first extract an operating point from simulation, in this case after 10 time units.

```
watertank_op2=findop('watertank',10);
```

Then create an operating point specification object.

```
watertank_spec=operspec('watertank');
```

Specify initial guesses in this object with the following command.

```
watertank_spec=initopspec(watertank_spec,watertank_op2)
```

This returns an operating point specification with the initial guess, or `x` property filled with operating point values from `watertank_op2`.

```
Operating Specification for the Model watertank.
(Time-Varying Components Evaluated at time t=0)
```

States:

```
(1.) watertank/PID Controller/Integrator
      spec: dx = 0, initial guess:      1.69
(2.) watertank/Water-Tank System/H
      spec: dx = 0, initial guess:     10.1
```

```
Inputs: None
```

```
-----
```

```
Outputs: None
```

```
-----
```

This operating point specification can now be used with `findop` to find an optimized steady state operating point. You can access individual elements of this object using the `get` function or dot-notation as in “Configuring the Operating Point Specification Object” on page 3-8.

Adding Output Constraints to Specifications

When you want to constrain additional signals of the model, you can add an output constraint to the operating point specification object with the function `addoutputspec`.

For example, to add an output constraint to the operating point specification created in “Alternative Method for Specifying Initial Guesses” on page 3-11, use the following command:

```
watertank_spec=addoutputspec(watertank_spec, 'watertank/Water-Tank System/Sum',1)
```

This adds a constraint on the signal between the Sum block and the integrator block, H, within the Water-Tank System block. The constraint is associated with an output of a block, in this case output 1 of the block `watertank/Water-Tank System/Sum` (the preceding block).

```
Operating Specification for the Model watertank.  
(Time-Varying Components Evaluated at time t=0)
```

```
States:
```

```
-----
```

```
(1.) watertank/PID Controller/Integrator  
    spec: dx = 0, initial guess:      1.69  
(2.) watertank/Water-Tank System/H  
    spec: dx = 0, initial guess:     10.1
```

```
Inputs: None
```

```
-----
```

Outputs:

(1.) watertank/Water-Tank System/Sum
spec: none

You can edit the specifications for this output in the same way as you would for any other specifications, by changing the values of Known, y, Min, etc. There is no `SteadyState` option for outputs.

Specifying Completely Known Operating Points

In this section...
“Workflow for Specifying Completely Known Operating Points” on page 3-14
“Creating an Operating Point Object” on page 3-14
“Changing Operating Point Values” on page 3-15

Workflow for Specifying Completely Known Operating Points

To use functions to specify completely known operating points

- 1 Create an operating point object.
- 2 Make changes to the object values.

This section continues the example from “Example: Water-Tank System” on page 3-3. At this stage in the example, linearization point objects have been created in the MATLAB workspace. See “Selecting Inputs and Outputs for the Linearized Model” on page 5-4 for more information on creating linearization point objects.

Creating an Operating Point Object

An operating point object contains information about your system’s states and inputs at the operating point. When you know your operating point explicitly, use the function `operpoint` to create an operating point object for your model.

For example, to create an operating point object for the water-tank model, type

```
watertank_op=operpoint('watertank')
```

MATLAB software displays

```
Operating Point for the Model watertank.  
(Time-Varying Components Evaluated at time t=0)
```

```
States:
```



```

-----
(1.) watertank/PID Controller/Integrator
    x: 0
(2.) watertank/Water-Tank System/H
    x: 1

Inputs: None
-----

```

Within the operating point object are objects for all the states and inputs in the model. Each of these objects has a property called `x`, or `u` in the case of inputs, that gives the value of that state or input.

Changing Operating Point Values

Change the `x` and `u` properties of the operating point object to known values at the operating point. For example, to change the value of the first state to 1.26 and the second state to 10, type

```
watertank_op.States(1).x=1.26, watertank_op.States(2).x=10
```

which returns

```

Operating Point for the Model watertank.
(Time-Varying Components Evaluated at time t=0)

States:
-----
(1.) watertank/PID Controller/Integrator
    x: 1.26
(2.) watertank/Water-Tank System/H
    x: 10

Inputs: None
-----

```

The operating point object, `watertank_op`, now contains the known operating point of the system.

Extracting Values from Simulation

This section continues the example from “Example: Water-Tank System” on page 3-3. At this stage in the example, linearization point objects have been created in the MATLAB workspace. See “Selecting Inputs and Outputs for the Linearized Model” on page 5-4 for more information on creating linearization point objects.

Use Simulink Control Design software to extract operating points from a simulation of your model at specified times, such as when the simulation reaches a steady state solution.

For example, to create an operating point object for the water-tank model after it has simulated for 20 time units, type

```
watertank_op=findop('watertank',20)
```

MATLAB software displays the operating point at time t=20.

```
Operating Point for the Model watertank.  
(Time-Varying Components Evaluated at time t=20)
```

```
States:
```

```
-----
```

```
(1.) watertank/PID Controller/Integrator  
    x: 1.54  
(2.) watertank/Water-Tank System/H  
    x: 10.2
```

```
Inputs: None
```

```
-----
```

Using Structures and Vectors of Operating Point Values

This section continues the example from “Example: Water-Tank System” on page 3-3. At this stage in the example, linearization point objects and operating points have been created in the MATLAB workspace. See Chapter 3, “Operating Point Analysis Using the Command Line” for more information on creating operating point objects using functions.

Operating point objects store the operating point values. However, when you want to use an operating point’s values to initialize the simulation of a model, it is useful to work with *vectors* of operating point values, or with Simulink structures. Simulink structures have the benefits that you can use them to initialize models that reference other models using the Model block, and you do not need to worry about the ordering of states in the structure.

You can extract vectors and structures of operating point values from operating point objects using the functions `getxu` and `getstatestruct`. You can then use these vectors or structures to initialize a model for simulation. Models that reference other models using the Model block, must be initialized with a Simulink structure of values, such as those from simulation data, extracted with the `getstatestruct` function. See “Importing and Exporting States” in the Simulink documentation for details on initializing model reference models.

To extract a structure of operating point values from the operating point object, `watertank_op`, created in “Extracting Values from Simulation” on page 3-16, use the following command:

```
x=getstatestruct(watertank_op)
```

This extracts a structure of state values, `x` from the operating point object:

```
x =  
  
    time: 20  
  signals: [1x2 struct]
```

To access the values within this structure, use the following syntax:

```
x.signals.values
```

which returns

```
ans =  
    1.5431  
  
ans =  
    10.1872
```

Note that these values are in the same order as those used by Simulink.

To extract a vector of operating point values from the operating point object, `watertank_op`, use the following command.

```
[x,u]=getxu(watertank_op)
```

This extracts vectors of states, `x`, and inputs, `u`, as shown below.

```
x =  
    10.1872  
     1.5431  
  
u =  
    []
```

To create an operating point object from a vector, or structure, of values, such as those returned from a simulation, you can use the function `setxu`. To set operating point values in an operating point object using a vector or structure of known values, you can use the following command.

```
new_op=setxu(watertank_op2,x,u)
```

This command creates a new operating point, `new_op`, that is based on the operating point `watertank_op2`, but with the values from the vector or structure of state values, `x`, and the vector of input values, `u`.

The ordering of the states in `x` and the inputs in `u` must be the same as the ordering used by Simulink which is not necessarily the same as the order the states appear in the operating point object. When you extract values from simulation data they will already be in the correct order.

Exact Linearization Using the GUI

- “What Is Linearization?” on page 4-2
- “Ways to Linearize Models” on page 4-7
- “Steps for Linearizing Models Using the GUI” on page 4-8
- “Choosing Linearization Settings and Algorithms” on page 4-9
- “Configuring the Linearization of Specific Blocks and Subsystems” on page 4-36
- “Selecting Inputs and Outputs for the Linearized Model” on page 4-45
- “Linearizing a Block” on page 4-55
- “Linearizing the Model” on page 4-57
- “Viewing Linearization Results” on page 4-66
- “Validating Exact Linearization Results” on page 4-76
- “Troubleshooting Exact Linearization Results” on page 4-82

What Is Linearization?

In this section...

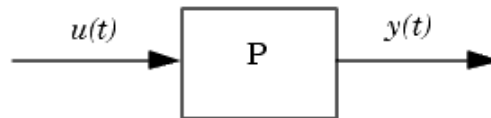
“Linearization Background” on page 4-2

“Analytic Representations of Linear Models” on page 4-3

Linearization Background

A linearized model is an approximation to a nonlinear system, which is valid in a small region around the operating point of the system. Engineers often use linearization in the design and analysis of control systems and physical models.

The following figure shows a visual representation of a nonlinear system as a block diagram. The diagram consists of an external input signal, $u(t)$, a measured output signal, $y(t)$, and the nonlinear system that describes the system’s states and its dynamic behavior, P .



You can also express a nonlinear system in terms of the state space equations

$$\dot{x}(t) = f(x(t), u(t), t)$$

$$y(t) = g(x(t), u(t), t)$$

where $x(t)$ represents the system’s states, $u(t)$ represents the inputs, and $y(t)$ represents the outputs. In these equations, the variables vary continuously with time. Discrete-time and multi-rate models are discussed in “Analytic

Representations of Linear Models” on page 4-3. A linear time-invariant approximation to this nonlinear system is valid in a region around the operating point at $t=t_0$, $x(t_0)=x_0$, and $u(t_0)=u_0$. If the values of the system’s states, $x(t)$ and inputs, $u(t)$ are close enough to the operating point, the system will behave approximately linearly.

Simulink uses a series of connected *blocks* to model physical systems and control systems. Input and output signals connect the blocks, which represent mathematical operations. The nonlinear system, P, in the previous figure, represents a series of connected Simulink blocks.

The Simulink Control Design software linearizes both continuous and discrete-time nonlinear systems by computing the state-space matrices of the linearized model, A , B , C , and D , using one of the linearization algorithms described in “Choosing Linearization Settings and Algorithms” on page 4-9.

Analytic Representations of Linear Models

Linearization of Nonlinear Models

To describe the linearized model, it helps to first define a new set of variables centered about the operating point of the states, inputs, and outputs:

$$\delta x(t) = x(t) - x_0$$

$$\delta u(t) = u(t) - u_0$$

$$\delta y(t) = y(t) - y_0$$

The value of the outputs at the operating point is given by $y(t_0)=g(x_0, u_0, t_0)=y_0$.

Note When comparing a linearized model with the original model, remember that the convention used in this book is to write the linearized model in terms of δx , δu , and δy . The value of each of these variables at the operating point is zero.

The linearized state space equations written in terms of $\delta x(t)$, $\delta u(t)$, and $\delta y(t)$ are

$$\begin{aligned}\delta\dot{x}(t) &= A\delta x(t) + B\delta u(t) \\ \delta y(t) &= C\delta x(t) + D\delta u(t)\end{aligned}$$

where A , B , C , and D are constant coefficient matrices. These matrices are defined as the Jacobians of the system, evaluated at the operating point

$$\begin{aligned}A &= \left. \frac{\partial f}{\partial x} \right|_{t_0, x_0, u_0} & B &= \left. \frac{\partial f}{\partial u} \right|_{t_0, x_0, u_0} \\ C &= \left. \frac{\partial g}{\partial x} \right|_{t_0, x_0, u_0} & D &= \left. \frac{\partial g}{\partial u} \right|_{t_0, x_0, u_0}\end{aligned}$$

The transfer function of the linearized model can be used in place of the system, P , in the previous figure. To find the transfer function, divide the Laplace transform of $\delta y(t)$ by the Laplace transform of $\delta u(t)$:

$$P_{lin}(s) = \frac{\delta Y(s)}{\delta U(s)}$$

Linearization of Discrete-Time Models

Discrete-time models are similar to continuous models, discussed in the previous section, with the exception that the values of system variables change at discrete times, t_k , where k is an integer value. The state-space equations for a nonlinear, discrete-time system are

$$\begin{aligned}x_{k+1} &= f(x_k, u_k, t_k) \\ y_k &= g(x_k, u_k, t_k)\end{aligned}$$

A linear time-invariant approximation to this system is valid in a region around the operating point

$$t_k = t_{k_0}, x_k = x_{k_0}, u_k = u_{k_0}, \text{ and } y_k = g(x_{k_0}, u_{k_0}, t_{k_0}) = y_{k_0}$$

If the values of the system's states, x_k , inputs, u_k , and outputs, y_k , are close enough to the operating point, the system will behave approximately linearly.

As with continuous time systems it is helpful to define variables centered about the operating point values

$$\delta x_k = x_k - x_{k_0}$$

$$\delta u_k = u_k - u_{k_0}$$

$$\delta y_k = y_k - y_{k_0}$$

where the value of the outputs at the operating point are defined as:

$$y_{k_0} = g(x_{k_0}, u_{k_0}, t_{k_0})$$

The linearized state-space equations can then be written in terms of these new variables

$$\delta x_{k+1} \approx A \delta x_k + B \delta u_k$$

$$\delta y_k \approx C \delta x_k + D \delta u_k$$

where A , B , C , and D are given by

$$A = \left. \frac{\partial f}{\partial x_k} \right|_{t_0, x_0, u_0} \quad B = \left. \frac{\partial f}{\partial u_k} \right|_{t_0, x_0, u_0}$$

$$C = \left. \frac{\partial g}{\partial x_k} \right|_{t_0, x_0, u_0} \quad D = \left. \frac{\partial g}{\partial u_k} \right|_{t_0, x_0, u_0}$$

Linearization of Multirate Models

Multirate models involve states with various sampling rates. This means that the state variables change values at different times and with different frequencies, with some variables possibly changing continuously. The general state-space equations for a nonlinear, multirate system are

$$\begin{aligned}\dot{x}(t) &= f(x(t), x_1(k_1), \dots, x_m(k_m), u(t), t) \\ x_1(k_1 + 1) &= f_1(x(t), x_1(k_1), \dots, x_m(k_m), u(t), t) \\ &\vdots \\ x_m(k_m + 1) &= f_i(x(t), x_1(k_1), \dots, x_m(k_m), u(t), t) \\ y(t) &= g(x(t), x_1(k_1), \dots, x_m(k_m), u(t), t)\end{aligned}$$

where k_1, \dots, k_m are integer values and t_{k_1}, \dots, t_{k_m} are discrete times.

The linearized equations will approximate this system as a single-rate discrete model:

$$\begin{aligned}\delta x_{k+1} &\approx A\delta x_k + B\delta u_k \\ \delta y_k &\approx C\delta x_k + D\delta u_k\end{aligned}$$

For more information, see the Simulink Control Design demo “Linearization of Multirate Models”.

Ways to Linearize Models

You can linearize Simulink models either:

- Interactively in the GUI

For more information, see “Steps for Linearizing Models Using the GUI” on page 4-8.

- Programmatically at the command line using MATLAB code

For more information, see Chapter 5, “Exact Linearization Using the Command Line”. If you are computing multiple linearizations of large models when only a few blocks or model references change per linearization, see “Computing Multiple Linearizations for Large Models” on page 5-13.

Tip You can automatically generate MATLAB code from your GUI configuration.

You can also compute the frequency response of Simulink models, including models with event-based dynamics that might not linearize using exact linearization. Examples of event-based dynamics are

- Stateflow charts
- Triggered subsystems
- PWM signals

For information on how to compute the frequency response of a Simulink model, see Chapter 6, “Frequency Response Estimation of Simulink Models”.

Steps for Linearizing Models Using the GUI

The main steps to linearize a model using the Simulink Control Design GUI are as follows:

- 1** Create or open a model. See “Creating or Opening a Simulink Model” on page 1-2.
- 2** Create a new linearization task on the Control and Estimation Tools Manager. See “Creating a Linearization Task” on page 1-8.
- 3** Specify an operating point for the model. See “Creating Operating Points” on page 2-10.
- 4** (optional) Specify the linearization settings and method. See “Choosing Linearization Settings and Algorithms” on page 4-9.
- 5** (optional) Configure how specific blocks and subsystems in your model linearize. See “Configuring the Linearization of Specific Blocks and Subsystems” on page 4-36.
- 6** Insert linearization input and output points in the model. See “Selecting Inputs and Outputs for the Linearized Model” on page 4-45.
- 7** Linearize the model. See “Linearizing the Model” on page 4-57.
- 8** Inspect the linearization. See “Viewing Linearization Results” on page 4-66.
- 9** If necessary, validate and troubleshoot the linearization. See “Validating Exact Linearization Results” on page 4-76 and “Validating Exact Linearization Results” on page 4-76.
- 10** Save your project and export the results to the MATLAB Workspace. See “Saving Projects” on page 1-11 and “Exporting Results” on page 1-13.

The first three steps in this process were completed in the previous chapters. This chapter continues the magball model example to give a detailed discussion of the remaining steps.

Choosing Linearization Settings and Algorithms

In this section...

“How to Choose Linearization Settings and Algorithms” on page 4-9

“Options for Linearization Algorithm Method” on page 4-11

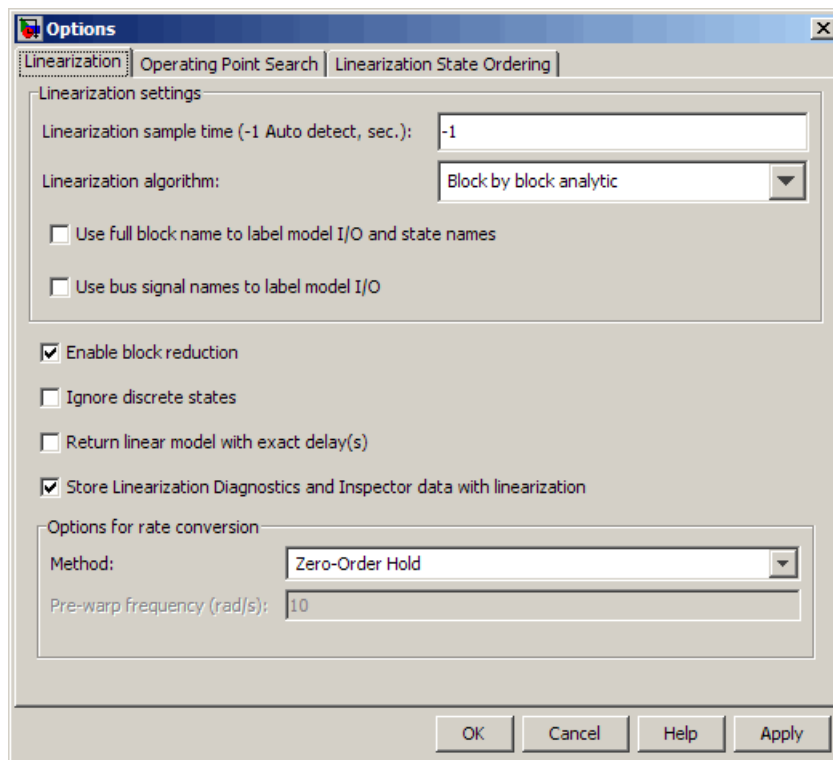
“Block-by-Block Analytic Linearization” on page 4-12

“Numerical-Perturbation Linearization” on page 4-24

“Changing State Ordering in the Linearized Model” on page 4-34

How to Choose Linearization Settings and Algorithms

To change the linearization settings and algorithms, select **Tools > Options** in the Control and Estimation Tools Manager window, and then click the **Linearization** tab. This opens the Linearization Task Options dialog box.



To get help on each option or setting in the Options dialog box, right-click an option's label and select **What's This?**.



For more information on these settings, refer to the `linoptions` reference page. For information about numerical-perturbation linearization, which is used when you select `Numerical perturbation` as the **Linearization algorithm** parameter, see “Numerical-Perturbation Linearization” on page 4-24.

Options for Linearization Algorithm Method

You can choose from the following two linearization methods in the Simulink Control Design software:

- Block-by-block analytic linearization (the default method)
- Numerical-perturbation linearization

Note To use numerical-perturbation linearization, you must select an option in the Linearization Options dialog box of the GUI, or if you are using functions, with the `linoptions` function.

Advantages of Block-by-Block Analytical Linearization

The default linearization method, block-by-block analytic linearization, linearizes the blocks individually and then combines the results to produce the linearization of the whole system. This method has several advantages:

- It divides the linearization problem into several smaller, easier problems.
- It defines the system being linearized by input and output markers on the signal lines rather than root-level inport and outport blocks.
- It supports open-loop analysis.
- You can control the linearization of each block by using an analytic linearization that is programmed into the block or by selecting a perturbation level for the block.
- You can compute linearized models with exact representations of continuous time delays.

For more information, see “Block-by-Block Analytic Linearization” on page 4-12.

Advantages and Disadvantages of Numerical-Perturbation Linearization

Numerical-perturbation linearization linearizes the *whole system* by numerically perturbing the system’s inputs and states around the operating

point. The advantage of this method is that it is quick and simple, especially for large or complicated systems. However, there are also several disadvantages with this method:

- It relies on root-level inport and outport blocks to define the system being linearized.
- There is no support for open-loop analysis.
- You have limited control over the perturbation levels for each block.
- It does not use any of the analytic, preprogrammed block linearizations.
- It is sensitive to scaling issues (models with large and small signal values).

For more information, see “Numerical-Perturbation Linearization” on page 4-24.

Block-by-Block Analytic Linearization

- “What Is Block-by-Block Analytic Linearization?” on page 4-12
- “Linearizing Individual Blocks Using Analytic Linearization” on page 4-13
- “Linearizing Individual Blocks Using Block Perturbation” on page 4-13
- “Linearizing Models with Time Delays” on page 4-15
- “Blocks with Discontinuities” on page 4-17
- “Integrator Blocks Near Saturation or a Reset Point” on page 4-18
- “Event-Based Models and Triggered Subsystems” on page 4-20
- “Pulse Width Modulation” on page 4-22

What Is Block-by-Block Analytic Linearization?

Block-by-block analytic linearization is the default linearization method in the Simulink Control Design software. This method linearizes each block within the linearization path individually.

There are two types of block-by-block linearization:

- “Linearizing Individual Blocks Using Analytic Linearization” on page 4-13

- “Linearizing Individual Blocks Using Block Perturbation” on page 4-13

Each method has options that you can control to create accurate linearized models.

Linearizing Individual Blocks Using Analytic Linearization

You can linearize blocks with analytic Jacobians using analytic linearization. This type of linearization results in an exact linearization of each block. When you linearize a system using block-by-block analytic linearization, the Simulink Control Design software uses these exact linearizations instead of numerically perturbing the block. This approach is especially useful for blocks that contain discontinuities and do not give good results using numerical perturbation.

Note The preprogrammed, analytic block linearizations are only used in block-by-block analytic linearization. When you use the numerical-perturbation linearization method, such blocks are numerically perturbed with the rest of the system.

Linearizing Individual Blocks Using Block Perturbation

When you cannot use a preprogrammed block linearization, the Simulink Control Design software automatically computes the block linearization by numerically perturbing the states and inputs of the block about the operating point of the block. As opposed to the numerical-perturbation linearization method, this perturbation is local and its propagation through the rest of the model is restricted.

Block Perturbation Algorithm. The block perturbation algorithm introduces a small perturbation to the nonlinear block and measuring the response to this perturbation. Both the perturbation and the resulting response are used to create the matrices in the linear state-space model of this block. Changing the size of the perturbations changes the resulting linearized model.

As described in “What Is Linearization?” on page 4-2, you can write a nonlinear Simulink block as a state-space system:

$$\begin{aligned}\dot{x}(t) &= f(x(t), u(t), t) \\ y(t) &= g(x(t), u(t), t)\end{aligned}$$

In these equations, $x(t)$ represents the states of the block, $u(t)$ represents the inputs of the block, and $y(t)$ represents the outputs of the block.

A linearized model of this system is valid in a small region around the operating point $t=t_0$, $x(t_0)=x_0$, $u(t_0)=u_0$, and $y(t_0)=g(x_0, u_0, t_0)=y_0$. Subtracting the operating point values from the states, inputs, and outputs defines a set of variables centered about the operating point:

$$\begin{aligned}\delta x(t) &= x(t) - x_0 \\ \delta u(t) &= u(t) - u_0 \\ \delta y(t) &= y(t) - y_0\end{aligned}$$

You can write the linearized model in terms of these new variables. The representation is usually valid when the variables are small, i.e., when the departure from the operating point is small:

$$\begin{aligned}\delta \dot{x}(t) &= A\delta x(t) + B\delta u(t) \\ \delta y(t) &= C\delta x(t) + D\delta u(t)\end{aligned}$$

The state-space matrices A , B , C , and D of this linearized model represent the Jacobians of the block, as defined in “What Is Linearization?” on page 4-2. To compute the matrices, the states and inputs are perturbed, one at a time, and the response of the system to this perturbation is measured by computing $\delta \dot{x}$ and δy . The perturbation and response are then used to compute the matrices in the following way:

$$\begin{aligned}A(:, i) &= \frac{\dot{x}|_{x_{p,i}} - \dot{x}_o}{x_{p,i} - x_o}, & B(:, i) &= \frac{\dot{x}|_{u_{p,i}} - \dot{x}_o}{u_{p,i} - u_o} \\ C(:, i) &= \frac{y|_{x_{p,i}} - y_o}{x_{p,i} - x_o}, & D(:, i) &= \frac{y|_{u_{p,i}} - y_o}{u_{p,i} - u_o}\end{aligned}$$

where

- $x_{p,i}$ is the state vector whose i th component is perturbed from the operating point value.
- x_o is the state vector at the operating point.
- $u_{p,i}$ is the input vector whose i th component is perturbed from the operating point value.
- u_o is the input vector at the operating point.
- $\dot{x}|_{x_{p,i}}$ is the value of \dot{x} at $x_{p,i}, u_o$.
- $\dot{x}|_{u_{p,i}}$ is the value of \dot{x} at $u_{p,i}, x_o$.
- \dot{x}_o is the value of \dot{x} at the operating point.
- $y|_{x_{p,i}}$ is the value of y at $x_{p,i}, u_o$.
- $y|_{u_{p,i}}$ is the value of y at $u_{p,i}, x_o$.
- y_o is the value of y at the operating point.

Linearized models of discrete-time are computed in a similar way. For more information, see “Linearizing Discrete-Time and Multirate Models” on page 4-65 for the equations of linearized discrete-time and multirate systems.

Note A *perturbed* value is one that has been changed by a very small amount from the operating point value. The default difference between the perturbed value and the operating point value is $10^{-5}(1+|x|)$ for block-by-block analytic linearization, where x is the operating point value.

Linearizing Models with Time Delays

You can linearize models with time delays using:

- Padé approximation provides the following results:

- An approximate representation of continuous delays using the Padé order you specify in the block dialog for the delay Simulink blocks

- Discrete delays as states

For more information, see “Finding Linearized Models with Padé Approximation of Delays” on page 4-16.

- Exact linearization provides the following results:

- An exact representation of continuous delays
- An internal representation of discrete delays

These discrete delays do not appear as states in the linearized model but are accounted for as internal delays.

For more information, see “Finding Linearized Models with Exact Delays” on page 4-17.

Blocks with Delays. The delays in your model can arise from any of the following Simulink blocks:

- Transport Delay
- Variable Time Delay
- Variable Transport Delay
- Integer Delay
- Unit Delay

For more information on time delays, see “Time Delays” in the Control System Toolbox documentation.

Finding Linearized Models with Padé Approximation of Delays. To find linearized models with Padé approximations of delays, first adjust the order of the Padé approximation in the Block Parameters window for any block with delay. Then, perform the linearization.

Note To use a Padé approximation for continuous delay blocks, set the `UseExactDelayModel` option of the `linoptions` function to the default setting, `off`.

For more information on Padé approximations, see “Eliminating Time Delays: Padé Approximation” in the Control System Toolbox documentation.

Finding Linearized Models with Exact Delays. You can use block-by-block analytic linearization to find linear models with exact time delays. You can do this in the following ways:

- In the Linearizations Options dialog box, select the **Return linear model with exact delay(s)** option.

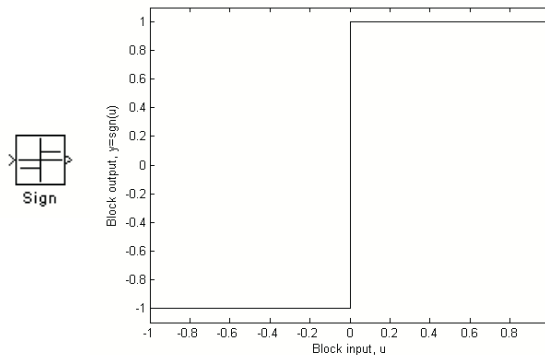
For more information on the Linearization Option dialog box, see “Choosing Linearization Settings and Algorithms” on page 4-9.

- At the command line, set the `linoptions` function option `UseExactDelayModel` to on.

For more information, see the “Linearizing Models with Time Delays” demo listed under the Simulink Control Design Demos in the demos browser.

Blocks with Discontinuities

There are several Simulink blocks that contain discontinuities, such as the Sign block, whose behavior is shown in the following figure.



The very large derivatives that occur at the point of discontinuity can cause problems with linearization. For example, the Sign block has the following linearization

$$D = 0, u \neq 0$$

$$D = \infty, u = 0$$

where D is a state-space matrix, and u is the input signal to the block.

When these blocks are within the linearization path of your model, the resulting linearized model could potentially have very large values. There is no obvious solution to this problem and it is recommended that you remove or replace these blocks. However, when your model operates in a region away from the point of discontinuity, the linearization will be zero. This should not cause any problems, although when the linearizations of several blocks are multiplied together (as in a feedback path) it can cause the linearization of the system to be zero.

When these blocks are outside the linearization path, they can still contribute to the definition of the operating point of the model but will not otherwise affect the linearization. It is safe to use them for reference signals, disturbances, and any other signals and blocks that are not being linearized.

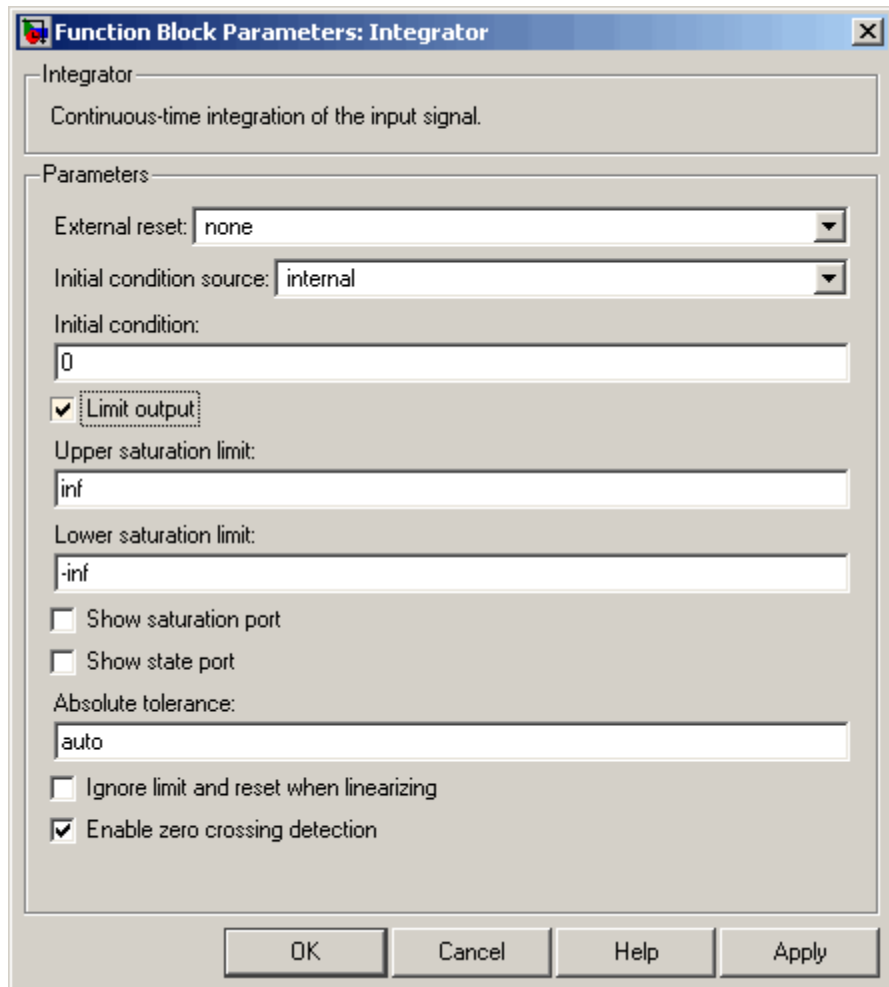
Other examples of blocks with discontinuities include

- Relational Operator blocks
- Relay block
- Logical Operator blocks
- Stateflow blocks
- Quantizer block (has an option to treat as a gain when linearizing)
- Saturation block (has an option to treat as a gain when linearizing)
- Deadzone block (has an option to treat as a gain when linearizing)

Integrator Blocks Near Saturation or a Reset Point

When an Integrator block has an external reset condition or output limitations (saturation) and the model is operating near the point where the Integrator is reset or the output is limited, it might be more meaningful for the linearization to ignore the effect of the saturation or reset. To linearize a model around an operating point that causes the integrator to reset or saturate, select **Ignore**

limit and reset when linearizing in the Integrator block parameters dialog box. Selecting this option causes the linearization to treat this block as unresettable and as having no limits on its output, regardless of the settings of the block's reset and output limitation (saturation) options.

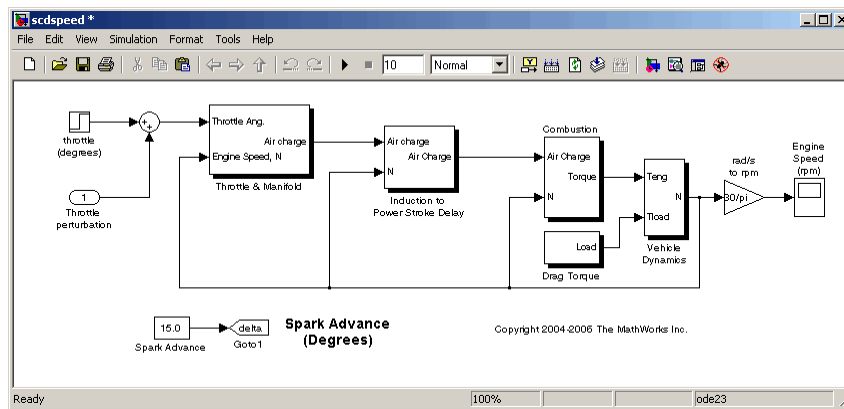


Event-Based Models and Triggered Subsystems

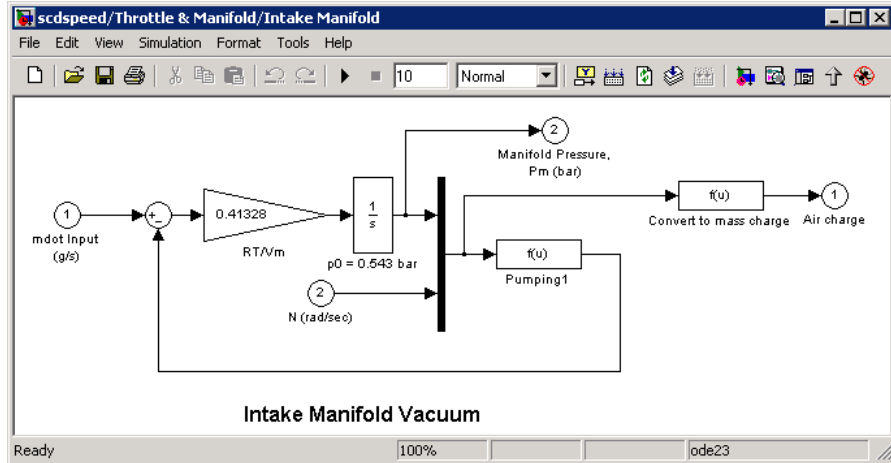
The linearization of triggered subsystems and other event-based models can be particularly difficult because of the system's dependence on previous events. In particular, the execution of a triggered system depends on previous signal events such as zero crossings. Therefore, for linearization, which takes place at a particular moment in time, a trigger event will never happen. Thus, while the event-based dynamics contribute to the definition of the system's operating point, this information is not captured by the list of values of states and inputs that typically describe the operating point for linearization.

Triggered events describe many different systems. One such system is an internal combustion (IC) engine. When an engine piston approaches the top of a compression stroke, a spark is introduced and combustion occurs. The timing of the spark for combustion is dependent on the speed and position of the engine crankshaft. An example of a Simulink model that models this behavior is `engine.mdl` which is included as a demonstration model in the Simulink product.

In `engine.mdl`, triggered subsystems generate events when the pistons reach both the top and bottom of the compression stroke. The linearization will not be meaningful because of the presence of these triggered subsystems. However, you can get a meaningful linearization while still preserving the simulation behavior by recasting the event-based dynamics. For example, you can use curve fitting to approximate the event-based behavior. This is done in `scdspeed.mdl`, a demonstration model included in the Simulink Control Design product, shown in the figure below:



The basic functional approximation in `scdspeed` is included within the Convert to mass charge block inside the subsystem `scdspeed/Throttle & Manifold/Intake Manifold` where a quadratic polynomial is used to approximate the relationship between the Air Charge, the Manifold Pressure, and the Engine Speed.



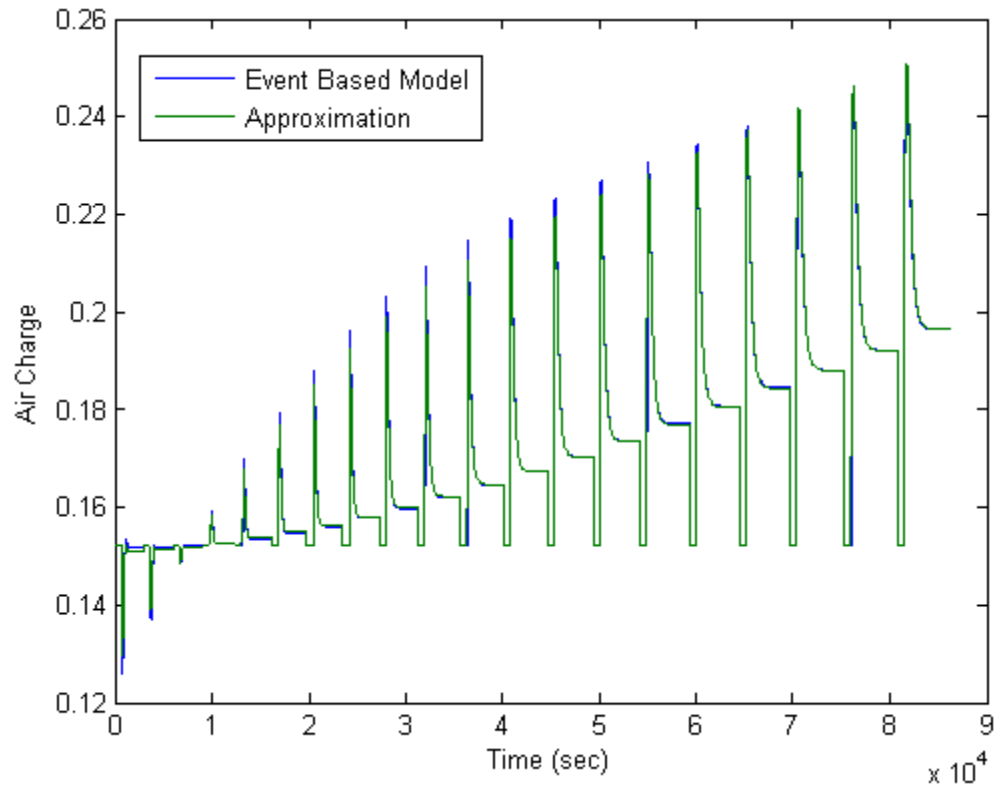
The approximation has the following form:

$$\text{Air Charge} = p_1 \times \text{Engine Speed} + p_2 \times \text{Manifold Pressure} + p_3 \times (\text{Manifold Pressure})^2 + p_4 \times \text{Manifold Pressure} \times \text{Engine Speed} + p_5$$

Simulation data from the original model is used to compute the unknown parameters p_1 , p_2 , p_3 , p_4 , and p_5 using a least squares fitting technique.

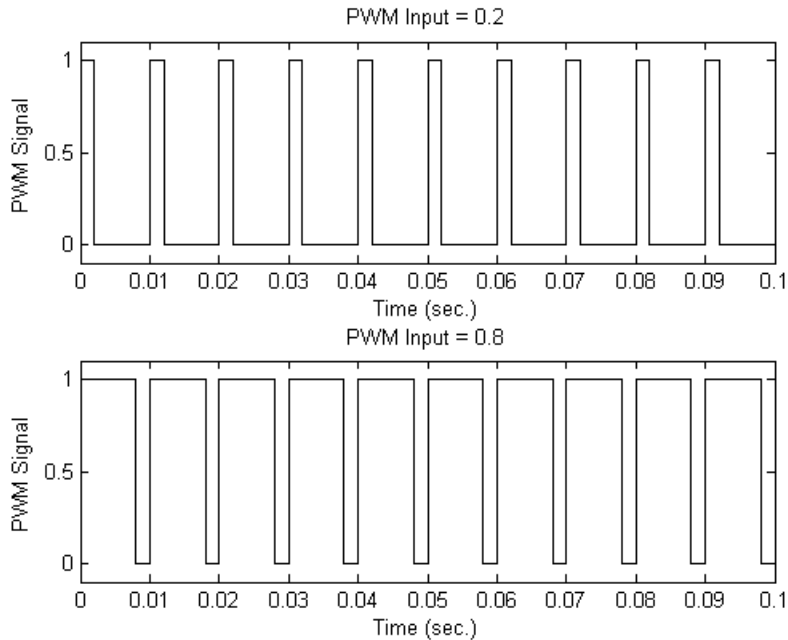
When measured data for internal signals is available, you can use the Simulink Design Optimization software to compute the unknown parameters. This method is outlined in the Simulink Design Optimization demo called Engine Speed Model Parameter Estimation. The demo also shows how to linearize this model and use the linearization to design a feedback controller.

The approximated model can now accurately simulate and linearize the engine from approximately 1500 to 5500 RPM. The following figure shows the comparison between a simulation of the original event-based model, and a simulation of the new approximated model.

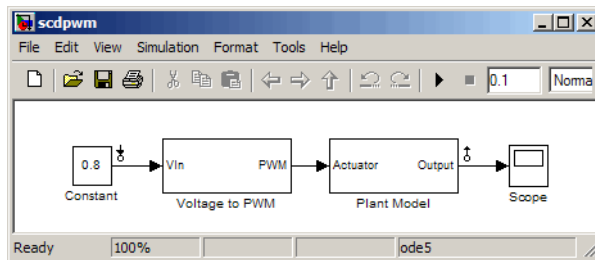


Pulse Width Modulation

Many industrial applications use Pulse Width Modulation (PWM) signals because of their robustness in the presence of noise. The following figure shows two examples of PWM signals. In the first example, a DC voltage of 0.2V is represented by a PWM signal with a 20% duty cycle (a value of 1 for 20% of the cycle, followed by a value of 0 for 80% of the cycle). The average signal value is 0.2V. The second example shows a PWM representation of a 0.8V DC signal, where the duty cycle is 80%.



The model, `scdpwm.mdl`, shown below, converts a constant signal to a PWM signal.

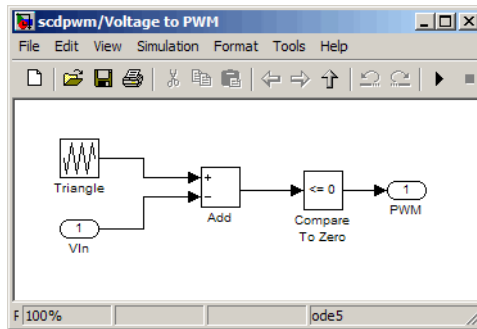


When linearizing a model containing PWM signals there are two effects of linearization you should consider:

- The signal level at the operating point is one of the discrete values within the PWM signal, not the DC signal value. For example, in the model above,

the signal level is either 0 or 1, not 0.8. This change in operating point affects the linearized model.

- The creation of the PWM signal within the subsystem `Voltage to PWM`, shown in the following figure, uses a comparator block, the `Compare to Zero` block. Comparator blocks do not linearize well due to their discontinuities and the nondouble outputs.



A solution to these two problems is to consider removing the PWM block before linearizing the model.

Numerical-Perturbation Linearization

- “What is Numerical-Perturbation Linearization?” on page 4-24
- “Invoking Numerical-Perturbation Linearization” on page 4-25
- “Perturbation Algorithm” on page 4-26
- “Controlling the Results of Numerical-Perturbation Linearization” on page 4-28

What is Numerical-Perturbation Linearization?

An alternative linearization method available for use in the Simulink Control Design software is numerical-perturbation linearization, which computes state-space matrices for the linearized model by numerical perturbation of the *whole system*. The method is relatively quick and simple, although as mentioned in “Options for Linearization Algorithm Method” on page 4-11, it does have some disadvantages.

Numerical-perturbation linearization requires that root-level inport and outport blocks be present in the model. These blocks define the portion of the model that you want to linearize instead of inserting input and output points by right-clicking on the signal lines. Any input, output, or open-loop points on signal lines in the model will be ignored when using numerical-perturbation linearization.

The perturbation is introduced to the system at the root level inport blocks and in the states of the system. The response to the perturbation is measured at the outport blocks. Suggestions for controlling the results of numerical-perturbation linearization to create accurate linearized models are given in “Controlling the Results of Numerical-Perturbation Linearization” on page 4-28

Invoking Numerical-Perturbation Linearization

Prior to Simulink 3.0, numerical-perturbation linearization was the only linearization method available with the Simulink product. Although block-by-block analytic linearization is now the default linearization method, you might choose to use numerical-perturbation linearization if your model is very large or complicated.

To use numerical-perturbation linearization with the Simulink Control Design GUI, select **Tools > Options** while in the **Linearization Task** node of the Control and Estimation Tools Manager and select Numerical-Perturbation from the **Linearization Algorithms** menu.

To use numerical-perturbation linearization with the `linearize` function, set the `LinearizationAlgorithm` option to 'numericalpert' with the `linoptions` function.

```
linopt=linoptions('LinearizationAlgorithm','numericalpert')
```

To linearize the model, type

```
sys=linearize('modelname',op,linopt)
```

where `modelname` is the name of the model being linearized and `op` is the operating point object for the system.

Perturbation Algorithm

The numerical perturbation algorithm involves introducing a small perturbation to the nonlinear model and measuring the response to this perturbation. Both the perturbation and the response are used to create the matrices in the linear state-space model. Changing the size of the perturbations will change the resulting linearized model.

As described in “What Is Linearization?” on page 4-2, a nonlinear Simulink model can be written as a state-space system:

$$\begin{aligned}\dot{x}(t) &= f(x(t)u(t),t) \\ y(t) &= g(x(t)u(t),t)\end{aligned}$$

In these equations, $x(t)$ represents the states of the model, $u(t)$ represents the inputs of the model, and $y(t)$ represents the outputs of the model.

A linearized model of this system is valid in a small region around the operating point $t=t_0$, $x(t_0)=x_0$, $u(t_0)=u_0$, and $y(t_0)=g(x_0,u_0,t_0)=y_0$. Subtracting the operating point values from the states, inputs, and outputs defines a set of variables centered about the operating point:

$$\begin{aligned}\delta x(t) &= x(t) - x_0 \\ \delta u(t) &= u(t) - u_0 \\ \delta y(t) &= y(t) - y_0\end{aligned}$$

The linearized model can be written in terms of these new variables and is usually valid when these variables are small, i.e. when the departure from the operating point is small:

$$\begin{aligned}\delta \dot{x}(t) &= A\delta x(t) + B\delta u(t) \\ \delta y(t) &= C\delta x(t) + D\delta u(t)\end{aligned}$$

The state-space matrices A , B , C , and D of this linearized model represent the Jacobians of the system, as defined in “What Is Linearization?” on page 4-2. To compute the matrices, the states and inputs are perturbed, one at a time, and the response of the system to this perturbation is measured by computing

$\delta \dot{x}$ and δy . The perturbation and response are then used to compute the matrices in the following way

$$A(:,i) = \frac{\dot{x}|_{x_{p,i}} - \dot{x}_o}{x_{p,i} - x_o}, \quad B(:,i) = \frac{\dot{x}|_{u_{p,i}} - \dot{x}_o}{u_{p,i} - u_o}$$

$$C(:,i) = \frac{y|_{x_{p,i}} - y_o}{x_{p,i} - x_o}, \quad D(:,i) = \frac{y|_{u_{p,i}} - y_o}{u_{p,i} - u_o}$$

where

- $x_{p,i}$ is the state vector whose i th component is perturbed from the operating point value.
- x_o is the state vector at the operating point.
- $u_{p,i}$ is the input vector whose i th component is perturbed from the operating point value.
- u_o is the input vector at the operating point.
- $\dot{x}|_{x_{p,i}}$ is the value of \dot{x} at $x_{p,i}$, u_o .
- $\dot{x}|_{u_{p,i}}$ is the value of \dot{x} at $u_{p,i}$, x_o .
- \dot{x}_o is the value of \dot{x} at the operating point.
- $y|_{x_{p,i}}$ is the value of y at $x_{p,i}$, u_o .
- $y|_{u_{p,i}}$ is the value of y at $u_{p,i}$, x_o .
- y_o is the value of y at the operating point.

Linearized models of discrete-time or multirate systems are computed in a similar way. For more information, see “Linearizing Discrete-Time and Multirate Models” on page 4-65.

Note A perturbed value is one that has been changed by a very small amount from the operating point value. The default difference between the perturbed value and the operating point value is $10^{-5} + 10^{-8}|x|$ for numerical-perturbation linearization.

Controlling the Results of Numerical-Perturbation Linearization

Several factors influence the creation of accurate linearized models. “What Is Linearization?” on page 4-2 discusses some of these factors, such as careful selection of operating points. Factors that are particular to numerical-perturbation linearization are presented here, with suggestions for controlling them.

Setting the Perturbation Level. In numerical-perturbation linearization, there are three options for setting the perturbation levels of states and inport blocks:

- You can accept the default perturbation levels. The default perturbation levels for the states are $10^{-5} + 10^{-8}|x|$, where x is a Simulink structure or vector of the operating point values for the states in the model. Similarly, default perturbation levels for the inport blocks are $10^{-5} + 10^{-8}|u|$, where u is a Simulink structure or vector of the operating point values for the inputs in the model.
- You can edit the linearization property `NumericalPertRel` using the `linoptions` function. The value of this property adjusts the perturbations in the following way:
 - The perturbation of the states is
$$\text{NumericalPertRel} + 10^{-3} \times \text{NumericalPertRel} \times |x|.$$
 - The perturbation of the inputs is
$$\text{NumericalPertRel} + 10^{-3} \times \text{NumericalPertRel} \times |u|.$$

When using the Control and Estimation Tools Manager graphical interface, select **Tools > Options** to open the Options dialog, and then select the

Linearization tab-pane. Within the **Linearization** pane, make sure that you have selected Numerical perturbation as the **Linearization algorithm** and then enter a value for **Relative Perturbation level** under **Options for numerical perturbation algorithm**.

- You can provide individual perturbation levels for each state and inport block. These values override the values computed using the NumericalPertRel value. Set the perturbation levels using the linoptions function to edit the linearization properties NumericalXPert and NumericalUPert. To specify the absolute perturbation levels for NumericalXPert and NumericalUPert, you can use the operpoint function to create an operating point object and then edit the operating point values using dot-notation or the set function.

In the Control and Estimation Tools Manager graphical interface, select **Tools > Options** to open the Options dialog box, and then select the **Linearization** tab-pane. In the **Linearization** pane, verify that you have selected Numerical perturbation as the **Linearization algorithm**. Then enter values for **State Perturbation level** and **Input Perturbation level** under **Options for numerical perturbation algorithm**. You can enter either scalars or operating point objects created with the operpoint function. **State Perturbation level** and **Input Perturbation level** values override **Relative Perturbation level** values.

Example: Linearizing a Model Using Numerical-Perturbation at the MATLAB Command Line. The following example illustrates how to linearize a model at the MATLAB command line using numerical perturbation.

- 1 Open the model.

This example uses the scdairframe_reference.mdl model, included with Simulink Control Design product. The model uses a Model block to reference another Simulink model, eom.mdl.

At the MATLAB command line, enter

```
scdairframe_reference
```

to open this model.

- 2 Set Inport and Outport blocks.

Linearization using the numerical perturbation algorithm is between the root level Inport and Outport blocks, rather than input and output points on signal lines. If your model does not already contain Inport or Outport blocks, you need to add them to the points where you want to perturb the model and measure the response.

Note The `scdairframe_reference` model already contains one Inport block and two Outport blocks.

3 Create an operating point object for the model.

There are several possible methods for creating an operating point object. Which one you use depends on the model you are using and the information you have about the operating point. For more information on creating operating points, see “Ways to Create Operating Points” on page 2-9.

In this example, you create a default operating point with the following command:

```
op_point=operpoint('scdairframe_reference')
```

4 Specify the linearization algorithm.

By default, the linearization algorithm is set to block-by-block linearization. To change the algorithm to numerical perturbation you need to create a linearization options object and set the 'LinearizationAlgorithm' field to 'numericalpert', using the following command:

```
options=linoptions('LinearizationAlgorithm','numericalpert')
```

5 Set the perturbation levels.

By default, the state and input perturbation levels are set to

$$1e^{-5} + 1e^{-8}|x|$$

and

$$1e^{-5} + 1e^{-8}|u|$$

respectively, where $|x|$ and $|u|$ are the absolute values of the states and inputs. These values should be sufficient for most applications and you should not typically need to change them. However, if you want to specify individual perturbation values for each state, you can:

- a Create an operating point object, and edit the state values within this object
- b Then, assign these values to the `NumericalXPert` option, using the following commands:

```
state_pert=operpoint('scdairframe_reference');
state_pert.states(1).x=[1e-8;1e-9];
state_pert.states(2).x=1e-7;
state_pert.states(3).x=[1e-7;1e-8];
state_pert.states(4).x=1e-9;
options.NumericalXPert=state_pert;
```

6 Linearize the model.

The following command linearizes the model about the chosen operating point, using the perturbation settings in the linearization options object, and returns the state-space matrices of the linearized model:

```
sys=linearize('scdairframe_reference',op_point,options)
```

Example: Linearizing a Model Using Numerical-Perturbation

in the GUI. The previous example showed how to linearize the `scdairframe_reference.mdl` using Simulink Control Design functions for numerical perturbation. The following example uses the numerical perturbation algorithm to linearize the same model within the Control and Estimation Tools Manager graphical interface.

1 Open the model.

This example uses the `scdairframe_reference.mdl` model, included with the Simulink Control Design product. The model uses a Model block to reference another Simulink model, `eom.mdl`.

At the MATLAB command line, enter

`scdairframe_reference`

to open this model.

2 Set Inport and Outport blocks.

Linearization using the numerical perturbation algorithm relies on perturbing root level Inport and Outport blocks, rather than input and output points on signal lines. If your model does not already contain Inport or Outport blocks, you need to add them to the points where you want to perturb the model, and measure the response.

Note In this example, the `scdairframe_reference` model already contains one Inport block and two Outport blocks.

You should notice that since you will numerically perturb this model using root-level Inport and Outport blocks, you cannot specify any linearization points in the **Analysis I/Os** pane of the **Linearization Task**.

3 Open a linearization task for the model in the Control and Estimation Tools Manager. Then, in the `scdairframe_reference.mdl` model window, select **Tools > Control Design > Linear Analysis**.

This opens the Control and Estimation Tools Manager and creates a task for linearization.

4 Create an operating point object for the model.

There are several possible methods for creating operating point objects. Which one you use depends on the model you are using and the information you have about the operating point. For more information on creating operating points, see “Ways to Create Operating Points” on page 2-9.

This example uses the default operating point for the linearization.

5 Specify the linearization algorithm.

To select numerical perturbation linearization as the algorithm, select **Tools > Options** within the Control and Estimation Tools Manager to open the Options dialog, select the **Linearization** pane in the Options

dialog, and then select Numerical perturbation as the **Linearization algorithm**.

6 Set the perturbation levels.

To use perturbation levels other than the default settings, select **Tools > Options** within the Control and Estimation Tools Manager to open the Options dialog, and then select the **Linearization** pane. Under **Options for numerical perturbation algorithm**, enter perturbation values. The perturbation values can be either scalars, vectors, operating point objects, or Simulink structures of state values.

For this example, enter $1e-9$ in the **State perturbation level** box. This value overrides the state perturbation values computed from the **Relative perturbation level** setting. However, because you have not explicitly specified the **Input perturbation level**, the algorithm still uses the **Relative perturbation level** setting to compute input perturbations.

Note These perturbation values are not the same as the perturbation values used in the previous example.

7 Linearize the model:

- a** Select **Linearization Task** in the pane on the left of the Control and Estimation Tools Manager.
- b** Select the **Operating Points** pane on the right.
- c** Within the **Operating Points** pane, select the operating point that you want to use for the linearization. For this example, there should be only one choice, the default operating point.
- d** Click the **Linearize Model** button to linearize the model around this operating point. The results are plotted in the LTI Viewer.

Handling Special Blocks.

Blocks Containing Discontinuities

Certain blocks, especially those containing discontinuities such as `Saturation` or `Transport Delay`, may not linearize well using numerical perturbation. Although these blocks often have preprogrammed linearizations that are used with block-by-block analytic linearization instead of numerically perturbing them, they are *not* used in numerical-perturbation linearization. As an alternative, you can replace these blocks with an appropriate block before linearizing your model. For example, you might choose to replace a `Saturation` block with a `Gain` block.

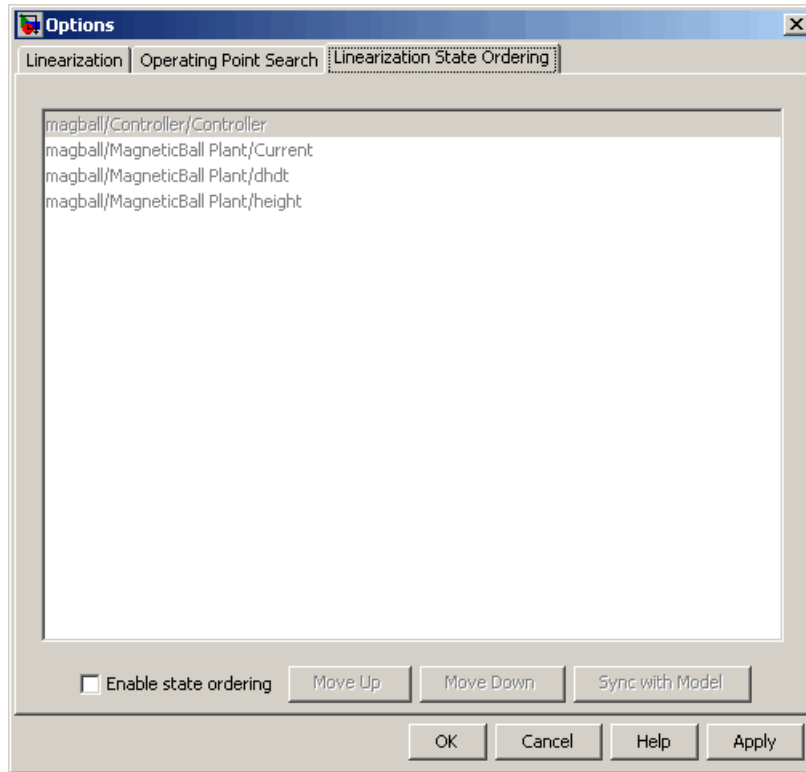
Random Number Blocks

Random Number blocks inside models that reference other models through acceleration using the `Model` block, can also sometimes cause inaccurate numerical perturbation linearization results. Care should be taken when linearizing or computing operating points with model reference models that use these blocks. It is recommended that you set model references to normal mode.

Handling Feedback Loops. “Performing Open-Loop Analysis” on page 4-49 discusses the effect of feedback loops on the results of a linearization. With block-by-block analytic linearization, you can perform open-loop analysis without removing feedback loops. When using numerical-perturbation linearization, the only way to remove the effect of feedback loops is to manually remove them from the model *and* manually force the operating point to remain the same as the original model.

Changing State Ordering in the Linearized Model

In some control applications it may be necessary to order the states of the linearized models. To specify the state ordering in the GUI, select **Tools > Options**, and then click the **Linearization State Ordering** tab. This opens the Linearization Task Options dialog box.



To specify the order of the states, select the **Enable state ordering** check box at the bottom of the tab. Then, use the **Move Up** and **Move Down** buttons to move states to a new position in the list. When you add new states to or remove existing states from the model diagram, click the **Sync with Model** button to update the list.

Note For information on changing state ordering using the command line, see the `linearize` reference page.

Configuring the Linearization of Specific Blocks and Subsystems

In this section...

“Ways to Configure Blocks and Subsystems For Linearization” on page 4-36

“Controlling the Analytic Linearization of Individual Blocks” on page 4-36

“Specifying the Linearization of Blocks and Subsystems” on page 4-37

“Controlling the Block Perturbation Linearization of Individual Blocks” on page 4-40

Ways to Configure Blocks and Subsystems For Linearization

Most Simulink blocks are linear and do not require any special configuration. You can configure the linearization of the following blocks:

- Blocks with discontinuities, by setting any built-in block linearization options available, as described in “Controlling the Analytic Linearization of Individual Blocks” on page 4-36
- Any block, subsystem, or model reference block, by specifying the actual linearization result, as described in “Specifying the Linearization of Blocks and Subsystems” on page 4-37.
- Blocks that linearize using numerical perturbation, by defining perturbation levels, as described in “Controlling the Block Perturbation Linearization of Individual Blocks” on page 4-40

Controlling the Analytic Linearization of Individual Blocks

You can control the linearization of several types of blocks by adjusting options in the Block Parameters window. For example, you can change the order of the Padé approximation used in the Transport Delay block or select the **Treat as gain when linearizing** option in the Saturation block. For more information on controlling the linearization of individual blocks, see the reference page for each block.

Locating Blocks in Your Model That Do Not Support Analytic Linearization

You can view a list of the blocks in your model that automatically linearize using numerical perturbation instead of analytic linearization because they do not contain analytic Jacobians. View this list in the diagnostic messages tab for your linearization results in the Control and Estimation Tools Manager. For more information on this list, see “Diagnosing Blocks” on page 4-82 in the Simulink Control Design documentation.

Specifying the Linearization of Blocks and Subsystems

You can specify the linearization of blocks, subsystems, or model references without replacing any blocks in your model using either:

- MATLAB Expressions, as described in “Specifying the Linearization Using a MATLAB Expression” on page 4-37
- Configuration Functions, as described in “Specifying the Linearization Using a Configuration Function” on page 4-38

Specifying linearization does not affect the simulation of your model. You can specify how blocks linearize when you use the block-by-block analytic linearization method (the default). To set a linearization method, see “Choosing Linearization Settings and Algorithms” on page 4-9.

Specifying the Linearization Using a MATLAB Expression

To specify the linearization of a block, subsystem, or model reference using a MATLAB expression:

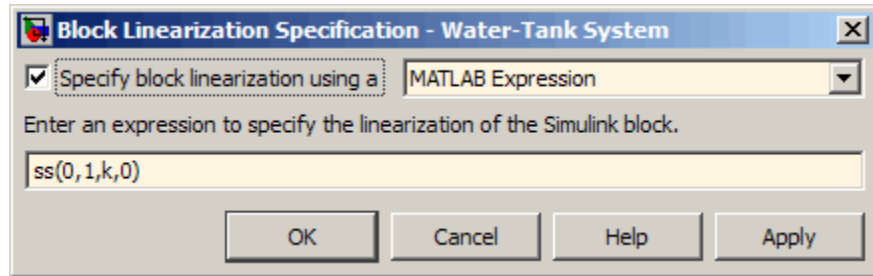
- 1** Right-click the block in the Simulink model, and select **Linear Analysis > Specify Linearization**.

The Block Linearization Specification dialog box opens.

- 2** Check **Specify block linearization using a**.

- 3** In the **Specify block linearization using a** drop-down, select **MATLAB Expression**.

- 4 In the text field, enter an expression to specify the linearization. This expression must return one of the following results:
- Linear model in the form of a D-matrix
 - Control System Toolbox LTI object
 - Robust Control Toolbox uncertain state space or uncertain real object (requires Robust Control Toolbox™ software)



Then, click **OK**.

When you linearize the model, the expression you enter follows the resolution rules of normal block, as described in “Resolving Symbols”.

For an example of specifying the linearization of a block using a MATLAB expression, see the Specifying Custom Linearizations for Simulink Blocks demo.

Specifying the Linearization Using a Configuration Function

You can specify the linearization of a block, subsystem, or model reference using a configuration function. The function must:

- Contain a single input argument `blockdata`, which is a structure with the following fields:
 - `BlockName` is the name of the Simulink block with the specified linearization.
 - `Parameters` is a structure array containing the evaluated values for the block. Each element of the array has the fields 'Name' and 'Value',



which contain the name and evaluated value, respectively, for the parameter.

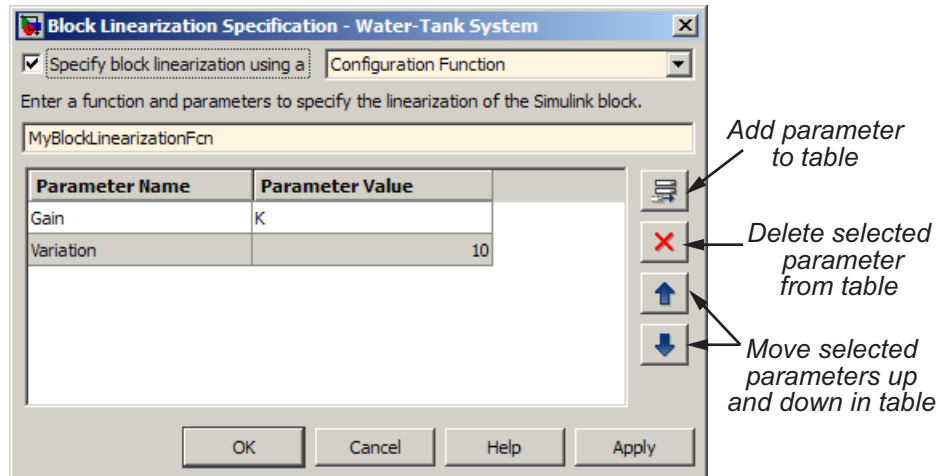
- Inputs is an array of input values.
- ny is the number of output channels of the block linearization.
- nu is the number of input channels of the block linearization.
- Returns one of the following results:
 - Linear model in the form of a D-matrix
 - Control System Toolbox LTI object
 - Robust Control Toolbox uncertain state space or uncertain real object (requires Robust Control Toolbox software)

To specify the linearization using a configuration function:

- 1** Right-click the block in the Simulink model, and select **Linear Analysis > Specify Linearization**.

The Block Linearization Specification dialog box opens.

- 2** Check **Specify block linearization using a**.
- 3** In the **Specify block linearization using a** drop-down, select **Configuration Function**.
- 4** In the text field, enter a function to specify the linearization.
- 5** In the table, add each parameter from your function and a corresponding parameter value. Click  to add a new parameter to the table. Click  to delete a row from the table. Use the arrows to move parameters up and down in the list.



Then, click **OK**. The parameters and values you specify automatically populate the Parameters field of the configuration function input argument structure `blockdata`.

Controlling the Block Perturbation Linearization of Individual Blocks

To set the perturbation size of any blocks that linearize using block perturbation, see “Changing Perturbation Size” on page 4-41. To locate blocks in your model that linearize using block perturbation, see “Locating Blocks in Your Model That Do Not Support Analytic Linearization” on page 4-37.

You can also control the block perturbations of the following blocks using pre-configurations available in the block dialogs:

- “Blocks Containing Two Inputs” on page 4-41
- “Model Reference Blocks” on page 4-42
- “Blocks with Nondouble Data Types” on page 4-42

Changing Perturbation Size

Changing the size of the perturbations changes the linearization results. The default perturbation size is $10^{-5}(1+|x|)$, where x is the operating point value of the state or input being perturbed. For example, to change the perturbation size of the states in the Magnetic Ball Plant block in the `magball` model to

$10^{-7}(1+|x|)$, type

```
blockname='magball/Magnetic Ball Plant'
set_param(blockname, 'StatePerturbationForJacobian', '1e-7')
```

To change the perturbation size of the input of the Magnetic Ball Plant block to $10^{-7}(1+|u|)$, where u is the input signal level, follow these steps:

- 1 Get the block's port handles:

```
ph=get_param('magball/Magnetic Ball Plant', 'PortHandles')
```

- 2 Next, get the inport:

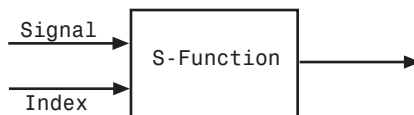
```
pin=ph.Inport(1)
```

- 3 Finally, set the perturbation level for this inport:

```
set_param(pin, 'PerturbationForJacobian', '1e-7')
```

Blocks Containing Two Inputs

If there is more than one inport, you can choose to assign a different perturbation level to each. The following figure shows an S-Function block with two input signals, the actual signal and an index variable. To avoid perturbing the index signal, you can assign a perturbation level of 0 to this inport.



Model Reference Blocks

When linearizing model reference blocks with accelerated mode, the Simulink Control Design software automatically uses block perturbation. If you instead want to linearize these referenced models using block-by-block analytical linearization, then change the block mode from accelerated to normal.

Note Model blocks with multiple sample times and accelerated mode cannot be linearized using block perturbation. To linearize blocks with multiple sample times, you must set the block mode to normal.

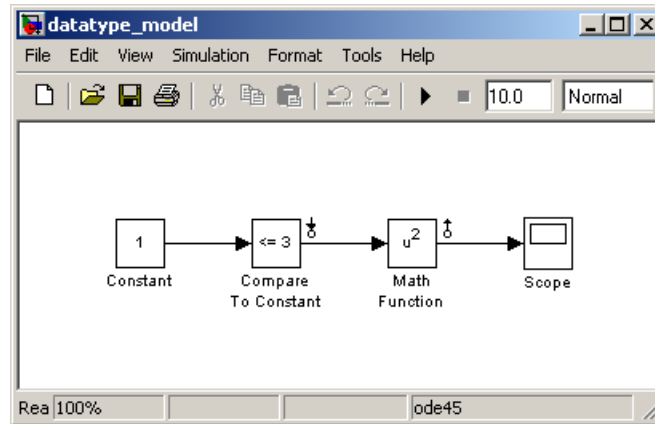
Note If your model contains multiple model blocks referencing the same Simulink model, you must set all of the blocks to accelerator mode.

Blocks with Nondouble Data Types

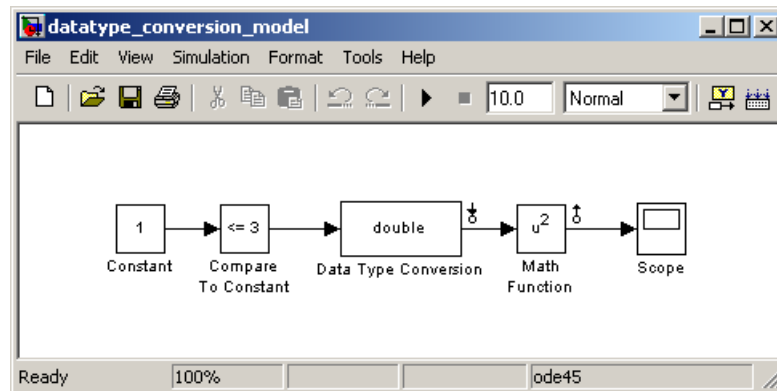
Blocks that have nondouble data type signals as either inputs or outputs, and which do not have a preprogrammed exact linearization, automatically linearize to zero as they cannot be numerically perturbed. For example, many logical operator blocks have Boolean outputs and therefore linearize to 0.

Workarounds for Blocks with Nondouble Data Types

To work around the problem of blocks with nondouble data types linearizing to zero, you can use a Data Type Conversion block. This block has a preprogrammed exact linearization, to convert your signals to doubles before linearizing the model. The following example illustrates this concept. The model in this example is configured to linearize the Square block at an operating point where the input is 1. The resulting linearized model should be 2, but the input to the Square block is Boolean and the linearization is zero.



However, by inserting a Data Type Conversion block before the linearization input point, you can make the input signal to the Square block a double. Thus, the linearized model gives the correct response of 2.



Overriding Nondouble Data Types

When you linearize a model that contains nondouble data types but still runs correctly in full double precision, you can choose to override all data types with doubles. To perform this override, in the model window select **Tools > Fixed-Point > Fixed-Point Tool** from the menu. This selection opens the Fixed-Point Settings window. Within this window select **True**

doubles from the **Data type override** menu. Now, when you linearize and simulate the model, it uses doubles for all data types.

Note This method does not work when the model relies on other data types in its algorithm, such as relying on integer data types to perform truncation from floats.

Selecting Inputs and Outputs for the Linearized Model

In this section...

“What are Linearization Points?” on page 4-45

“Inserting Linearization Points” on page 4-46

“Removing Linearization Points” on page 4-48

“Performing Open-Loop Analysis” on page 4-49

“Inspecting Analysis I/Os” on page 4-52

What are Linearization Points?

Before linearizing the model, you must first configure the model diagram. This involves selecting input and output linearization points (also called analysis I/Os), and then, if necessary, inserting open-loop points for performing open-loop analysis. You can inspect the selected linearization points using the **Analysis I/Os** pane within the **Linearization Task** node of the Control and Estimation Tools Manager.

A linearization input point defines an input to the linearized model while a linearization output point defines an output of the linearized model. Additionally, when the linearized models are computed using numerical perturbation, an input point is the point on the diagram where the small perturbation to the input signal is introduced and an output point is the point on the diagram where the small perturbation to the output signal is measured. Linearization input and output points are not the same as *operating points* which define the state of the model at the point of linearization.

The region between the input and output points defines the portion of the model that you want to linearize, unless a feedback loop feeds the output signal back into another section of the model. In some cases you might want to remove the effect of a feedback signal. For example, you might want to linearize only the plant model within a feedback control loop. When such a feedback loop is present, you can remove the effect of the loop without manually breaking signal lines by inserting an open-loop point. Instructions for inserting open-loop points are in “Performing Open-Loop Analysis” on page 4-49.

You can use the Simulink Control Design software to linearize the whole model, or any blocks or subsystems within the model. To define the system you are linearizing, place linearization points before and after it in the model diagram.

Note Linearization input and output points should not be confused with Simulink Inport and Outport blocks. Input and output points define linear analysis inputs and outputs, while Inport and Outport blocks define the operating point of the system.

Inserting Linearization Points

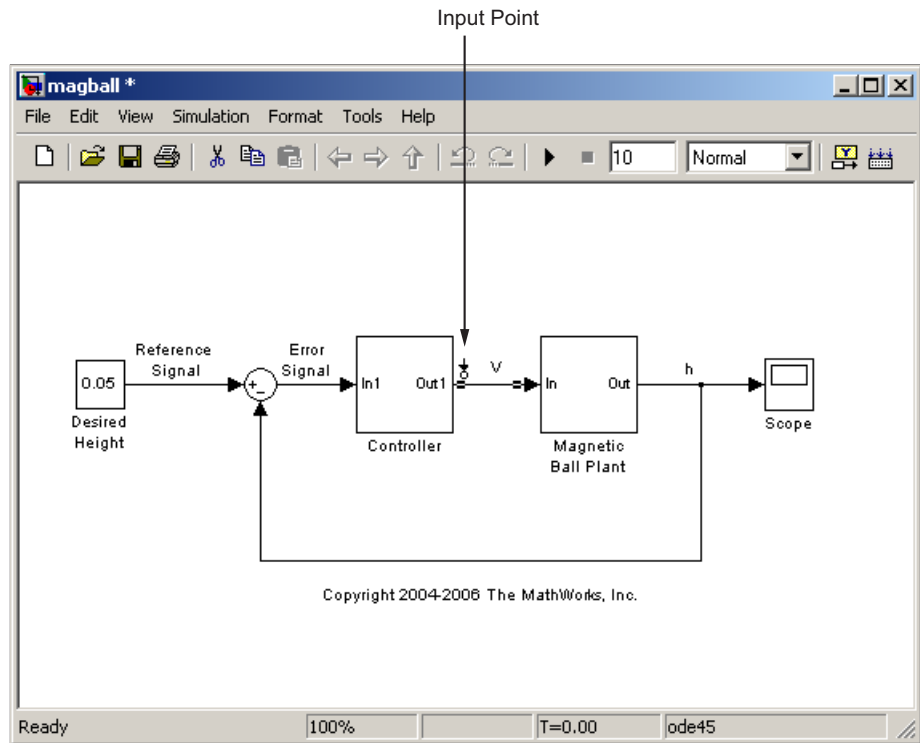
This section continues the magball example from “Extracting Operating Points From Simulation” on page 2-18. At this stage in the example, a linearization task has been created and operating points have been specified.

For a definition of linearization points, also known as analysis I/Os, see “What are Linearization Points?” on page 4-45.

In the magnetic ball model, the nonlinearities are all contained within the Magnetic Ball Plant. To linearize this subsystem:

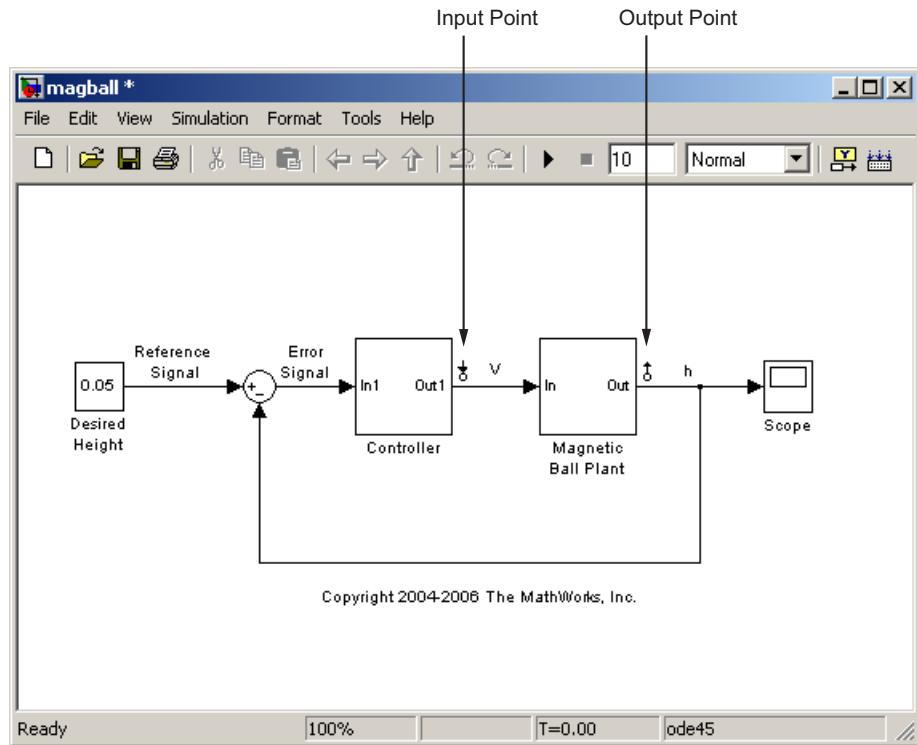
- 1 Select the **Linearization Task** node within the Control and Estimation Tools Manager.
- 2 On the magball model diagram, position the mouse on the signal line between the Magnetic Ball Plant and the Controller. Right-click and select **Linearization Points > Input Point** from the menu.

The model diagram now contains a small arrow pointing toward a circle just above the signal line between the Controller and the Magnetic Ball Plant, as in the following figure. This symbol indicates an input point for linearization has been placed there.



- 3 Right-click the signal line after the Magnetic Ball Plant and select **Linearization Points > Output Point** from the menu.

A small arrow pointing away from a circle on the signal line appears after the Magnetic Ball Plant indicating an output point for linearization has been placed there. The diagram should now look like that in the following figure.



Note All linearization points are referenced to the output port of the block the signal line originates from. For example, placing a linearization point anywhere on the feedback loop in the figure above will result in a linearization point at the output of the Magnetic Ball Plant block.

To inspect the linearization points in the Control and Estimation Tools Manager, see “Inspecting Analysis I/Os” on page 4-52.

Removing Linearization Points

To remove a linearization point from a signal line in your model, repeat the same process as for inserting a linearization point. For example, to remove an input point, right-click the signal line containing the input point and select

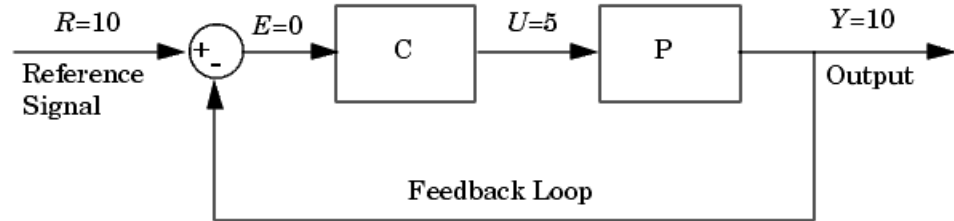
Linearization Points > Input Point from the menu. The input point disappears from the diagram.

Performing Open-Loop Analysis

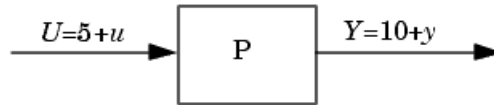
Due to the presence of feedback loops in a model, the input and output points might not completely define the portion of the model you want to linearize. In these cases, to remove the effect of signals feeding back into the portion of the model you are linearizing, you might choose to insert an open-loop point. “What Is Open-Loop Analysis?” on page 4-49 discusses the concepts behind open-loop analysis. “Inserting Loop Openings” on page 4-51 continues the magball example by inserting open-loop linearization points in the magball model. “Open-Loop Analysis Using Functions” on page 5-10 in the online documentation gives methods for assigning open-loop points in Simulink models using functions.

What Is Open-Loop Analysis?

Many control systems contain feedback loops. An example of such a system is shown in this figure.

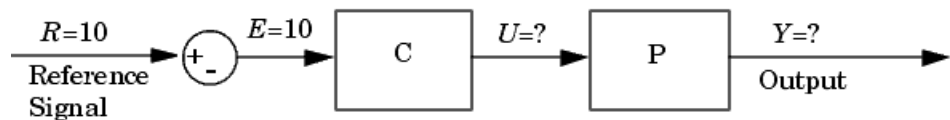


The model represented in this figure is at equilibrium. Consider linearizing the plant, P, about this equilibrium operating point by changing the input signal, U , by a small amount, u , and measuring the change in the output signal, y . The portion of the system that you want to linearize is shown in the following figure.



However, due to the presence of the feedback loop, the change in the output signal will feed back into the controller, C , and then into the plant. This affects the behavior of the system you are linearizing. In fact, if C and P were linear, the linearized model between U and Y would be $\frac{P(s)}{1+C(s)P(s)}$ rather than $P(s)$.

You could manually remove the feedback signal from the model in an attempt to resolve this issue. However, as shown in the following figure, this changes the operating point of the system since the error signal, E , is now equal to the reference signal, R . Linearizing about this new operating point would change the linearization results. Of course, this only makes a difference for nonlinear models. When the model is already linear, it has the same form regardless of the operating point.

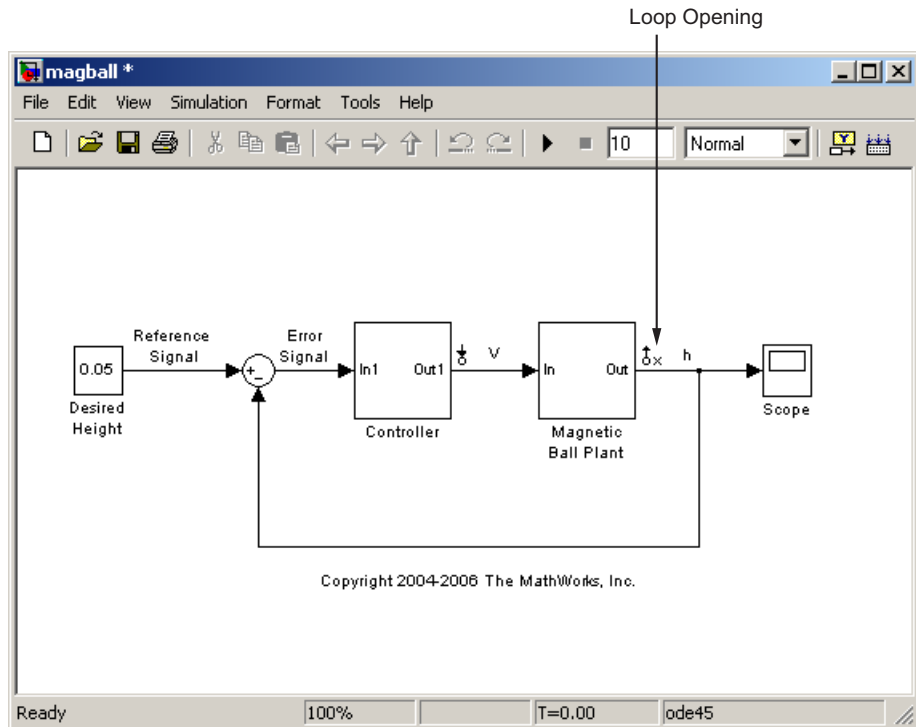


When linearizing Simulink models, label an input or output point as open loop. Doing so ensures that the output signal is not fed back into the model but keeps the operating point the same. In other words, in the linear case, you would compute $P(s)$ rather than $\frac{P(s)}{1+C(s)P(s)}$.

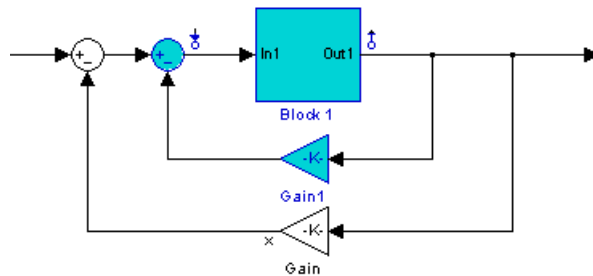
Inserting Loop Openings

This section continues the example from “Inserting Linearization Points” on page 4-46. At this stage in the example, a linearization task has already been created for the model, operating points have been specified, and linearization input and output points have been inserted.

To linearize only the Magnetic Ball Plant, right-click the signal line containing the output point and select **Linearization Points > Open Loop**. This inserts a small x next to the output point in the diagram, representing a loop opening.



Note You do not need to place a loop opening in the same place as an input or output point. For example, in the following figure the highlighted blocks are included in the linearization. The loop opening is placed after the gain on the outer feedback loop, which removes the effect of this loop from the linearization. Placing a loop opening at the same place as the output point would have removed the effect of the inner loop from the linearization as well.



Inspecting Analysis I/Os

This section continues the example from “Inserting Loop Openings” on page 4-51. At this stage in the example, a linearization task has already been created for the model, operating points have been specified, linearization input and output points have been inserted, and loop openings have been created.

To view the linearization points, click the **Analysis I/Os** tab in the **Linearization Task** node in the Control and Estimation Tools Manager, as shown in the following figure. Use this pane to inspect and make changes to your linearization points.

Make points active or inactive. Points that are inactive will not be used in the linearization.

Change the type of a linearization point

Add or remove loop openings

The screenshot displays the 'Control and Estimation Tools Manager' window. The 'Analysis I/Os' tab is active, showing a table of linearization points. The table has columns for 'Active', 'Block Name', 'Output Port', 'Signal Name', 'Configuration', and 'Open Loop'. Two points are listed: 'magball/Controller' (Input) and 'magball/Magnetic Ball Plant' (Output). Below the table are buttons for 'Highlight Selected Signal', 'Refresh Signal Names', and 'Delete Selected I/O'. At the bottom, there is a 'Linearize Model' button and a checkbox for 'Plot linear analysis result in a step response plot'. A status bar at the very bottom contains the message: 'An operating point Operating Point at t = 10 has been added to the node Operating Points.'

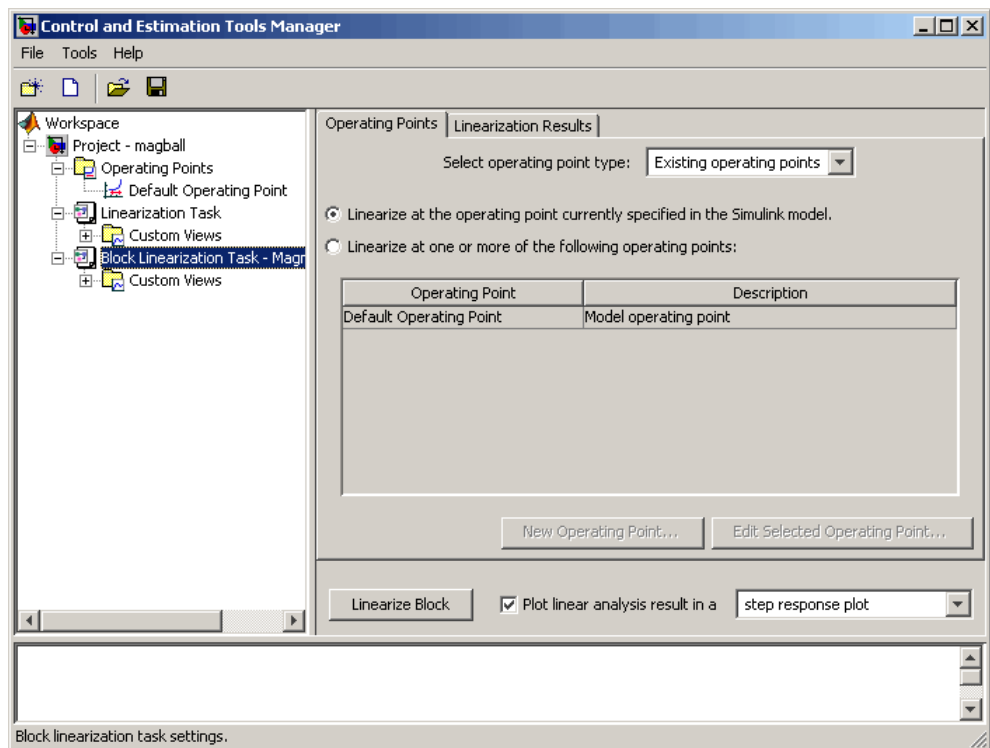
Active	Block Name	Output Port	Signal Name	Configuration	Open Loop
<input checked="" type="checkbox"/>	magball/Controller	1		Input	<input type="checkbox"/>
<input checked="" type="checkbox"/>	magball/Magnetic Ball Plant	1	Operating F	Output	<input checked="" type="checkbox"/>

Note You cannot make changes to the **Analysis I/Os** or perform open-loop analysis when you select **Numerical Perturbation** as the **Linearization Algorithm** in the Linearization Task Options window (accessed by selecting **Tools > Options** from the Control and Estimation Tools Manager window). For more information, see “How to Choose Linearization Settings and Algorithms” on page 4-9.

Linearizing a Block

With the Simulink Control Design software you can also linearize a single block in a Simulink model. To do this, right-click the block and select **Linearize Block** from the context menu.

This adds a **Block Linearization Task** node to the project tree as shown in the following figure.



To complete the linearization, specify an operating point in the same way as when linearizing models. See “Linearizing the Model” on page 4-57 for details. Then, click the **Linearize Block** button in the **Operating Points** pane of the **Block Linearization Task** node. You do not need to choose linearization input and output points because the inports and outports of the block are used. If you do have linearization input and output points in your

model, they will be ignored. You cannot linearize an individual block using numerical-perturbation linearization (when Numerical perturbation is selected as the **Linearization Algorithm** parameter).

Note To be linearized, an individual block must contain at least one data inport and output. Blocks from products based on the Simscape platform and SimPowerSystems™ blocks have connection ports instead of inports and outputs. Thus, they cannot be individually linearized.

Linearizing the Model

In this section...

“Linearizing at an Operating Point” on page 4-57

“Creating Other Types of Linear Models” on page 4-64


“Linearizing Discrete-Time and Multirate Models” on page 4-65

Linearizing at an Operating Point

You can use `linearize` around operating points. Refer to “What Are Operating Points?” on page 2-2 for more information on the role of operating points in linearization.

This section contains the following topics:

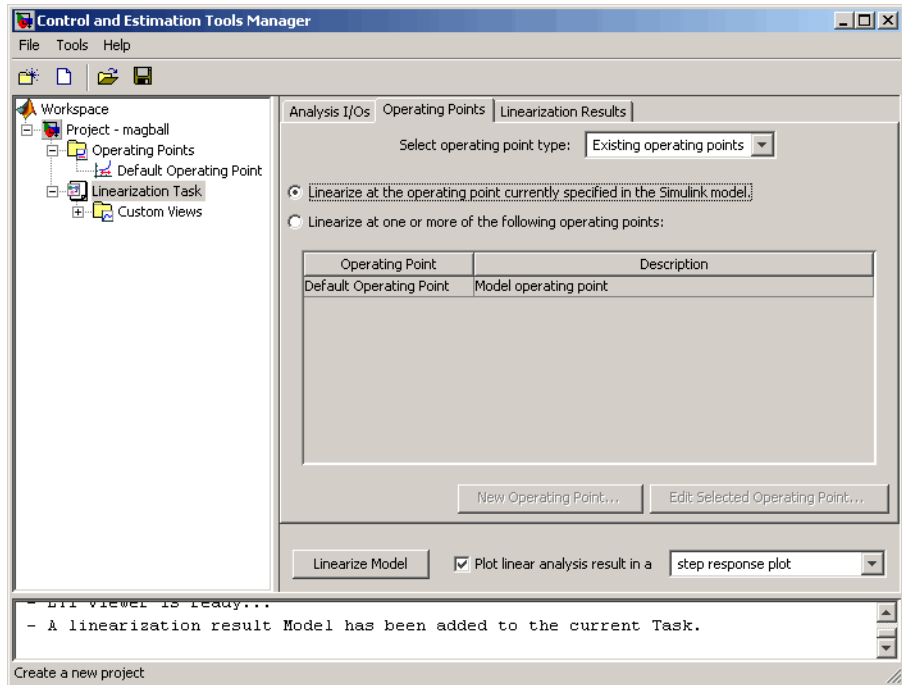
- “Linearizing at a Simulink Model Operating Point” on page 4-57
- “Linearizing at Captured Operating Points” on page 4-58
- “Linearizing at Specified Simulation Times” on page 4-61
- “Linearizing at Simulation Events” on page 4-63

Tip To automatically generate MATLAB code that linearizes your model as specified in the Control and Estimation Tools Manager, click  or select **File > Generate MATLAB Code**.

Linearizing at a Simulink Model Operating Point

To linearize around the operating point in the Simulink model:

- 1** Select the **Operating Points** tab in the **Linearization Task** node.
- 2** Select the **Linearize at the operating point currently specified in the Simulink model** option button. This button is selected by default.



3 Click **Linearize Model**. The software does the following:

- Simulates the model
- Computes the operating point of the model, including the nontrimmable states
- Linearizes around that operating point
- Adds the linearization result, labeled **Model**, to the **Linearization Task** node

Linearizing at Captured Operating Points

You can linearize around operating points that you captured in the **Operating Points** node.

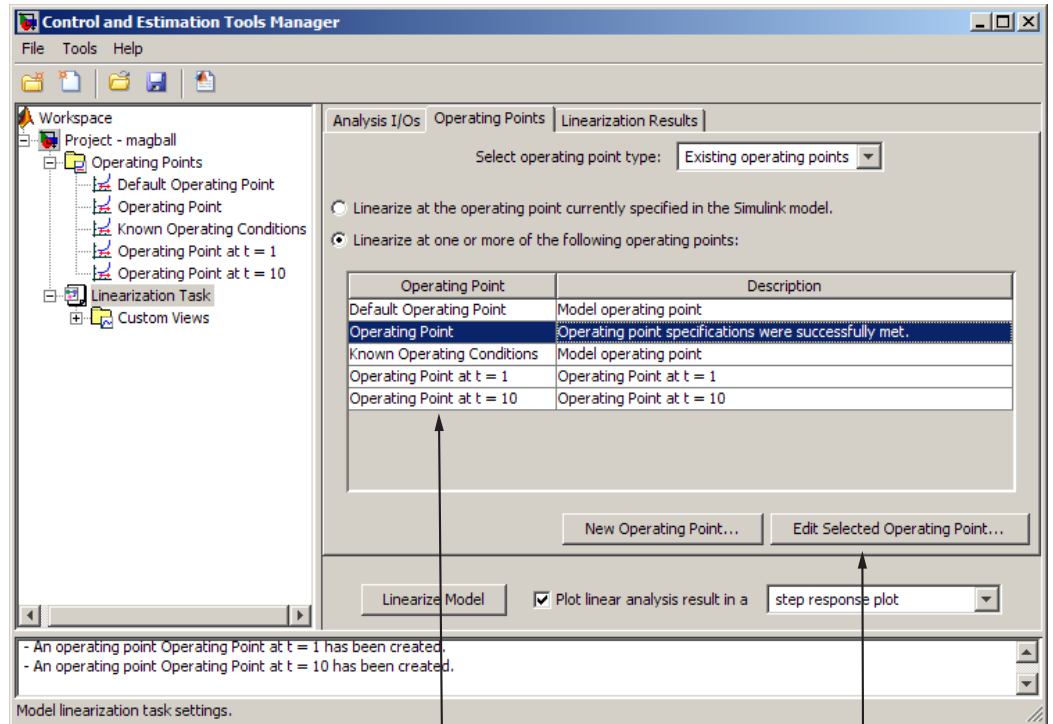
This section continues the example from “Inspecting Analysis I/Os” on page 4-52. At this stage in the example, a linearization task has been created,

operating points have been specified, and linearization points have been inserted.

In this step of the linearization task, you linearize the model using one of the operating points that you created in the project workspace. For information about additional linearization option, see “Linearizing the Model” on page 4-57.

Click the **Operating Points** tab in the **Linearization Task** node to select one or more operating points for the linearization. To linearize the magnetic ball model around the operating point computed from partial specifications, first select the **Linearize at one or more of the following operating points** option button. Then, select **Operating Point** in the list, as shown in the following figure.

4 Exact Linearization Using the GUI



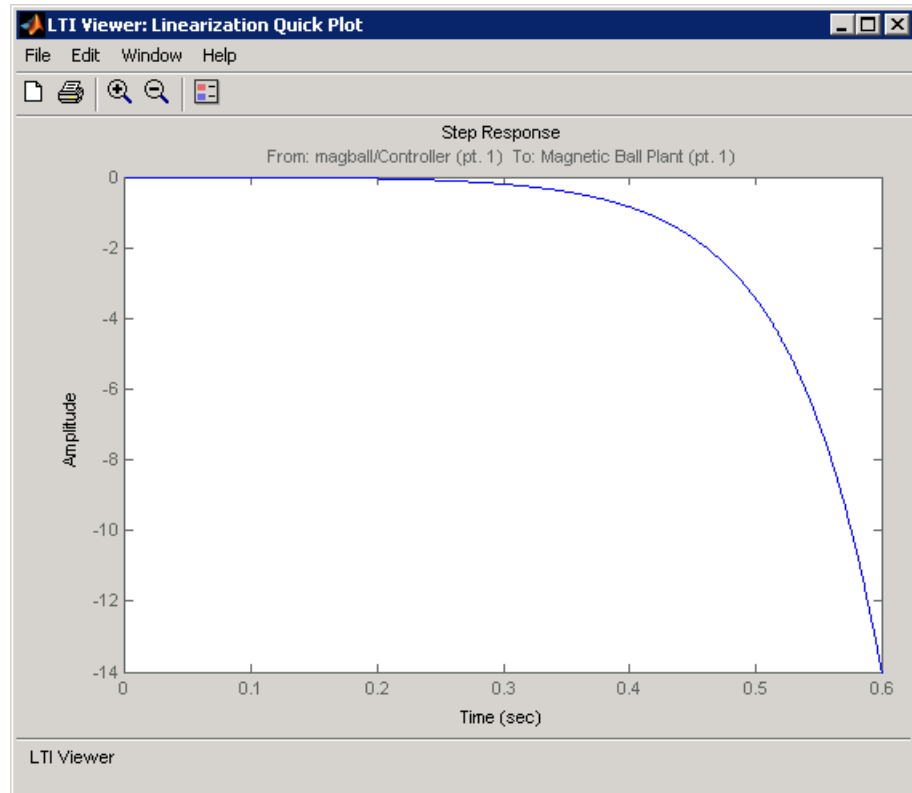
List of operating points available in project workspace

View and edit details of the selected operating point

To compute the linearized model:

- 1 In the menu to the right of the **Linearize Model** button, select from one of the nine options for displaying the results of the linearization. Clear the check box next to this menu when you do not want to display the results.
- 2 Click the **Linearize Model** button at the bottom of the pane. The linearized model is computed and displayed in the LTI Viewer.

The following figure shows the step response for the linearized magball model.



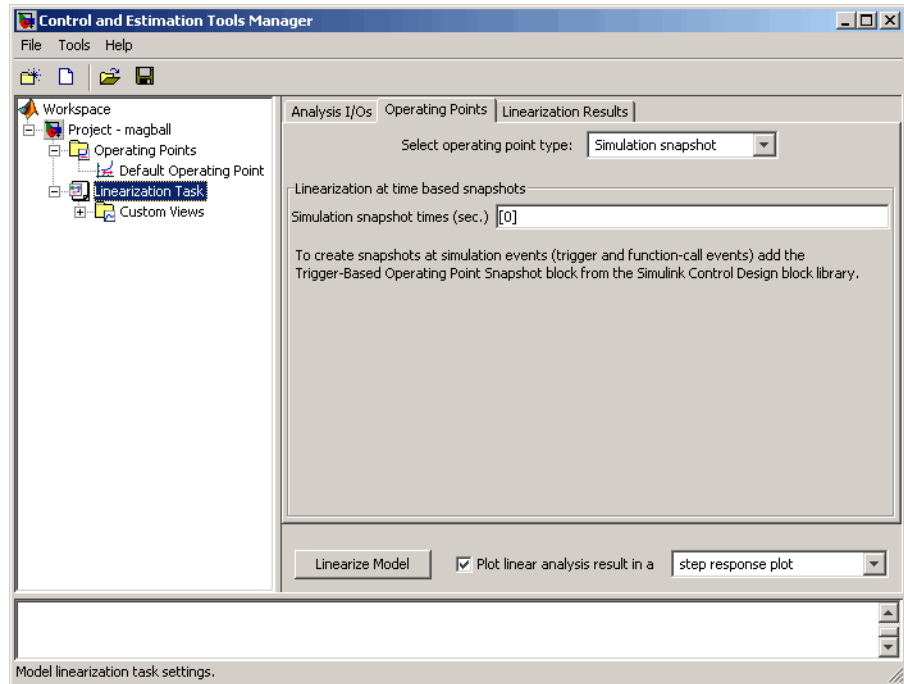
Linearizing at Specified Simulation Times

You can linearize around operating points extracted from a simulation of your model at specified times, such as when the simulation reaches a steady state solution.

To linearize around one or more simulation times:

- 1 Select the **Operating Points** tab in the **Linearization Task** node.

2 From the **Select operating point type** list, select **Simulation snapshot**.



3 Enter a vector of one or more times in the **Simulation snapshot times (sec.)** field. For example, enter [1 , 10] to compute operating points at $t=1$ and $t=10$.

4 Click **Linearize Model**. The software does the following:

- Simulates the model
- Extracts the specified operating points
- Linearizes around these operating points
- Adds the linearization result, labeled **Model**, to the **Linearization Task** node

Linearizing at Simulation Events

You can linearize around operating points extracted from a simulation of your model at one or more of the following simulation events:

- Trigger-based events
- Function-call events

For more information about modeling events in Simulink, see “Creating Conditional Subsystems” in the *Simulink User’s Guide*.

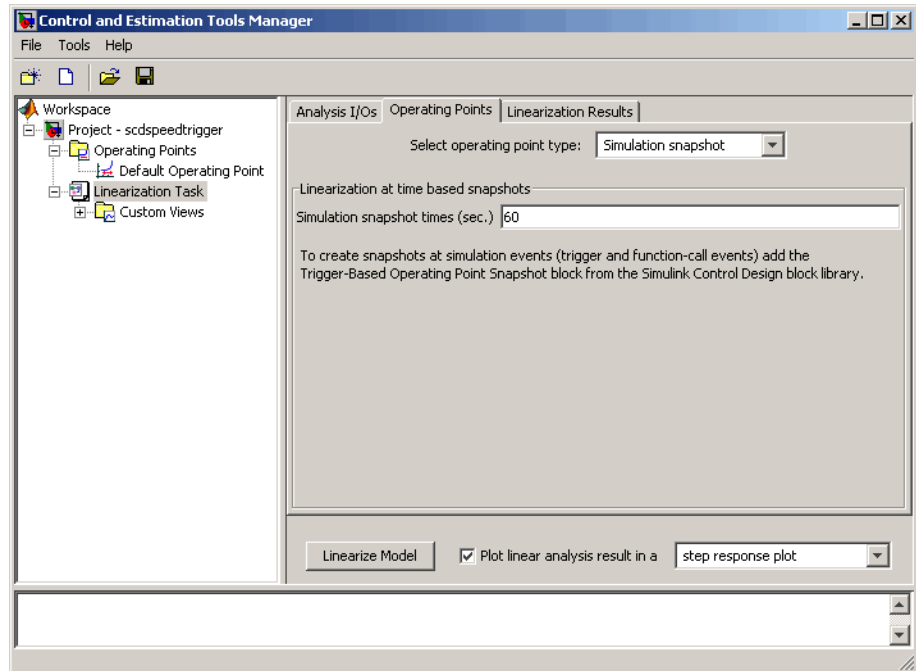
The Simulink Control Design software linearizes around the operating points of all simulation events within a specified simulation time. The linearization takes into account all states in the model operating point.

To linearize around one or more simulation events:

- 1** Add a Trigger-Based Operating Point Snapshot block to your model.

This block is in the Simulink Control Design block library. The model in the Trigger-Based Operating Point Snapshot demo shows the use of this block.

- 2** Select the **Operating Points** tab in the **Linearization Task** node.
- 3** From the **Select operating point type** list, select **Simulation snapshot**.
- 4** Enter a scalar value that specifies the simulation end time in the **Simulation snapshot times (sec.)** field.



5 Click **Linearize Model**. The software does the following:

- Simulates the model for the specified duration
- Extracts the operating points at each simulation event
- Linearizes around these operating points
- Adds the linearization result, labeled **Model**, to the **Linearization Task** node

Creating Other Types of Linear Models

In addition to creating simple transfer functions using the input and output points, you can create more sophisticated linearized models using some of the other options in the **Linearization Points** menu.

- **Input-Output Point**, an input point immediately followed by an output point. This is useful for measuring sensitivity to output disturbances

- **Output-Input Point**, an output point immediately followed by an input point. This is useful for robust control. Use the resulting transfer function in mu analysis of your system.
- **Open Loop**, discussed in “Performing Open-Loop Analysis” on page 4-49
- **Output Constraint**, discussed in “Constraining Outputs” on page 2-27

Linearizing Discrete-Time and Multirate Models

The linearization method is the same for models containing discrete-time states or several different sample times. However, you can choose to adjust the **Linearization sample time** in the **Linearization** options pane. By default, this parameter is set to -1, in which case the software linearizes at the slowest sample rate in the model. To create a linearized model with a different sample time, enter a new value in the dialog box. A value of 0 gives a continuous-time model.

To change the method that Simulink Control Design software uses for converting a multirate model to a single-rate model, change the **Rate conversion method** in the Options dialog box.

For more information, and examples, on methods and algorithms for rate conversions and linearization of multirate models, see the “Linearization of Multi-Rate Models” and “Rate Conversion Method Selection for Linearization” demos listed under the Simulink Control Design Demos in the demos browser.

Viewing Linearization Results

In this section...
“Using the LTI Viewer” on page 4-66
“Displaying Linearization Results” on page 4-66
“Highlighting Blocks in the Linearization” on page 4-68
“Inspecting the Linearization Results Block by Block” on page 4-69
“Comparing the Results of Multiple Linearizations” on page 4-71

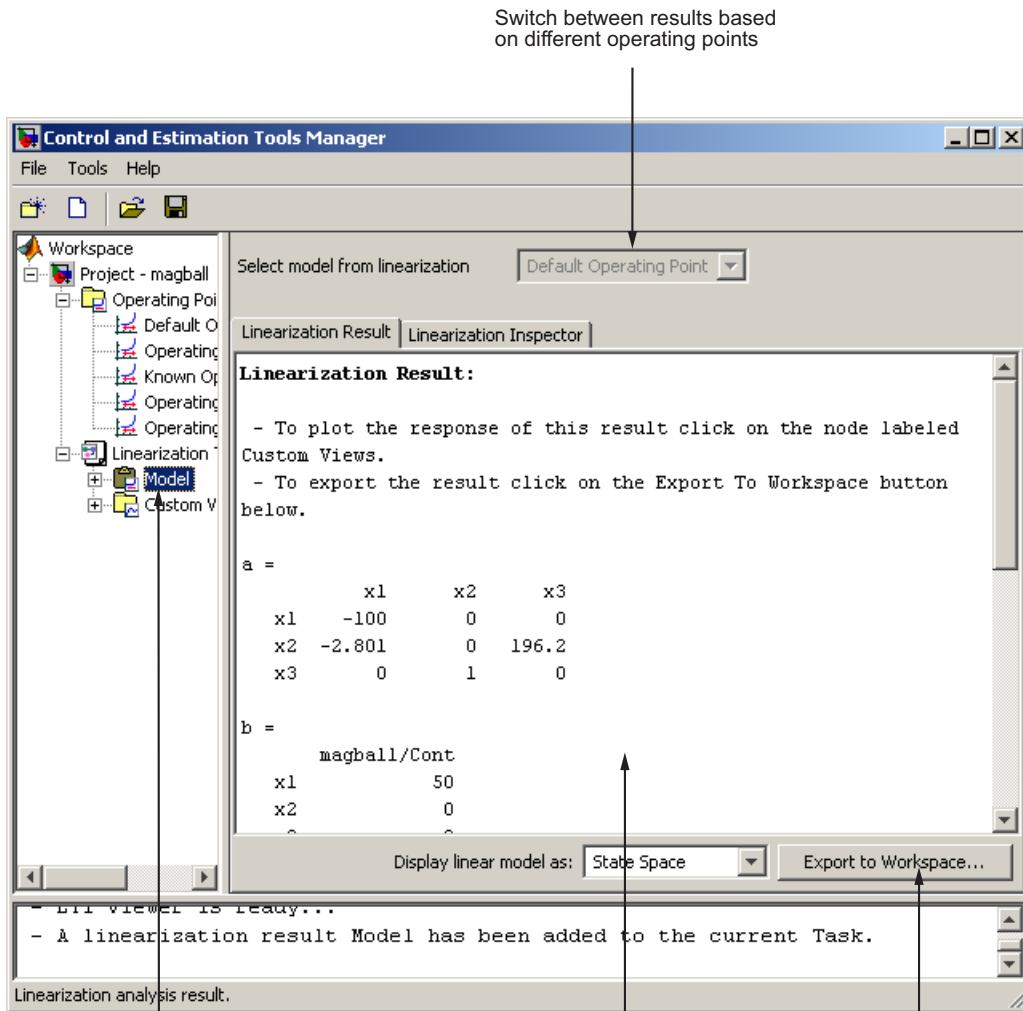
Using the LTI Viewer

To add characteristics such as settling time or peak response to your plot, right-click anywhere in the plot area and select an option from the menu. Add data markers by clicking the point you want to mark. In addition, you can display up to six plots at one time. To change the number of plots, select **Edit > Plot Configurations**. Export the linearized model to the workspace by selecting **File > Export**. For more information, see “LTI Viewer” in the *Control System Toolbox Getting Started Guide*.

Displaying Linearization Results

This section continues the magball example from “Linearizing the Model” on page 4-57. At this stage in the example, a linearization task has been created, operating points have been specified, linearization points have been inserted, and a linearized model has been computed.

To display the **Linearization Result** pane, as shown in the following figure, select the **Model** node in the project tree. To delete the linearization result, right-click **Model** and select **Delete** from the menu. The **Linearization Result** pane displays a mathematical representation of the linearized model. By default it appears as a state space system and it displays the state space matrices. However, within the pane you can choose to view the model as a zero-pole-gain system or as a transfer function. You can also export the model to the MATLAB workspace.



View the operating points used in the analysis by clicking the + to the left of the Model node

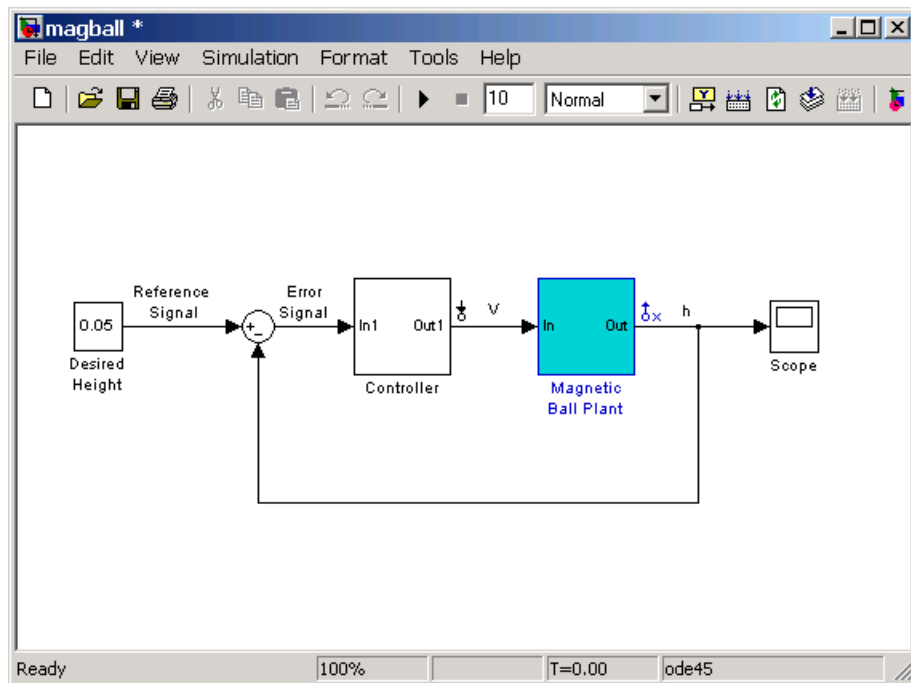
Linearization Result gives the state-space matrices of the linearized model

Export the linearized model to the workspace

Highlighting Blocks in the Linearization

This section continues the magball example from “Displaying Linearization Results” on page 4-66. At this stage in the example, a linearization task has been created, operating points have been created, linearization points have been inserted, and a linearized model has been computed.

To verify that the correct portion of the model was linearized, you can highlight the blocks in the model that were used in the linearization. To do this, right-click the **Model** node and select **Highlight Blocks in Linearization**. The following figure shows that the Magnetic Ball Plant block, including all blocks within it, was used in the linearization.



To remove the highlighting, right-click the **Model** node and select **Remove Highlighting**.

Note You cannot use the **Highlight Blocks in Linearization** or **Remove Highlighting** option for models you linearize with numerical-perturbation linearization (when Numerical perturbation is selected as the **Linearization Algorithm** parameter in the Linearization Task Options window).

Inspecting the Linearization Results Block by Block

This section continues the example from “Displaying Linearization Results” on page 4-66. At this stage in the example, a linearization task has been created, operating points have been created, linearization points have been inserted, and a linearized model has been computed.

To verify that blocks of interest linearized as expected, you can inspect linearization results block by block using the Simulink Control Design Linearization Inspector. An example of a block of interest is a block with a sharp discontinuity.

Note For information on troubleshooting blocks that did not linearize as expected, see “Troubleshooting Exact Linearization Results” on page 4-82.

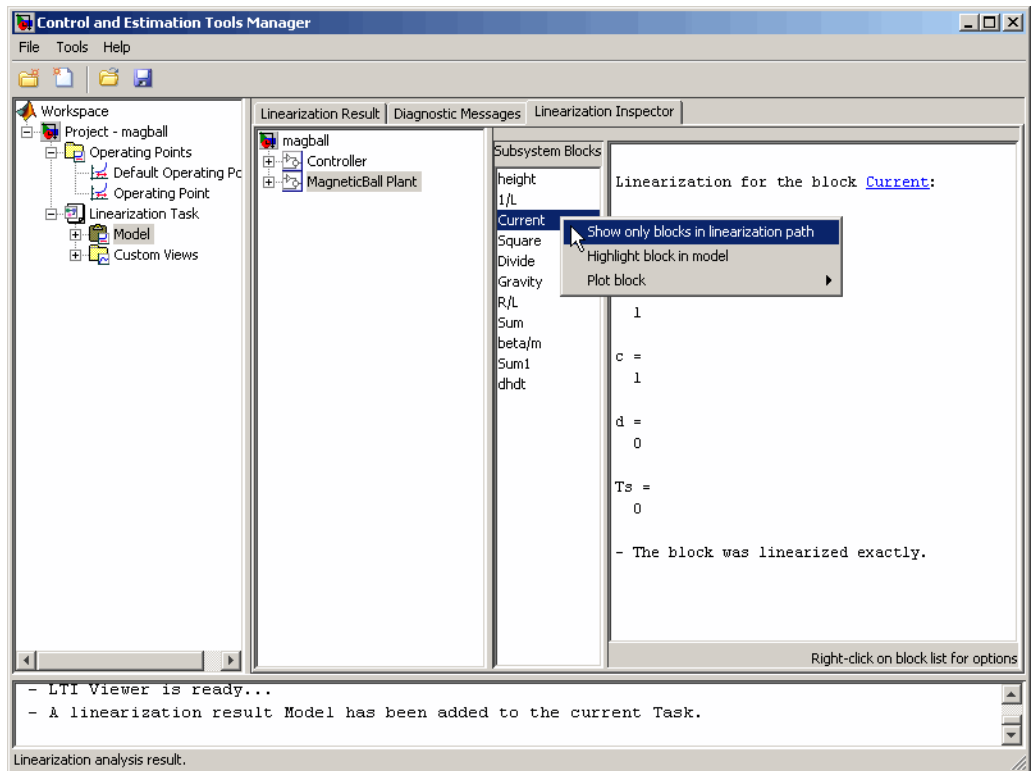
To inspect the linearization results for the *Current* block in the Magnetic Ball Plant:

- 1 Select the **Model** node. Then, select the **Linearization Inspector** tab.
- 2 In the list of blocks under **magball**, select the **MagneticBall Plant** subsystem.
- 3 In the **Subsystems Blocks** column, select the **Current** block.

Tip You can filter the Subsystem Blocks list to include only blocks in the linearization path (the blocks between the input and output points). To filter the list, right-click the list, and select **Show only blocks in linearization path**.

This action displays the following in the Linearization Inspector pane:

- State-space matrices A , B , C , and D for the linearized *Current* block.
- Sample time T_s , which equals zero and shows that the Magnetic Ball Plant is continuous.
- A note saying that the block was linearized exactly.



You can plot the linearization results for a block. To create a step plot of the *Current* block linearization results, right-click the **Current** block in the **Subsystems Blocks** column and select **Plot block > Step Plot**.

Note The **Linearization Inspector** pane is not available for models that are linearized with numerical-perturbation linearization (when **Numerical perturbation** is selected as the **Linearization Algorithm** parameter in the Linearization Task Options window). Such linearizations have no individual block linearizations to inspect.

Disabling the Linearization Inspector

When you do not want diagnostic information for your linearization, you disable the linearization inspector.

To disable the linearization inspector:

- 1 Select **Tools > Options** in the Control and Estimation Tools Manager window. Then, click the **Linearization** tab.
- 2 Deselect the **Store Linearization Diagnostics and Inspector data with linearization** option.

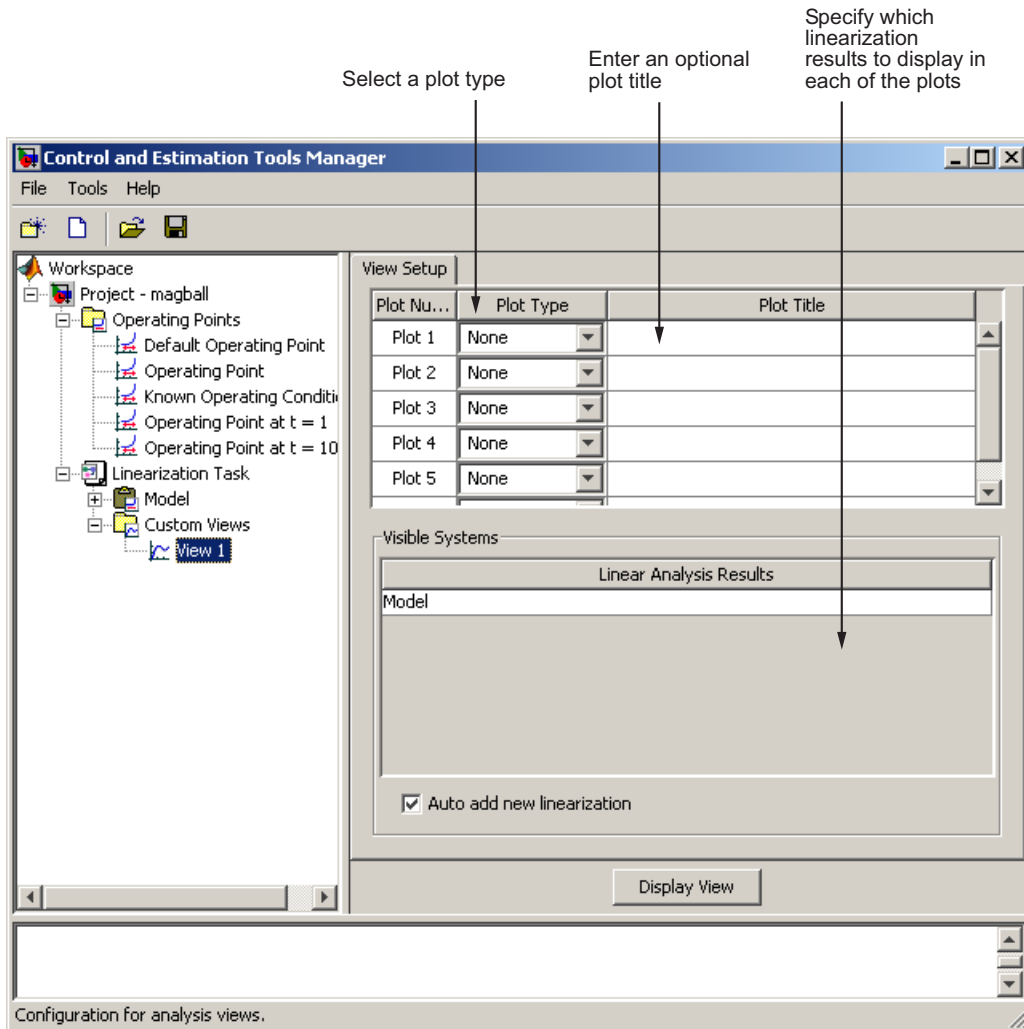
Note You can also disable the linearization inspector globally in the Simulink Control Design tab of the MATLAB preferences dialog box. When you set this global preference, it takes effect in each subsequent creation of linearization tasks and persists from session to session.

For more information on changing linearization options, see “Choosing Linearization Settings and Algorithms” on page 4-9.

Comparing the Results of Multiple Linearizations

This section continues the magball example from “Inspecting the Linearization Results Block by Block” on page 4-69. At this stage in the example, a linearization task has been created, operating points have been created, linearization points have been inserted, and a linearized model has been computed and inspected.

The LTI Viewer automatically displays a **Linearization Quick Plot** of the linearization results after the linearization. To create additional, customized plots, especially for comparing multiple linearizations, right-click the **Custom Views** node under the **Linearization Task** node and select **Add View**. A new view, **View1**, is added to the **Custom Views** node. Select the **View1** node to display the **View Setup** pane and set up this view.



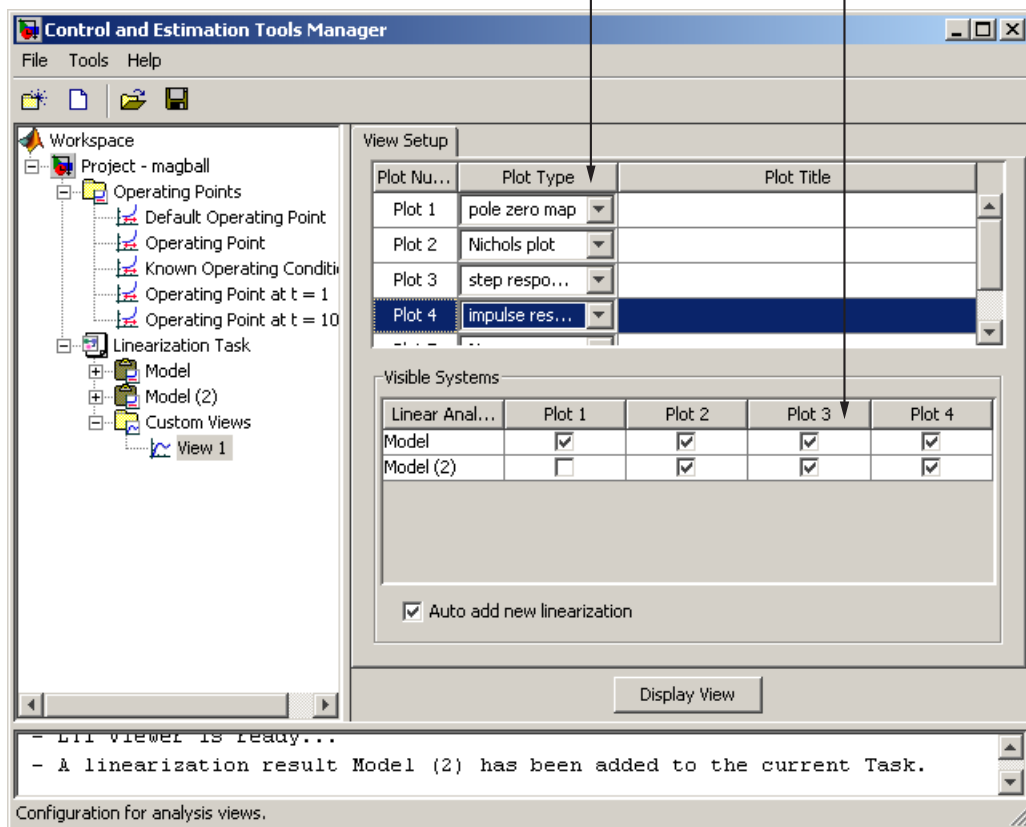
Example

Before configuring the **View Setup** pane for the magnetic ball system, run another linearization of this model using the operating point labeled **Known Operating Conditions**, that you computed in “Specifying Operating Points from Known Values” on page 2-15. This adds **Model (2)** to the **Linearization Task** node.

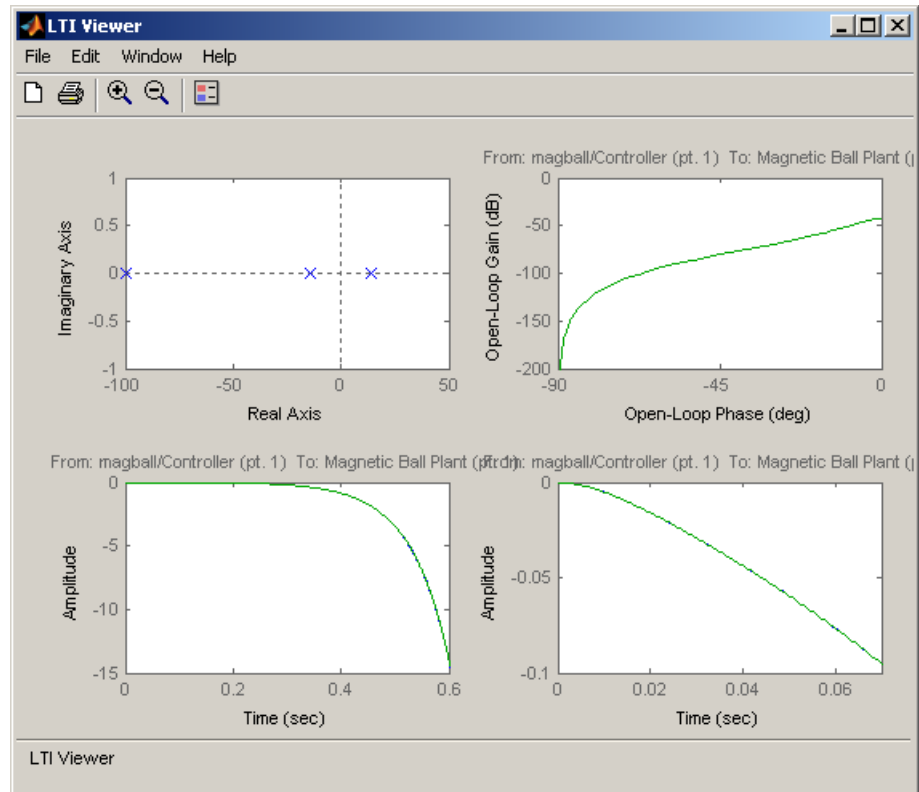
Configure **View Setup** as shown in the following figure:

Select plot types
pole zero map,
Nichols plot, step
response, and
impulse response

Display Model in
plots 1, 3, and 4.
Display Model (2)
in plots 2, 3, and 4.



Click **Display View** to display the LTI Viewer, as shown in the following figure:



Before proceeding to Chapter 7, “Designing Compensators”, close the Control and Estimation Tools Manager and the magball model. You do not need to save any projects or any changes to the model.

Validating Exact Linearization Results

In this section...
“Ways to Validate Exact Linearization Results” on page 4-76
“Creating Input Signals for Validation” on page 4-76
“Frequency-Domain Validation” on page 4-77
“Time-Domain Validation” on page 4-80

Ways to Validate Exact Linearization Results

You can validate the exact linearization results for your Simulink model in either the time domain or frequency domain:

- Frequency-domain validation — Compare the linear model to a frequency response estimation of the nonlinear model. When it computes the estimation, the software simulates your model while injecting the input signal at the linearization input point. For more information, see “Frequency-Domain Validation” on page 4-77.
- Time-domain validation — Compare the simulation output of the linear model and the nonlinear model using the same input signal. For more information, see “Time-Domain Validation” on page 4-80.

Validation requires an input signal that you create in “Creating Input Signals for Validation” on page 4-76.

Creating Input Signals for Validation

To validate linearization results, first create an input signal for simulation. For both time- and frequency- domain validation, the software simulates your model while injecting the input signal at the linearization input point. Then, the software measures the output signal at the linearization output point. For frequency-domain validation, the software uses the output signal to compute a frequency response estimation.

This table summarizes the available input signals:

Type of input signal	Validate models with this signal...	For example syntax, see...
Set of sinusoids	At a particular set of frequencies	<code>frest.Sinestream</code>
Set of sinusoids with fixed sample time	At a particular set of frequencies when your Simulink model has linearization I/Os on a signal with discrete sample times.	<code>frest.createFixedTsSinestream</code>
Chirp	Over a frequency range	<code>frest.Chirp</code>
Random	When the model contains noise models	<code>frest.Random</code>
Step	When the model sees step inputs	<code>frest.createStep</code>

For Sinestream, Chirp, and Random input signals, you can specify different options, such as amplitude either:

- Automatically based on the existing linear system you are validating
- Manually by typing in comma-separated name/value pairs

For more information, see the input signal reference pages.

Tip After you create an input signal, you can view a plot of your input signal by typing `plot(input)`.

Frequency-Domain Validation

Before you validate linearization results in the frequency domain, complete the following tasks:

- Linearize your Simulink model using exact linearization, as described in Chapter 4, “Exact Linearization Using the GUI” or Chapter 5, “Exact Linearization Using the Command Line”.

If you linearize the model using the GUI, export the linear model and operating point to the MATLAB workspace, as described in “Exporting Linearization Results” on page 1-13.

- Create an input signal for simulation, as described in “Creating Input Signals for Validation” on page 4-76.

To validate linearization results in the frequency domain:

- 1 Open the Simulink model that you linearized using exact linearization.
- 2 Estimate the frequency response of the model using the `frestimate` command:

```
[sysest,simout] = frestimate('model',op,io,input);
```

The software simulates your model while injecting the input signal at the linearization input point. It then measures the output signal at the linearization output point. Then, the software computes the estimation as

$$\text{Frequency Response Model} = \frac{\text{fft of } y_{est}(t)}{\text{fft of } u_{est}(t)}$$

Note For Sinestream input signals with the `ApplyFilteringInFRESTIMATE` option set to on, the software filters the steady-state portion of the input and output signal before computing the estimation. This setting on by default.

The software returns both:

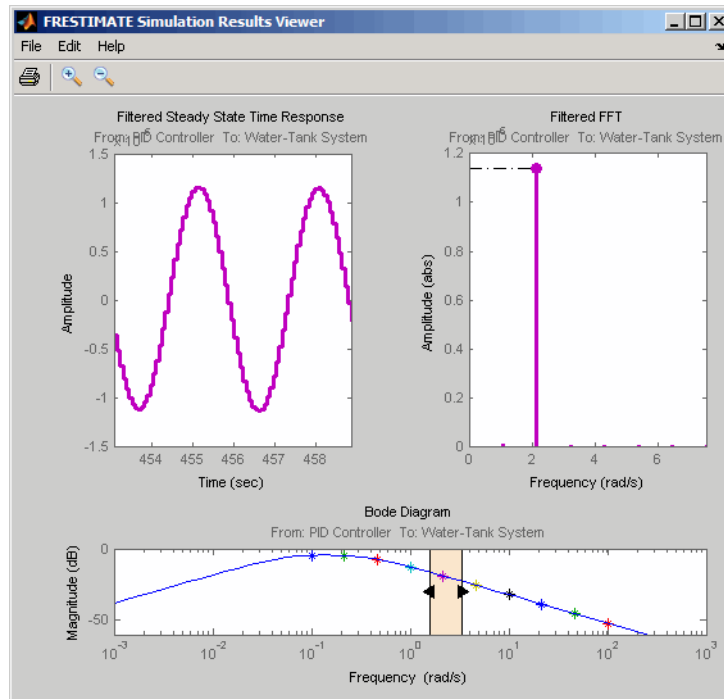
- Frequency response data (FRD) model `sysest`
 - Simulation output `simout`
- 3 Compare the frequency responses of your linear model, `sys`, and the new estimated model. Use either the `bode` command or the `frest.simView` command:

```
frest.simView(simout,input,sysest,sys)
```

This command opens the Simulation Results Viewer, which includes a Bode plot of both:

- Linear model, which displays as a solid line
- New estimated model, which displays as asterisks

The Viewer also includes a time response and FFT of the input signal.



In the Bode plot, if the linear model matches the frequency response estimation, your linearization results are accurate. For more information on using the Simulation Results Viewer, see “Analyzing the Simulated Output and FFT at Specific Frequencies” on page 6-23.

- 4 If the linear model and frequency response estimation do not match, troubleshoot as follows:
 - a Troubleshoot the estimation as described in “Troubleshooting a Frequency Response Estimation” on page 6-26.

- b** If you determine the estimation is accurate, troubleshoot the exact linearization as described in “Troubleshooting Exact Linearization Results” on page 4-82.

Time-Domain Validation

Before you can validate linearization results in the time domain, you must have already performed the following tasks:

- Linearize your Simulink model using exact linearization, as described in Chapter 4, “Exact Linearization Using the GUI” or Chapter 5, “Exact Linearization Using the Command Line”.

Note If you linearized the model using the GUI, export the linear model and operating point to the MATLAB workspace as described in “Exporting Linearization Results” on page 1-13.

- Create an input signal for simulation, as described in “Creating Input Signals for Validation” on page 4-76

To validate linearization results in the time domain:

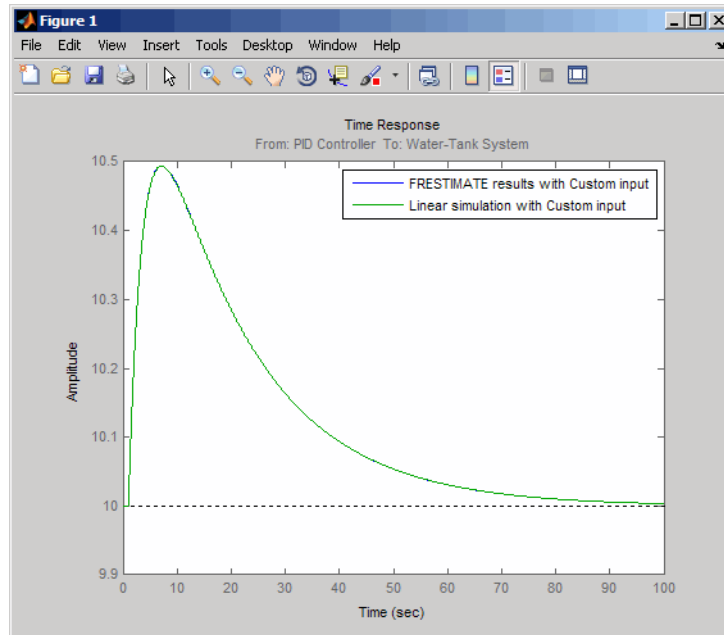
- 1** Open Simulink model that you linearized using exact linearization.
- 2** Simulate the Simulink model with the inputs signal, and capture the outputs using the `frestimate` command:

```
[~,simout] = frestimate('model',op,io,input);
```

The software returns the simulation output `simout`.

- 3** Simulate you linear model `sys`, and compare the time-domain responses of the Simulink model `simout` and the linear model using the `frest.simcompare` command:

```
frest.simCompare(simout,sys,input)
```



If the simulations of the linear model and of the Simulink model match, your linearization results are accurate.

- 4** If the simulations of the linear model and of the Simulink model do *not* match, troubleshoot the exact linearization as described in “Troubleshooting Exact Linearization Results” on page 4-82.

Troubleshooting Exact Linearization Results

In this section...
“Diagnosing Blocks” on page 4-82
“Troubleshooting Your Model at the Subsystem and Block Level” on page 4-86
“Troubleshooting Linearization Settings” on page 4-87
“Troubleshooting Models with Events-Based Subsystems” on page 4-88
“Troubleshooting Your Operating Point” on page 4-88

Diagnosing Blocks

- “Blocks in Your Model That Impact Linearization Results” on page 4-82
- “Which Blocks Linearize Correctly?” on page 4-83
- “Blocks with Configuration Warnings” on page 4-83
- “Unsupported Blocks for Linearization” on page 4-84
- “Blocks That Automatically Linearize Using Numerical Perturbation” on page 4-84
- “Using Diagnostics Messages to Find Problematic Blocks in Your Model” on page 4-85

Blocks in Your Model That Impact Linearization Results

During block-by-block analytic linearization, the linearization of each block in the linearization path of your model impacts the overall linearization results.

You can locate the blocks that impact your linearization results by highlighting the blocks in the linearization path. For instructions about highlighting blocks in the linearization path, see “Highlighting Blocks in the Linearization” on page 4-68.

Note If one of the blocks in the linearization path of your model does not highlight, it might have a linearization result of zero. You can use the Simulink Control Design diagnostic messages to determine if other blocks in your model are causing this block to linearize to zero. For information on how to view diagnostic messages, see “Using Diagnostics Messages to Find Problematic Blocks in Your Model” on page 4-85.

Which Blocks Linearize Correctly?

Nearly all core Simulink blocks give accurate linearization results. The exceptions are the blocks that are not supported for linearization. See “Unsupported Blocks for Linearization” on page 4-84 for information on how to locate these blocks.

In some cases, the accurate linearization results you obtain might not be the result you expect. If you encounter unexpected linearization results, you can use the Simulink Control Design diagnostic messages to identify which blocks are causing problems in your linearization. For information on how to view diagnostic messages, see “Using Diagnostics Messages to Find Problematic Blocks in Your Model” on page 4-85.

Note Blocks with discontinuities, such as relay and Boolean logic, linearize to zero or infinity. These blocks do not display in the diagnostic messages. If your model contains these blocks, verify that they linearize in the way that you expect. For information on how to find these blocks in your model, see “Troubleshooting Your Model at the Subsystem and Block Level” on page 4-86 and “The Model Explorer”. For more information on blocks with discontinuities, see “Blocks with Discontinuities” on page 4-17.

Blocks with Configuration Warnings

Some linearization-compatible blocks encounter warning messages during linearization. You can use these messages as a guide for modifying your model to obtain the linearization results you expect.

To locate blocks in your model with configuration warnings and to read the warning messages, view the list of block with warning in the Simulink Control Design **Diagnostics Messages** tab. For information on how to view diagnostic messages, see “Using Diagnostics Messages to Find Problematic Blocks in Your Model” on page 4-85.

Unsupported Blocks for Linearization

Some Simulink blocks are not supported for linearization and give the wrong answer when linearized. If your model contains blocks that are not supported for linearization, you must replace them to obtain accurate linearization results.

You can find a list of unsupported blocks for linearization in the Simulink Control Design **Diagnostics Messages** tab. For information about viewing diagnostic messages, see “Using Diagnostics Messages to Find Problematic Blocks in Your Model” on page 4-85.

Blocks That Automatically Linearize Using Numerical Perturbation

Blocks that do not have pre-programmed exact analytic Jacobians automatically linearize using the numerical perturbation algorithm instead of block-by-block analytic linearization. Block behavior and numerical perturbation levels affect the accuracy of linearization results. For most blocks, you do not need to check these settings. However, to obtain the results you expect, consider adjusting the numerical perturbation levels in the following situations:

- Blocks that are located near discontinuous regions

Some example of discontinuous regions are $1/u$, where u is near zero, and regions between values in lookup tables.

- Blocks that have nondouble inputs and states

These blocks linearize to zero.

To locate blocks in your model that automatically linearize using numerical perturbation, click the hyperlink in the **blocks without pre-programmed exact Jacobian (linearized using numerical perturbation)** section of the Simulink Control Design **Diagnostics Messages** tab. For information

about viewing diagnostic messages, see “Using Diagnostics Messages to Find Problematic Blocks in Your Model” on page 4-85.

Note Blocks that have nondouble inputs and states also appear in the list of blocks with warnings in the Simulink Control Design **Diagnostics Messages** tab.

For more information about numerical perturbation linearization and setting numerical perturbation levels, see “Numerical-Perturbation Linearization” on page 4-24.

Using Diagnostics Messages to Find Problematic Blocks in Your Model

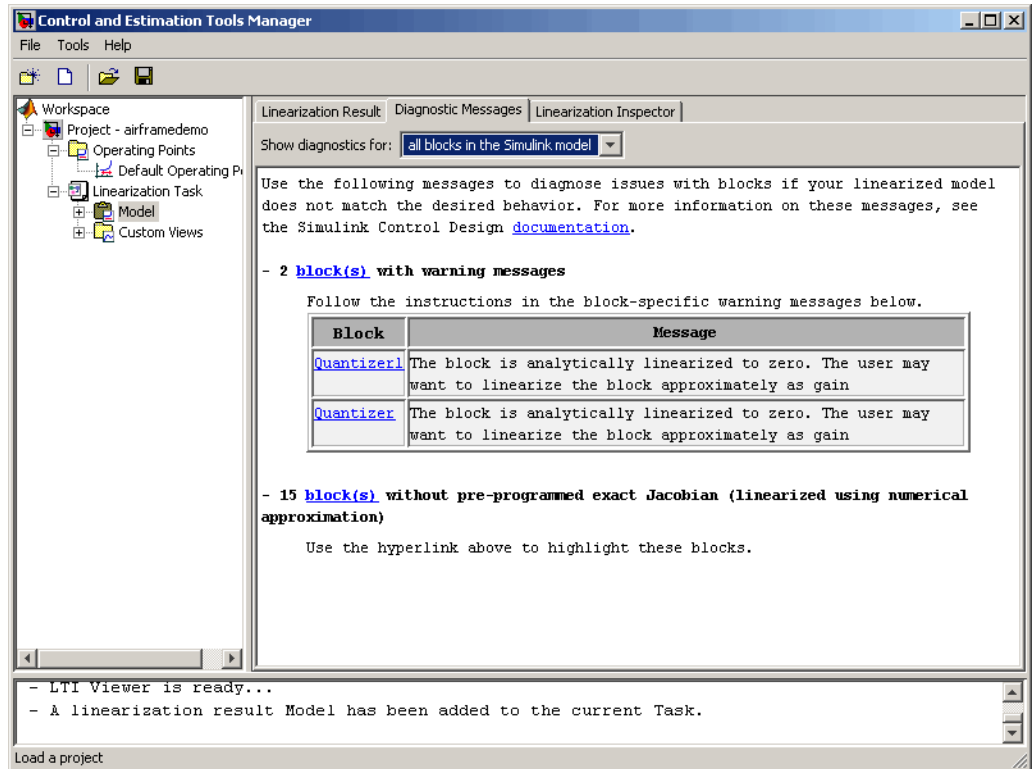
Diagnostic messages identify the following types of blocks in your model:

- “Unsupported Blocks for Linearization” on page 4-84
- “Blocks with Configuration Warnings” on page 4-83
- “Blocks That Automatically Linearize Using Numerical Perturbation” on page 4-84

To view the diagnostic messages for the blocks in your linearized model, perform these steps:

- 1** Select the **Model** node in the project tree for your linearized model.
- 2** Select the **Linearization Diagnostic Messages** tab.
- 3** In the **Show diagnostics for** drop-down list, choose one of the following options:
 - **Blocks in the linearization path**
 - **All blocks in the Simulink model**

Your result resembles the following figure.



Note To highlight the blocks listed in the **Diagnostic Messages** tab in your model, click the hyperlinks.

Troubleshooting Your Model at the Subsystem and Block Level

If you obtain an unexpected linearization result, you can examine the subsystems and blocks in your model to determine which part of your model is not linearizing properly. If you find subsystems and blocks in your model that are not linearizing properly, replace them and then linearize your model again.

Some of the issues that affect your linearization results at the subsystem and block level include:

- Linearizations that result in zero or infinity.
- Poles that do not make sense. For example, poles whose values are not characteristic of the system.

To examine subsystems and blocks in your model, you can use the following techniques:

- Inspect linearization results for each block using the linearization inspector.

For instructions on using the linearization inspector, see “Inspecting the Linearization Results Block by Block” on page 4-69.

- Linearize individual subsystems and blocks in your model.

This approach is an effective way to debug the linearization of a large model because it allows you to quickly narrow down the problem. For instructions on linearizing subsystems and blocks, see “Linearizing a Block” on page 4-55.

Note In Simulink Control Design block and subsystem linearization, the blocks and subsystems in your model are linearized about the operating point of the entire model.

Troubleshooting Linearization Settings

The following linearization settings affect your linearization results. Verify that these linearization settings are appropriate for your linearization.

- Rate Conversion Method

When you linearize models with multiple sample times, such as a discrete controller with a continuous plant, a rate conversion algorithm generates a single-rate linear model. The algorithm you select affects linearization results. For information about on these effects, see the Simulink Control Design demos “Linearization of Multirate Models” and “Rate Conversion Method Selection for Linearization” listed under the Simulink Control Design Demos in the demos browser.

- Return model with exact delay

When you linearize models with delays, you can use exact delay representation. For information about how exact delay representations affect linearization results, see the Simulink Control Design demo “Linearizing Models with Delays” listed under the Simulink Control Design Demos in the demos browser.

For more information about these settings, see “How to Choose Linearization Settings and Algorithms” on page 4-9 in the Simulink Control Design documentation and the `linoptions` reference page.

Troubleshooting Models with Events-Based Subsystems

Event-based subsystems do not trigger during linearization and therefore, you cannot linearize these subsystems. You must replace each of the event-based subsystem in the linearization path of your model with a representation that you can linearize.

For more information about linearizing event-based subsystems, see “Event-Based Models and Triggered Subsystems” on page 4-20.

Troubleshooting Your Operating Point

If you obtain unexpected linearization results, check if the operating point you used for linearization is causing inaccurate results.

If you find that the operating point you selected was not ideal for the linearization, choose another operating point and linearize your model again.

For more information, see the following information in the Simulink Control Design documentation:

- How the operating point you choose affects your linearization results — See “Why Are Operating Points Important?” on page 2-6.
- How to creating accurate operating points — See “Recommendations for Computing Operating Points” on page 2-31.

Exact Linearization Using the Command Line

- “Steps for Linearizing Models Using the Command Line” on page 5-2
- “Configuring the Linearization for Specific Blocks and Subsystems” on page 5-3
- “Selecting Inputs and Outputs for the Linearized Model” on page 5-4
- “Linearizing the Model Using Functions” on page 5-11
- “Analyzing the Results Using Functions” on page 5-14

Steps for Linearizing Models Using the Command Line

As discussed in “Purpose of Linearization” in the Simulink Control Design getting started documentation, linearization is an important process in the design and analysis of control systems. The main steps involved in the linearization of Simulink models using the Simulink Control Design functions are

- 1** Creating or opening a Simulink model. See “Creating or Opening a Simulink Model” on page 1-2.
- 2** Configuring the linearization for specific blocks and subsystems. See “Configuring the Linearization for Specific Blocks and Subsystems” on page 5-3.
- 3** Selecting inputs and outputs for the linearized model. See “Selecting Inputs and Outputs for the Linearized Model” on page 5-4.
- 4** Specifying operating points. See Chapter 3, “Operating Point Analysis Using the Command Line”.
- 5** Linearizing the model. See “Linearizing the Model Using Functions” on page 5-11.
- 6** Analyzing the results and saving your work. See “Analyzing the Results Using Functions” on page 5-14.

Although this chapter focuses on the Simulink Control Design functions for linearizing models, you can also use the Graphical User Interface (GUI) for some steps in the process. For example, after specifying the operating points in the GUI, you can export the results to the MATLAB workspace and use the functions to continue the analysis. For discussion of the advantages and disadvantages of the GUI versus the functions, refer to “Using the GUI Versus Command-Line Functions” in the Simulink Control Design getting started documentation. A particular advantage of the linearization functions is the ability to write scripts to automate the linearization process or perform *batch linearization*.

Configuring the Linearization for Specific Blocks and Subsystems

When you linearize using the command line, you can configure the linearization of block, subsystem, or model reference block in the following ways:

- Configure block-specific settings to control how the block linearizes

See “Controlling the Analytic Linearization of Individual Blocks” on page 4-36 and “Controlling the Block Perturbation Linearization of Individual Blocks” on page 4-40.

- Specify the actual linearization result for any block, subsystem, or model reference block in either:

- The Simulink model before you linearize

For more information, see “Specifying the Linearization of Blocks and Subsystems” on page 4-37.

- A structure before you linearize

In this case, use the syntax `lin = linearize('sys',blocksubs)`, where `blocksubs` is the structure specifying the linearization. For more information, see the `linearize` reference page.

For an example of specifying the linearization of a block using a MATLAB expression, see the Specifying Custom Linearizations for Simulink Blocks demo.

Selecting Inputs and Outputs for the Linearized Model

In this section...
“Workflow for Selecting Inputs and Outputs” on page 5-4
“Choosing and Storing Linearization Points” on page 5-4
“Extracting Linearization Points from a Model” on page 5-7
“Editing an I/O Object” on page 5-8
“Open-Loop Analysis Using Functions” on page 5-10

Workflow for Selecting Inputs and Outputs

This section describes how to configure the model for linearization using functions. For a description of how to use the graphical interface for this task, see “Selecting Inputs and Outputs for the Linearized Model” on page 4-45.

Before linearizing the Simulink model of your system, configure it by

- 1 Choosing linearization input and output points.
- 2 Storing linearization points in an input/output (I/O) object.
- 3 Editing the I/O object, when necessary, such as when computing the open-loop model.

The input and output points define the portion of your model being linearized. Setting the `OpenLoop` property of a linearization point to 'on' allows you to compute an open-loop model. Refer to “What are Linearization Points?” on page 4-45 for more information on linearization input and output points.

Choosing and Storing Linearization Points

This section continues the example from “Example: Water-Tank System” on page 3-3.

In the `watertank` model, the nonlinearities are in the water-tank system itself. To linearize this portion of the model, place an input point before it and

an output point after it. Information about the linearization points is stored in an input/output (I/O) object in the MATLAB workspace.

Each linearization point is associated with an output of a block. To place an input point before the Water-Tank System block, you need to associate this input point with the output of the PID Controller block.

To create an I/O object for the input point, use the `linio` function.

```
watertank_io(1)=linio('watertank/PID Controller',1,'in')
```

This creates an object, `watertank_io`, in the MATLAB workspace and displays the object as shown below.

```
Linearization IOs:
```

```
-----
```

```
Block watertank/PID Controller, Port 1 is marked with the following properties:
```

- No Loop Opening
- An Input Perturbation
- No signal name. Linearization will use the block name

The first input argument of the `linio` function is the name of the block that the linearization point is associated with. The second argument is the number of the output on this block that the linearization point is associated with. These two arguments allow the linearization point to be placed on a specific signal line. The third argument is the type of linearization point. Available types are

'in'	input point
'out'	output point
'inout'	input point followed by output point
'outin'	output point followed by input point

To create a second object within `watertank_io` for an output point, use the following command.

```
watertank_io(2)=linio('watertank/Water-Tank System',1,'out')
```

This creates an I/O object for the output point that is located at the first output of the block watertank/Water-Tank System. The object watertank_io is displayed, as shown below.

```
Linearization IOs:
```

```
-----
```

```
Block watertank/PID Controller, Port 1 is marked with the following properties:
```

- No Loop Opening
- An Input Perturbation
- No signal name. Linearization will use the block name

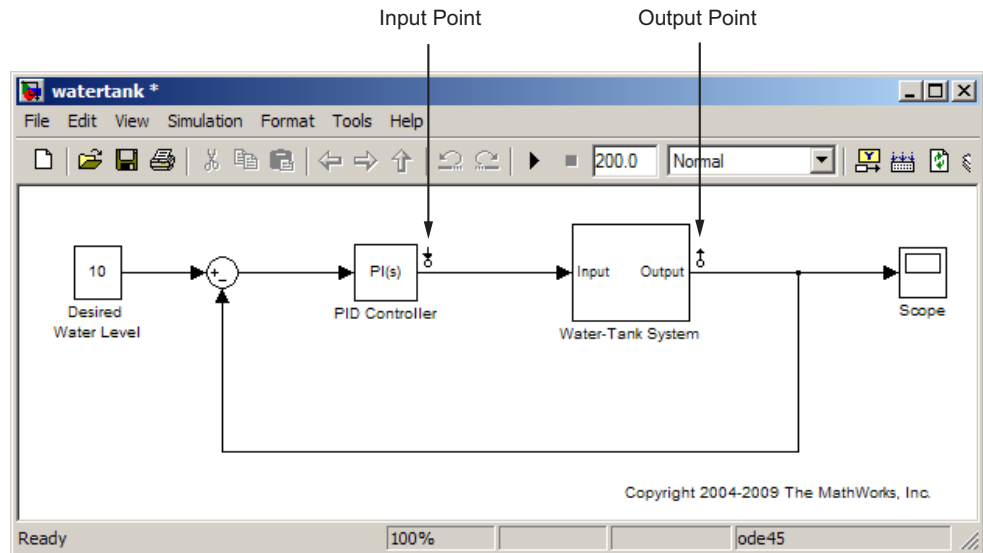
```
Block watertank/Water-Tank System, Port 1 is marked with the following properties:
```

- An Output Measurement
- No Loop Opening
- No signal name. Linearization will use the block name

Both the input and output points are now stored in the MATLAB workspace in the I/O object watertank_io. To view the linearization points on the model diagram, upload the settings in watertank_io using the setlinio function.

```
setlinio('watertank',watertank_io)
```

The model diagram should now look like that in the following figure.



Water-Tank Model with Input and Output Points Selected

Extracting Linearization Points from a Model

This section continues the example from “Example: Water-Tank System” on page 3-3. At this stage in the example, linearization points have been inserted in the model. See “Choosing and Storing Linearization Points” on page 5-4 for more information on inserting linearization points in the model using functions.

An alternative way to create an I/O object is to extract the linearization points from the model diagram when they have been selected using the right-click menus described in “Inserting Linearization Points” on page 4-46. The extracted linearization points are stored in an I/O object. Use the `getlinio` function to extract the linearization points in the following way.

```
watertank_io=getlinio('watertank')
```

This returns

```
Linearization IOs:
-----
Block watertank/Controller, Port 1 is marked with the following
properties:
- No Loop Opening
- An Input Perturbation
- No signal name. Linearization will use the block name

Block watertank/Water-Tank System, Port 1 is marked with the
following properties:
- An Output Measurement
- No Loop Opening
- No signal name. Linearization will use the block name
```

Editing an I/O Object

This section continues the example from “Example: Water-Tank System” on page 3-3. At this stage in the example, linearization points have been inserted in the model and extracted to an object in the MATLAB workspace. See “Extracting Linearization Points from a Model” on page 5-7 for more information on extracting linearization points from a model using functions.

Typing the name of the I/O object at the command line returns a formatted display of key object properties. To view a list of all properties, use the `get` function. Each object within the I/O object has six properties. For example, to view the properties of the second object in `watertank_io`, type

```
get(watertank_io(2))
```

MATLAB displays

```
Active: 'on'
Block: 'watertank/Water-Tank System'
OpenLoop: 'off'
PortNumber: 1
Type: 'out'
Description: ''
```

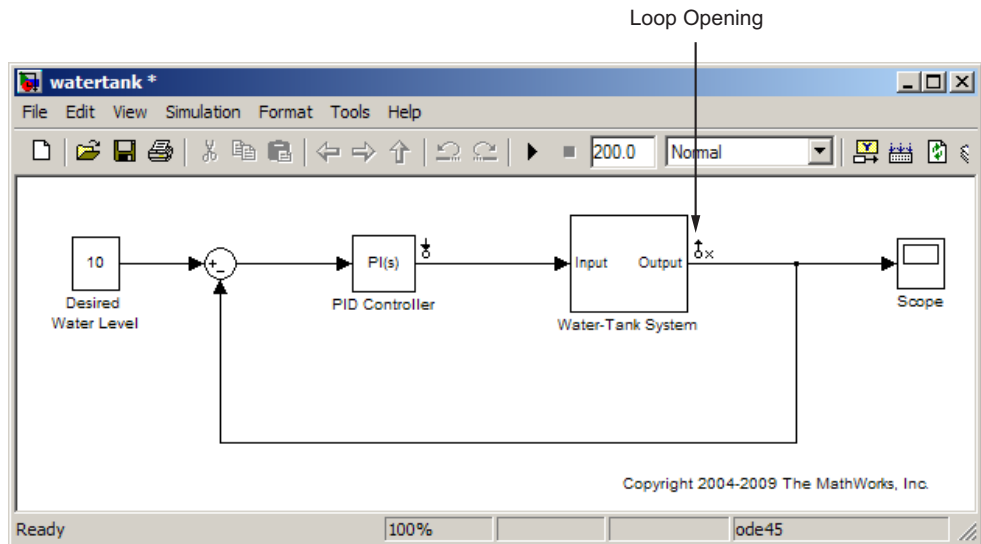
You can edit this object to make any necessary changes. For example, to make this linearization point a loop opening as well, type

```
watertank_io(2).OpenLoop='on'
```

To refresh the model diagram so that it reflects any changes made to the I/O object using the functions, use the `setlinio` function.

```
setlinio('watertank', watertank_io);
```

A small x appears next to the output point in the diagram, indicating a new loop opening, as shown in this figure.



Water-Tank Model with Loop Opening

You can edit the other properties of I/O objects in a similar way. For more information about each property and the possible values it can take, see the `getlinio` reference page.

Open-Loop Analysis Using Functions

When you want to remove the effect of signals feeding back into the portion of the model you are linearizing, it is often convenient to insert open-loop points in the model. For methods on inserting loop openings with the Simulink Control Design GUI, refer to “Performing Open-Loop Analysis” on page 4-49. An alternative method of inserting loop openings, using functions, is to edit the I/O object as described in “Editing an I/O Object” on page 5-8.

Linearizing the Model Using Functions

In this section...

“Linearizing the Model” on page 5-11

“Linearizing Discrete-Time and Multirate Models” on page 5-13

“Computing Multiple Linearizations for Large Models” on page 5-13

Linearizing the Model

This section describes how to linearize the model using functions. For a description of how to use the graphical interface for this task, see “Linearizing the Model” on page 4-57.

This section also continues the example from “Example: Water-Tank System” on page 3-3. At this stage in the example, linearization point objects and operating point have been created in the MATLAB workspace. See Chapter 3, “Operating Point Analysis Using the Command Line” for more information on creating operating point objects using functions.

After creating an I/O object and determining the operating point, you are ready to linearize the system, using the `linearize` command. For example:

```
watertank_lin=linearize('watertank',watertank_op,watertank_io)
```

MATLAB returns the matrices `a`, `b`, `c`, and `d` of a linear, time-invariant, state-space model that approximates your nonlinear system in a region around the operating point.

```
a =
      H
      H  -0.01581

b =
      PID Control
      H          0.25

c =
      H
```

```
Water-Tank S 1

d =
          PID Controll
Water-Tank S      0

Continuous-time model.
```

To change the linearization options, use the `linoptions` function before running the linearization. For example, to change the sample time for the linearization model to be 1 instead of continuous, use the following command:

```
linopt=linoptions('SampleTime',1);
```

Then, run the linearization with these options.

```
watertank_lin2=linearize('watertank',watertank_op,watertank_io,linopt)
```

This returns the discrete-time model shown below.

```
a =
          H
H 0.9845

b =
          PID Controll
H      0.2481

c =
          H
Water-Tank S 1

d =
          PID Controll
Water-Tank S      0

Sampling time: 1
Discrete-time model.
```


Linearizing Discrete-Time and Multirate Models

The linearization method is the same for models containing discrete-time states or several different sampling rates. However, you can choose to adjust the `SampleTime` parameter with the `linoptions` function as shown in the previous section. By default this parameter is set to `-1`, in which case the Simulink Control Design software will find the slowest sample rate in the model to use for the sample rate of the linearized model. To create a linearized model with different sample time, specify a new parameter value before linearizing the model. A value of `0` will give a continuous-time model. For more information, see the [Linearization of Multirate Models](#) demo.

Computing Multiple Linearizations for Large Models

To compute multiple linearizations for large models when only a few blocks or model references change per linearization:

- 1 Linearize the fixed portion of the model once using `linlft`.
- 2 Linearize the varying portion the desired number of times using `linearize`.
- 3 Combine the results using `linlftfold`.

The combined results are equal to the results you obtain by linearizing the entire model multiple times.

For more information, see the [Speeding Up the Computation of Multiple Linearizations with Block Variations](#) demo.

Analyzing the Results Using Functions

In this section...
“Options for Analyzing the Results” on page 5-14
“Using the LTI Viewer” on page 5-14
“Saving Your Work” on page 5-16
“Restoring Linearization I/O Settings” on page 5-16

Options for Analyzing the Results

This section describes how to analyze the linearization results using functions. For a description of how to use the graphical interface for this task, see “Viewing Linearization Results” on page 4-66.

You can analyze the linearized model by

- Using Control System Toolbox functions at the MATLAB prompt.
- Displaying it in the LTI Viewer.
- Incorporating the results into a block in a Simulink model.

For methods on simulating the linearized model for comparison with the original model, refer to “Validating Exact Linearization Results” on page 4-76

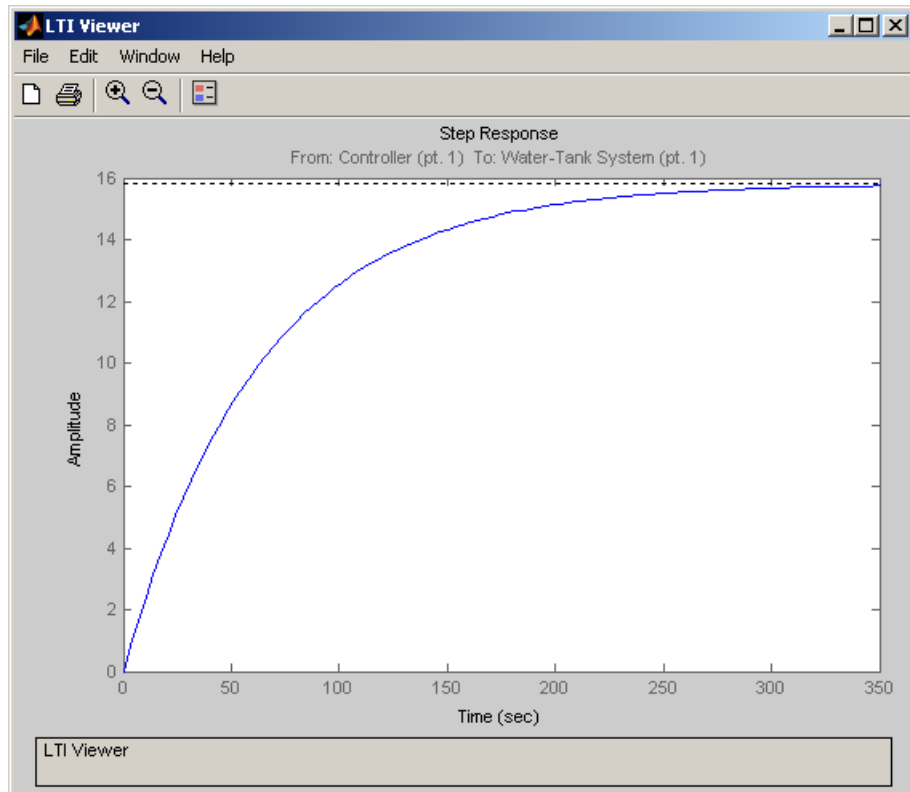
Using the LTI Viewer

This section continues the example from “Example: Water-Tank System” on page 3-3. At this stage in the example, linearization point objects and operating point have been created in the MATLAB workspace, and a linearized model has been computed. See “Linearizing the Model Using Functions” on page 5-11 for more information on computing a linearized model using functions.

To send your linearized model to the LTI Viewer for display, type

```
ltiview(watertank_lin)
```

The LTI Viewer opens, by default, with a step response of the linearized system, as shown in the following figure.



LTI Viewer Displaying a Step Response of the Linearized Model

You can use standard LTI Viewer features to display your results. For example, change the plot type by right-clicking anywhere in the plot area and choosing from the **Plot Types** menu. To add characteristics such as settling time or peak response to your plot, right-click anywhere in the plot area and choose from the **Characteristics** menu. Add data markers by clicking the point you want to mark.

You can display up to six plots in the LTI Viewer window. To change the number of plots, select **Edit > Plot Configurations**, choose a configuration in the Plot Configurations dialog box, and then click **OK**.

For more information on the LTI Viewer, refer to the Control System Toolbox documentation.

Saving Your Work

This section continues the example from “Example: Water-Tank System” on page 3-3. At this stage in the example, linearization point objects and operating point have been created in the MATLAB workspace, and a linearized model has been computed. See “Linearizing the Model Using Functions” on page 5-11 for more information on computing a linearized model using functions.

This section describes how to save a linearization project using functions. For a description of how to use the graphical interface for this task, see “Saving Projects” on page 1-11.

To save your linearized model for later analysis, use the `save` command. For example, to save the linearized model, operating points, and I/O object of the `watertank` model, type

```
save watertank_project watertank_lin watertank_op watertank_io
```

This creates a file named `watertank_project.mat` in the current directory. To reload this file, use the `load` function.

```
load watertank_project
```

Restoring Linearization I/O Settings

To save linearization I/O settings for use in a later session, use the `save` function. You can then restore the settings by loading them with the `load` function and using the `setlinio` function to upload them to the model diagram. For more information, see the function reference page for `setlinio`.

Alternatively, you can use the reloaded I/O settings object with the `linearize` function without uploading it to the model diagram.

Frequency Response Estimation of Simulink Models

- “About Frequency Response Estimation” on page 6-2
- “Creating Input Signals for Estimation” on page 6-7
- “Estimating Frequency Response” on page 6-17
- “Analyzing Estimated Frequency Response” on page 6-20
- “Troubleshooting a Frequency Response Estimation” on page 6-26
- “Estimating Models With Noise” on page 6-39

About Frequency Response Estimation

In this section...

“What Is a Frequency Response Model?” on page 6-2

“Overview of Frequency Response Estimation in Simulink” on page 6-3

What Is a Frequency Response Model?

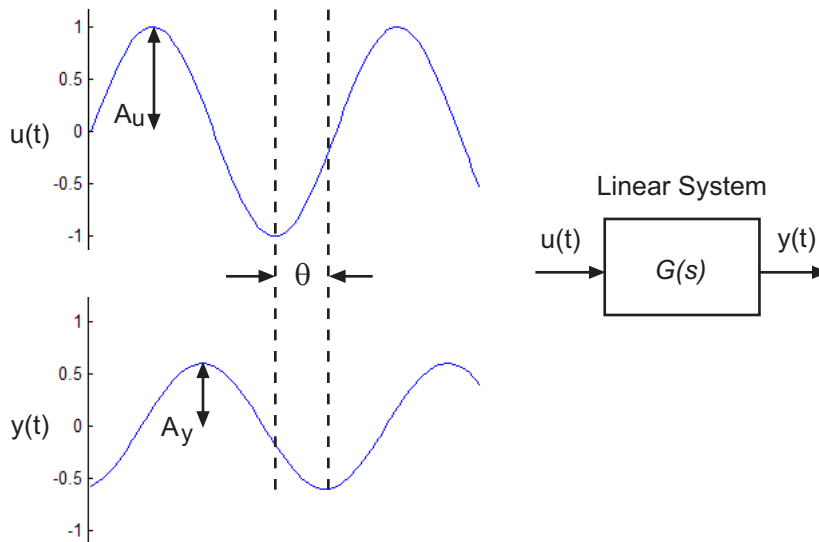
Frequency response describes the steady-state response of a system to sinusoidal inputs.

For a linear system, a sinusoidal input of frequency ω :

$$u(t) = A_u \sin \omega t$$

results in an output that is also a sinusoid with the same frequency, but with a different amplitude and phase θ :

$$y(t) = A_y \sin(\omega t + \theta)$$



Frequency response $G(s)$ for a stable system describes the amplitude change and phase shift as a function of frequency:

$$Y(s) = G(s)U(s)$$

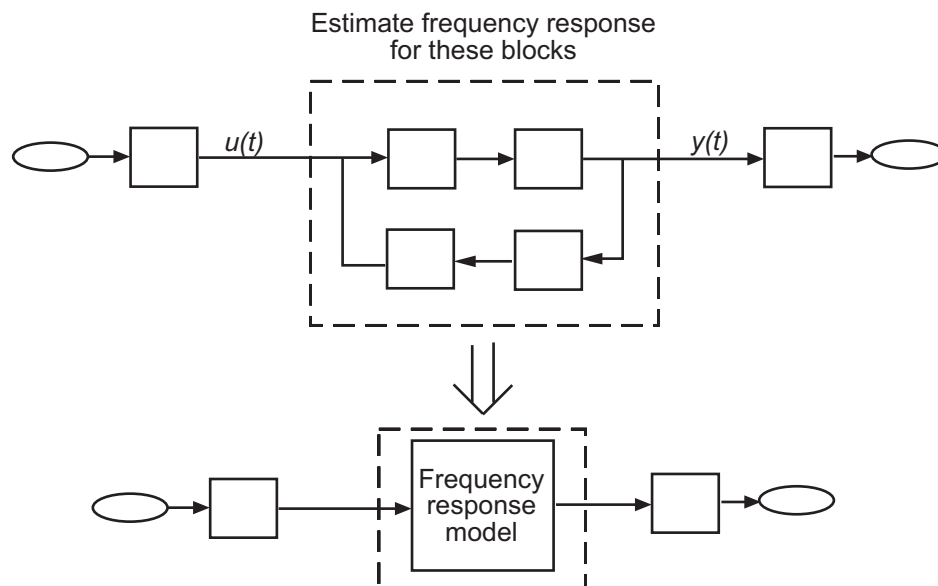
where $Y(s)$ and $U(s)$ are the Laplace transforms of $y(t)$ and $u(t)$, respectively.

Overview of Frequency Response Estimation in Simulink

- “Model Requirements” on page 6-4
- “Generating Input and Output Signals for Estimation” on page 6-5
- “Using Estimated Frequency Response Model” on page 6-6
- “Do Open-Loop Linearization I/O Points Impact Simulation?” on page 6-6
- “Simulating Output from MIMO Systems” on page 6-6

Model Requirements

You can use `frestimate` to estimate the frequency response of one or more blocks in a stable Simulink model at steady state. Frequency response estimation is available for any Simulink blocks—including blocks that contain event-based dynamics, such as Stateflow charts, triggered subsystems, pulse width modulation (PWM) signals.



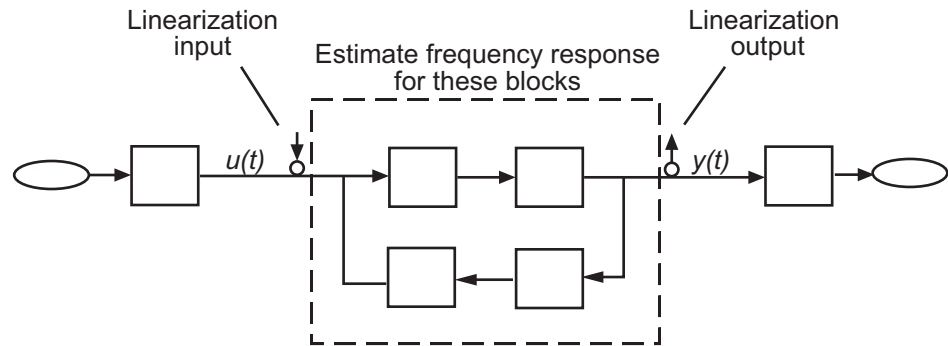
If you have source blocks that generate time-varying outputs in your model, you must disable the corresponding blocks before estimation because such signals interfere with the estimation. You can disable a block by opening the loop at the block output, as described in “Estimating Frequency Response” on page 6-17.

`frestimate` provides more accurate results when your model does not include blocks that simulate significant random disturbances. To learn more about modeling systems with noise using other products, see “Estimating Models With Noise” on page 6-39.

Generating Input and Output Signals for Estimation

`frestimate` automates the process of generating output signals in response to specific input signals for estimating the frequency response. Simulink Control Design lets you conveniently inject input signals anywhere in your model and log the simulated output, without requiring you to modify your model.

You design the signal that `frestimate` injects at the linearization input point to excite the model at frequencies of interest, such as a chirp or sinestream signal. A *sinestream* signal consists of a series of sinusoids, where each sine wave excites the system for a period of time. `frestimate` adds the signal you design to existing Simulink signals at the linearization input point, and simulates the model to obtain the output at the linearization output point.



For sinestream input, `frestimate` performs signal postprocessing to improve the accuracy of your model before estimation. For more information about available input signals and their impact on the estimation algorithm, see “Creating Input Signals for Estimation” on page 6-7.

`frestimate` estimates the frequency response as:

$$G(s) \approx \frac{\text{fast Fourier transform of } y_{est}(t)}{\text{fast Fourier transform } u_{est}(t)}$$

where $u_{est}(t)$ is the injected input signal and $y_{est}(t)$ is the corresponding simulated output signal.

For more information about estimating frequency response models, see “Estimating Frequency Response” on page 6-17.

Using Estimated Frequency Response Model

You can use the estimated frequency response to validate the results of exact linearization. Frequency response estimation uses a different algorithm to compute a linear model approximation, which serves as an independent test of exact linearization results. For more information, see “Validating Exact Linearization Results” on page 4-76.

The estimated frequency response model is an `frd` object. You can design a controller for this model using Control System Toolbox software.

You can also get parametric models using the System Identification Toolbox™ software, as described in “Example—Estimating State-Space Model Using System Identification Toolbox” on page 6-41.

Do Open-Loop Linearization I/O Points Impact Simulation?

`frestimate` correctly handles open-loop linearization input and output points. For example, if the input linearization point is open, `frestimate` adds the signal you design to the constant operating point value, and simulates the model output at the linearization output point. The operating point is the initial output of the block with a loop opening.

Simulating Output from MIMO Systems

For MIMO systems, `frestimate` injects the signal at each input channel separately to simulate the corresponding output signals. The estimation algorithm uses the inputs and the simulated outputs to compute the MIMO frequency response.

If you want to inject different input signal at the linearization input points of a multiple-input system, treat your system as separate single-input systems. Perform independent frequency response estimations for each linearization input point using `frestimate`, and concatenate your frequency response results.

Creating Input Signals for Estimation

In this section...
“Supported Input Signals” on page 6-7
“Creating the Sinestream Input Signal” on page 6-7
“Creating the Chirp Input Signal” on page 6-14

Supported Input Signals

Frequency response estimation using `frestimate` is optimized for the sinestream and chirp input signals. For information about other requirements, see “Model Requirements” on page 6-4.

Sinusoidal Signal	When to Use
Sinestream	<p>Recommended for most situations. Especially useful when:</p> <ul style="list-style-type: none"> Your system contains strong nonlinearities. You require highly accurate frequency response models. <p>See “Creating the Sinestream Input Signal” on page 6-7.</p>
Chirp	<ul style="list-style-type: none"> Your system is very linear in the simulation range. You do not require highly accurate frequency response models. <p>See “Creating the Chirp Input Signal” on page 6-14.</p>

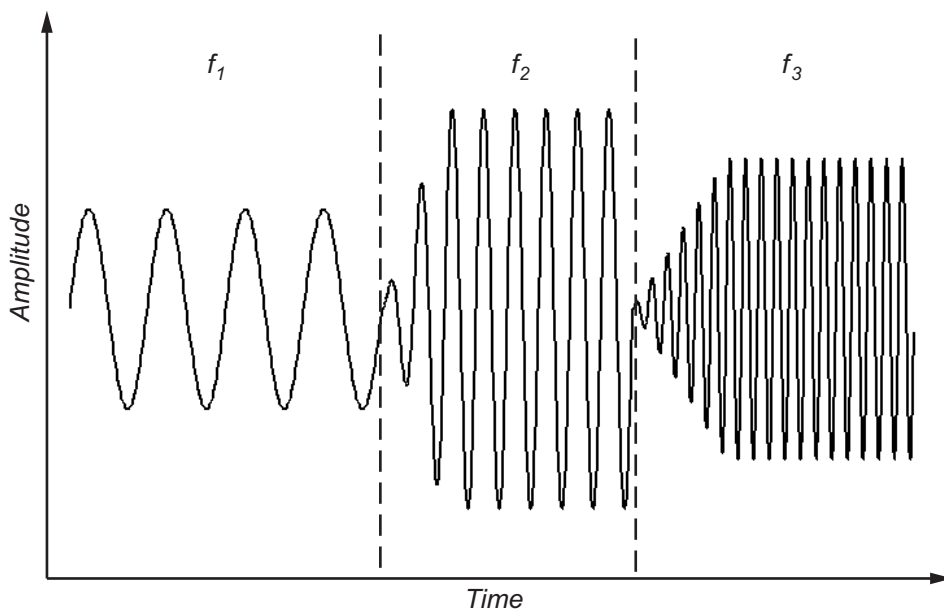
Creating the Sinestream Input Signal

- “What Is a Sinestream Signal?” on page 6-8
- “Designing the Sinestream Signal” on page 6-8

- “How Frequency Response Estimation Treats Sinestream Inputs” on page 6-11

What Is a Sinestream Signal?

Sinestream consists of several adjacent sine waves of varying frequencies. Each frequency excites the system for a period of time.



Designing the Sinestream Signal

You can create a sinestream signal from both continuous-time and discrete-time signals in Simulink models.

Signal at Input Linearization Point	Command
Continuous	<code>frest.Sinestream</code>
Discrete	<code>frest.createFixedTsSinestream</code>

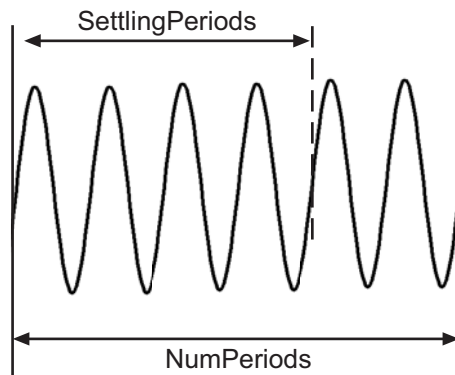
The most efficient way to create a sinestream signal is to use a linear model that accurately represents your system dynamics:

```
input = frest.Sinestream(sys)
```

`sys` can be the linear model you obtained using exact linearization techniques (see Chapter 4, “Exact Linearization Using the GUI”). You can also define a linear system based on your insight about the system using the `tf`, `zpk`, and `ss` commands.

`frest.Sinestream` uses the linear system to determine these signal characteristics:

- Frequencies at which the linear system has interesting dynamics (Frequency option).
- Number of periods for the system to reach steady state at each frequency (SettlingPeriods option).
- Total number of periods for each frequency (NumPeriods option).



For example, create a sinestream signal from a linearized model:

```
magball
io(1) = linio('magball/Desired Height',1);
io(2) = linio('magball/Magnetic Ball Plant',...
              1,'out');
sys = linearize('magball',io);
input = frest.Sinestream(sys)
```

The resulting input signal contains 26 frequency values in `Frequency`. `frest.Sinestream` automatically specifies `NumPeriods` and `SettlingPeriods` for each frequency:

```
Frequency           : [0.05786;0.092031;0.14638 ...] (rad/s)
Amplitude           : 1e-005
SamplesPerPeriod    : 40
NumPeriods          : [4;4;4;4 ...]
RampPeriods         : 0
FreqUnits (rad/s,Hz): rad/s
SettlingPeriods     : [1;1;1;1 ...]
ApplyFilteringInFRESTIMATE (on/off) : on
SimulationOrder (Sequential/OneAtATime): Sequential
```

You can plot your input signal using `plot(input)`.

Estimate a frequency response model to evaluate the quality of your input signal. If the output at a specific frequency did not reach steady state, you can modify the characteristics of the input signal at the corresponding frequency. For example, you can modify `NumPeriods` and `SettlingPeriods`:

```
input.NumPeriods(index)=NewNumPeriods;
input.SettlingPeriods(index)=NewSettlingPeriods;
```

where `index` is the frequency value index of the sine wave you want to modify. `NewNumPeriods` and `NewSettlingPeriods` are the new values of `NumPeriods` and `SettlingPeriods`, respectively.

To modify several signal properties at a time, it is more convenient to use `set`. For example:

```
input = set(input,'NumPeriods',NewNumPeriods,...
             'SettlingPeriods',NewSettlingPeriods)
```

After modifying the input signal, repeat the estimation.

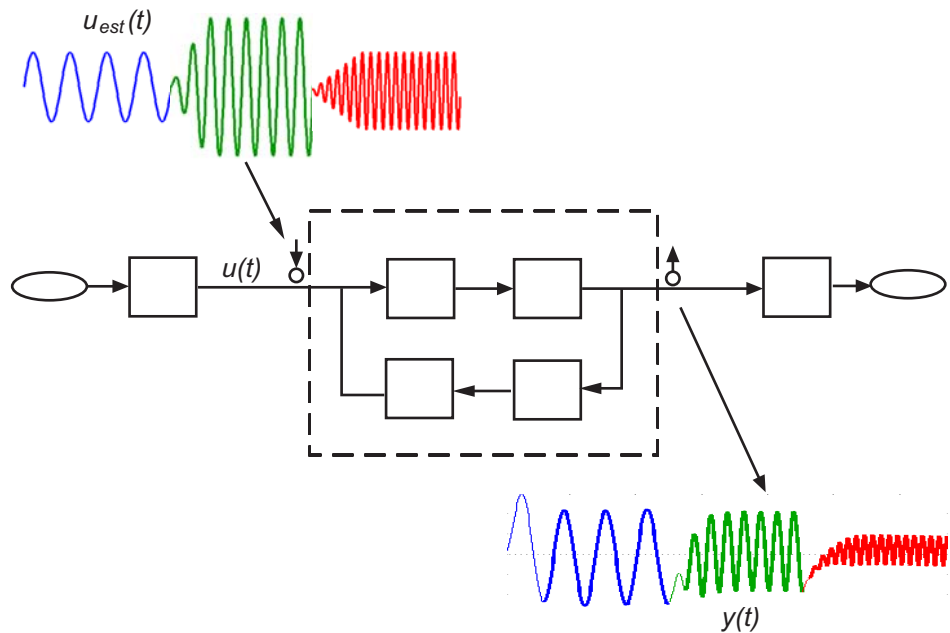
For more information about `sinestream` options, see the `frest.Sinestream` reference page.

How Frequency Response Estimation Treats Sinestream Inputs

`frestimate` performs the following operations when you use the sinestream input signal:

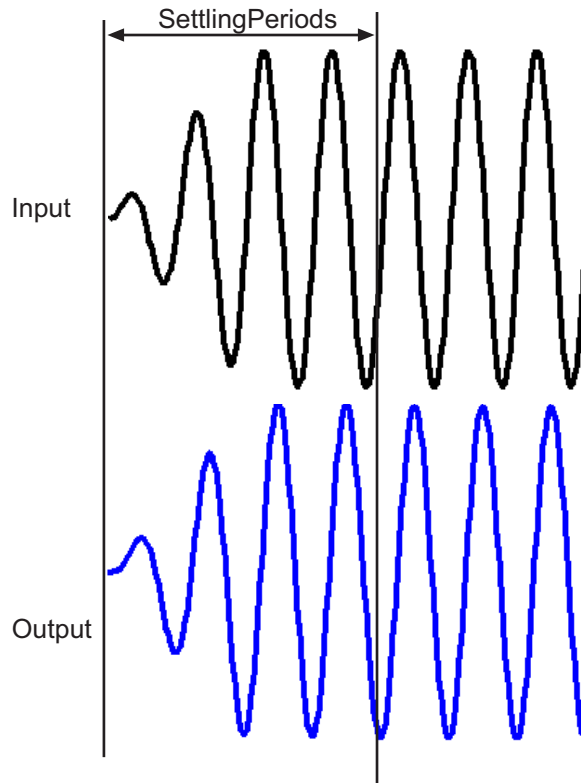
- 1 Injects the sinestream input signal you design, $u_{est}(t)$, at the linearization input point.
- 2 Simulates the output at the linearization output point.

`frestimate` adds the signal you design to existing Simulink signals at the linearization input point.



- Discards the `SettlingPeriods` portion of the output (and the corresponding input) at each frequency.

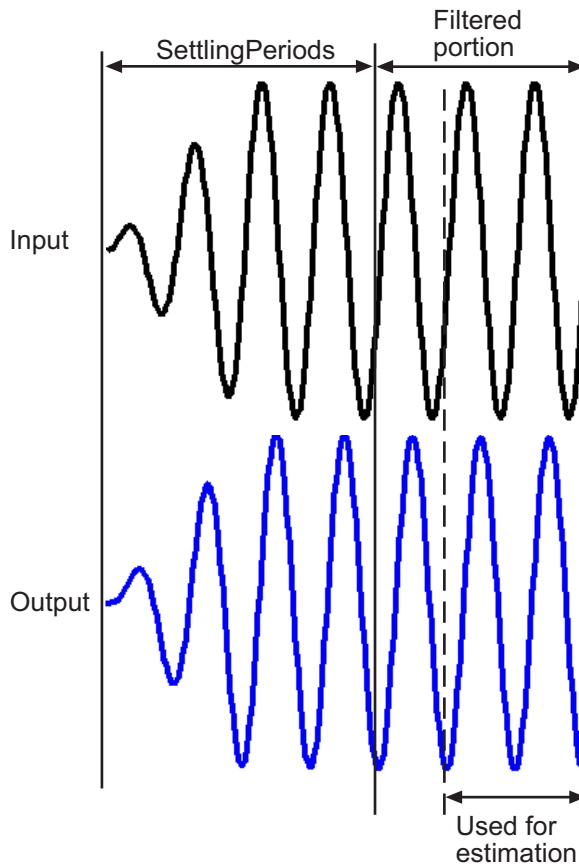
The simulated output at each frequency has a transient portion and steady state portion. `SettlingPeriods` corresponds to the transient components of the output and input signals. The periods following `SettlingPeriods` are considered to be at steady state.



- Filters the remaining portion of the output and the corresponding input signals at each input frequency using a bandpass filter. Because most models are not at steady state, the response might contain low-frequency transient behavior. Filtering typically improves the accuracy of your model by removing the effects of frequencies other than the input frequencies,

which are problematic when sampling and analyzing data of finite length. These effects are called *spectral leakage*.

Any transients associated with filtering are only in the first period of the filtered steady-state output. After filtering, `frestimate` discards the first period of the input and output signals. `frestimate` uses a finite impulse response (FIR) filter, whose order matches the number of samples in a period.



- 5 Estimates the frequency response of the processed signal by computing the ratio of the fast Fourier transform of the filtered steady-state portion

of the output signal $y_{est}(t)$ and the fast Fourier transform of the filtered input signal $u_{est}(t)$:

$$\text{Frequency Response Model} = \frac{\text{fft of } y_{est}(t)}{\text{fft of } u_{est}(t)}$$

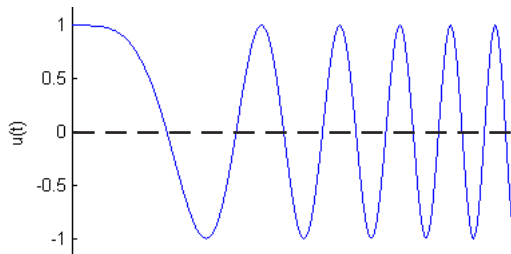
To compute the response at each frequency, `frestimate` uses only the simulation output at that frequency.

Creating the Chirp Input Signal

What Is a Chirp Signal?

The swept-frequency cosine (chirp) input signal excites your system at a range of frequencies, such that each frequency is active for an instant.

Alternatively, you can use the `sinestream` signal, which excites the system at each frequency for several periods. See “Supported Input Signals” on page 6-7 for more information about choosing your signal.



Designing the Chirp Signal

The most efficient way to create a chirp signal is to use a linear model that accurately represents your system dynamics:

```
input = frest.Chirp(sys)
```

`sys` can be the linear model you obtained using exact linearization techniques (see Chapter 4, “Exact Linearization Using the GUI”). You can also define a linear system based on your insight about the system using the `tf`, `zpk`, and `ss` commands.

`frest.Chirp` uses the linear system to determine these signal characteristics:

- Frequency range at which the linear system has interesting dynamics (`FreqRange` option).
- Sampling time of the signal (`Ts` option). To avoid aliasing, the Nyquist frequency of the signal is five times the upper end of the frequency range,

$$\frac{2\pi}{5 * \max(\text{FreqRange})}$$

- Number of samples in the signal is such that the frequency response estimation includes the lower end of the frequency range (`NumSamples` option).

For example, create a chirp signal from a linearized model:

```
magball
io(1) = linio('magball/Desired Height',1);
io(2) = linio('magball/Magnetic Ball Plant',...
             1,'out');
sys = linearize('magball',io);
input = frest.Chirp(sys)
```

The input signal is:

```

FreqRange           : [0.0578598408615998 10065.3895573969] (rad/s)
Amplitude           : 1e-005
Ts                  : 0.00012484733494616 (sec)
NumSamples          : 869808
InitialPhase        : 270 (deg)
FreqUnits (rad/s or Hz): rad/s
SweepMethod(linear/ : linear
                  quadratic/
                  logarithmic)
```

You can plot your input signal using `plot(input)`.

Estimate a frequency response model to evaluate the quality of your input signal. In some cases, the chirp signal might not lead to accurate frequency response estimation because it is sweeping through the frequency range too quickly. Such fast sweeps might not provide sufficient time for the output to reach steady state. You can typically improve the quality of the estimated frequency response by increasing the number of samples in the chirp signal:

```
input.NumSamples = NewNumSamples;
```

where *NewNumSamples* is the new value of NumSamples.

To modify several signal properties at a time, it is more convenient to use `set`. For example:

```
input = set(input, 'NumSamples', NewNumSamples, ...  
             'FreqRange', NewFreqRange)
```

After modifying the input signal, repeat the estimation.

For more information about chirp signal properties, see the `frest.Chirp` reference page.

Estimating Frequency Response

Prerequisites

- Open the Simulink model. For example:

```
mdl = 'f14';  
open_system(mdl)
```

To learn more about general model requirements, see “About Frequency Response Estimation” on page 6-2.

- (Optional) If your model is not at steady state, initialize the model at a steady state operating point. See `operspec` and `findop` reference pages. You can check whether your model is at steady state by simulating the model.
- Create an input signal for estimation. See “Creating Input Signals for Estimation” on page 6-7. For example:

```
io(1) = linio('f14/Sum1',1)  
io(2) = linio('f14/Gain5',1,'out')  
sys = linearize('f14',io);  
input = frest.Sinestream(sys)
```

When using the `sinestream` signal, you can specify to disable filtering during estimation using the signal `ApplyFilteringInFRESTIMATE` property. For more information, see “Example—Effects of Filtering on Frequency Response Estimation” on page 6-30.

- 1 Specify the Simulink blocks to approximate with the estimated frequency response using linearization I/O points. For example:

```
io(1) = linio('f14/Sum1',1)  
io(2) = linio('f14/Gain5',1,'out')
```

Caution Avoid placing I/O points on bus signals.

For more information about linearization I/O points, see “Choosing and Storing Linearization Points” on page 5-4 and the `linio` reference page.

- 2 Disable all source blocks that generate time-varying signals, which interfere with the signal you create for estimation. Such source blocks can result in inaccurate frequency response estimation.

To disable blocks from the Simulink model window, right-click the block output signal and select **Linearization Points > Open Loop**. Alternatively, use the `linio` command.

For example, these commands disable the Pilot and Wind Gust Disturbance source blocks in the `f14` model:

```
io(3) = linio('f14/Pilot',1,'none','on')
io(4) = linio('f14/Dryden Wind Gust Models',1,'none','on')
io(5) = linio('f14/Dryden Wind Gust Models',2,'none','on')
```

- 3 Estimate the frequency response. For example:

```
[sysest,simout] = frestimate('f14',io,input)
```

To initialize your system at an operating point *op* before estimating the frequency response, use this syntax:

```
[sysest,simout] = frestimate('model',op,io,input)
```

where *model* is the model name and *op* is the steady-state operating point. *sysest* is the estimated frequency response. *simout* is the simulated output that is a `Simulink.timeseries` object.

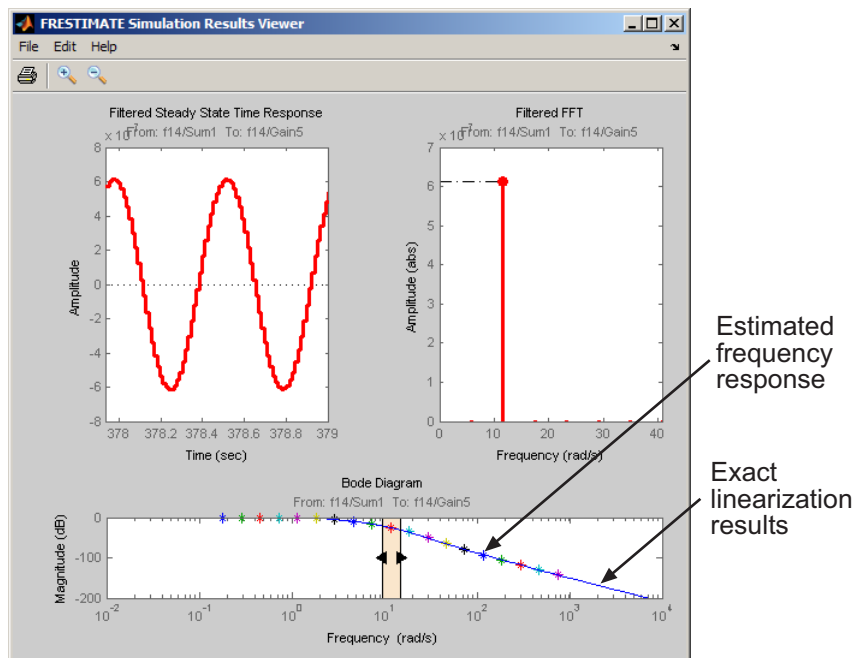
For more information about syntax and argument descriptions, see the `frestimate` reference page.

- 4 Open the Simulation Results Viewer to analyze the estimated frequency response. For example:

```
frest.simView(simout,input,syseset);
```

You can compare the estimated frequency response (syseset) to a system you linearized using exact linearization (sys):

```
frest.simView(simout,input,syseset,sys);
```



For more information, see “Analyzing Estimated Frequency Response” on page 6-20.

Analyzing Estimated Frequency Response

In this section...
“Opening the Simulation Results Viewer” on page 6-20
“Interpreting the Frequency Response Analysis Plots” on page 6-21
“Analyzing the Simulated Output and FFT at Specific Frequencies” on page 6-23
“Annotating Data on Plots Using Data Tips” on page 6-24
“Displaying Frequency Response of MIMO Systems” on page 6-25

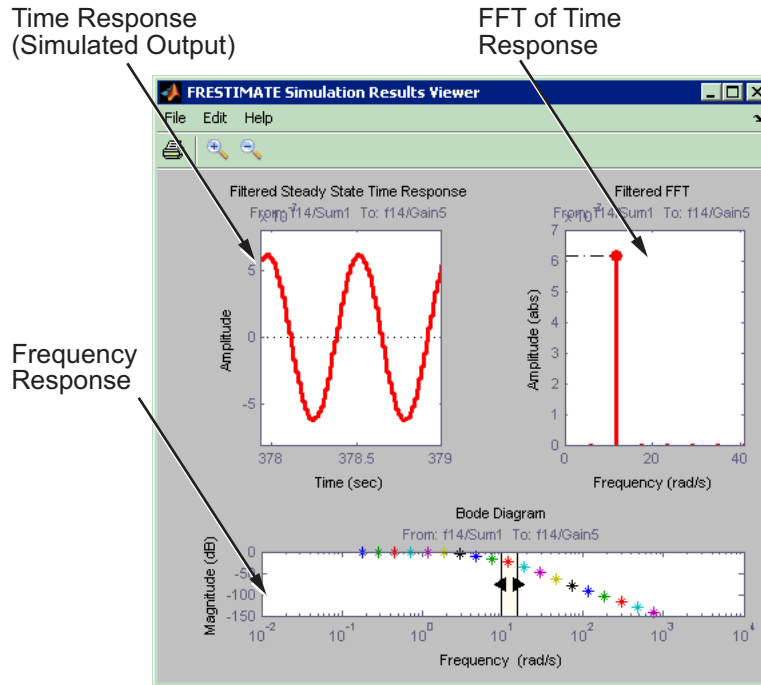
Opening the Simulation Results Viewer

Use the Simulation Results Viewer to analyze the results of your frequency response estimation, obtained by performing the steps in “Estimating Frequency Response” on page 6-17.

Open the Simulation Results Viewer:

```
frest.simView(simout,input,sysest)
```

where *simout* is the simulated output, *input* is the input signal you created, and *sysest* is the estimated frequency response.



Interpreting the Frequency Response Analysis Plots

By default, the Simulation Results Viewer shows these plots:

- “Frequency Response” on page 6-22
- “Time Response (Simulated Output)” on page 6-22
- “FFT of Time Response” on page 6-22

Toggle the display of specific plots in the Simulation Results Viewer by selecting the corresponding plot from the **Edit > Plots** menu. To modify plot settings, such as axis frequency units, right-click a plot and select the corresponding option.

Frequency Response

Use the Bode plot to analyze the frequency response. If the frequency response does not match the dynamics of your system, see “Troubleshooting a Frequency Response Estimation” on page 6-26 for information about possible causes and solutions. While troubleshooting, you can use the Bode plot controls to view the time response at the problematic frequencies.

Typical ways to improve estimation results involve modifying your input signal or disabling the model blocks that drive your system away from the operating point, and repeating the estimation.

Time Response (Simulated Output)

Use this plot to check whether the simulated output is at steady state at specific frequencies. If the response is not at steady state, see “Time Response Not at Steady State” on page 6-26 for possible causes and solutions.

If you used the `sinestream` input for estimation, check both the filtered and the unfiltered time response. You can toggle the display of filtered and unfiltered output by right-clicking the plot and selecting **Show filtered steady state output only**. If both the filtered and unfiltered response are at steady state, then your model is at steady state and you can explore other possible causes in “Troubleshooting a Frequency Response Estimation” on page 6-26.

Note If you used the `sinestream` input for estimation, toggling the filtered and unfiltered display only updates the Time Response and FFT plots. This selection does not change estimation results. For more information about filtering during estimation, see “How Frequency Response Estimation Treats Sinestream Inputs” on page 6-11.

FFT of Time Response

Use this plot to analyze the spectrum of the simulated output.

For example, you can use the spectrum to identify strong nonlinearities. When the FFT plot shows large amplitudes at frequencies other than the input signal, your model is operating outside of linear range. If you are interested in analyzing the linear response of your system for small perturbations, explore

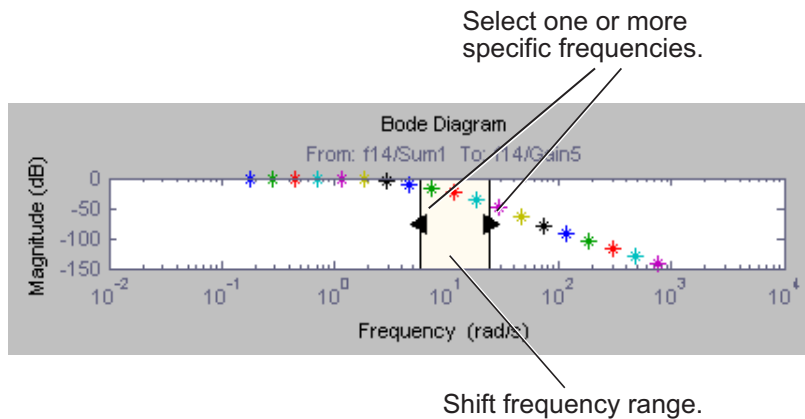
possible solutions in “FFT Contains Large Harmonics At Frequencies Other Than The Input Signal Frequency” on page 6-32.

Analyzing the Simulated Output and FFT at Specific Frequencies

In the Simulation Results Viewer, use Bode controls to display the simulated output and its spectrum at specific frequencies.

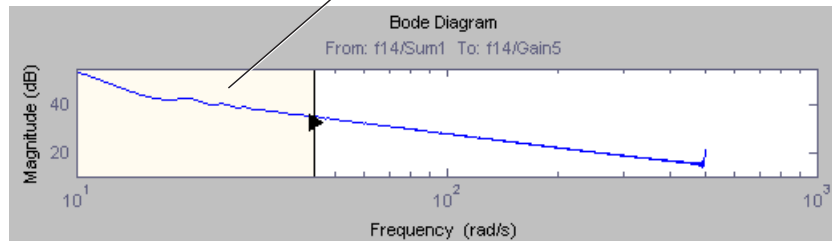
If you used the sinestream input signal in the estimation:

- Drag arrows individually to display the time response and FFT at specific frequencies.
- Drag the shaded region to shift the time response and FFT to a different frequency range.



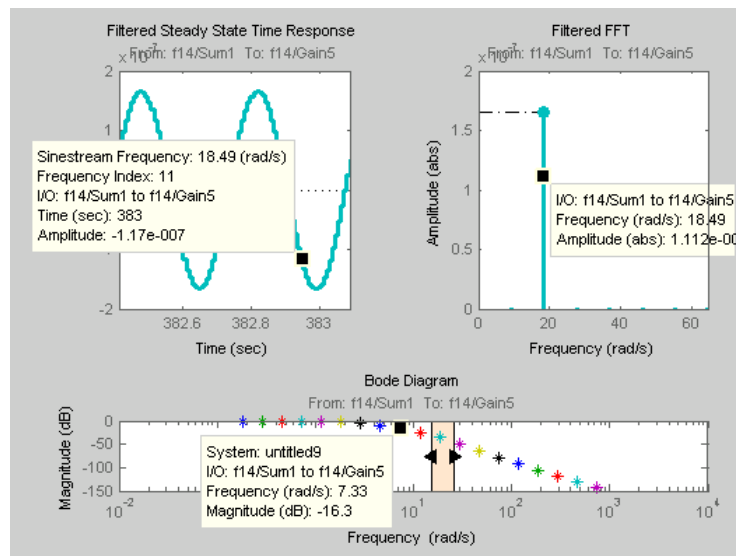
If you used the chirp input signal in the estimation, drag the shaded region to increase or decrease the frequency range of the displayed time response and FFT.

Increase or decrease selected frequency range.



Annotating Data on Plots Using Data Tips

You can display a data tip on the Time Response, FFT, and Bode plots in the Simulation Results Viewer by clicking the corresponding curve. Dragging the data tip updates the information.



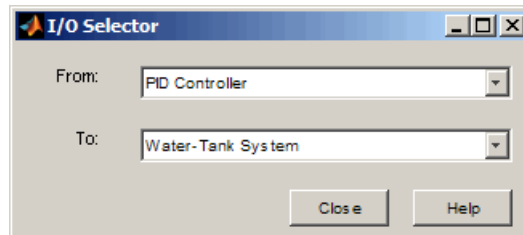
If you used several frequencies in your sinestream input signal and you need to modify the signal at a specific frequency, use the data tip to identify the frequency index. For example, the Time Response data tip shows that the frequency index is 11. You can modify the NumPeriods and SettlingPeriods of this input:

```
input.NumPeriods(11) = 80;  
input.SettlingPeriods(11) = 75;
```

Displaying Frequency Response of MIMO Systems

For MIMO systems, view frequency response information for specific input and output channels:

- 1 In the Simulation Results Viewer, right-click any plot, and select **I/O Selector**.
- 2 Choose the input channel in the **From** list. Choose the output channel in the **To** list.



Troubleshooting a Frequency Response Estimation

In this section...

“Time Response Not at Steady State” on page 6-26

“FFT Contains Large Harmonics At Frequencies Other Than The Input Signal Frequency” on page 6-32

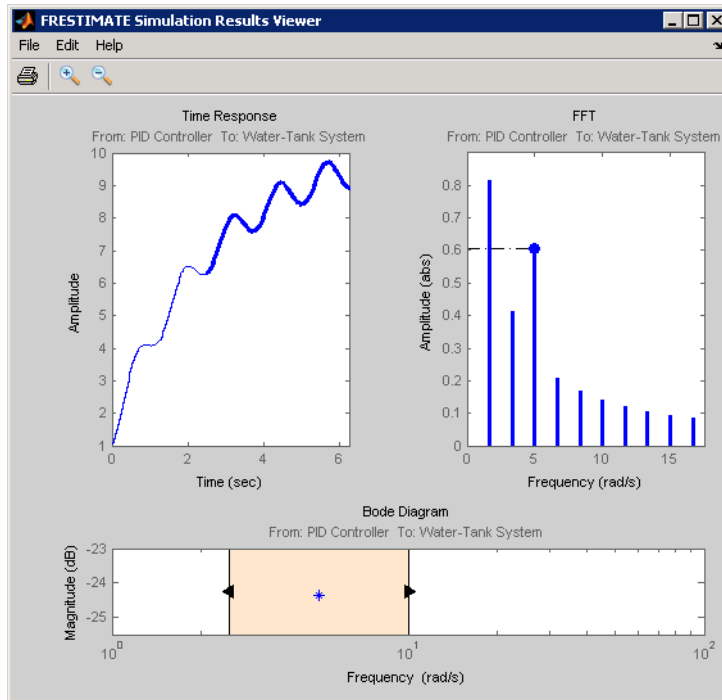
“Time Response Grows Without Bound” on page 6-34

“Time Response Is Discontinuous or Zero” on page 6-35

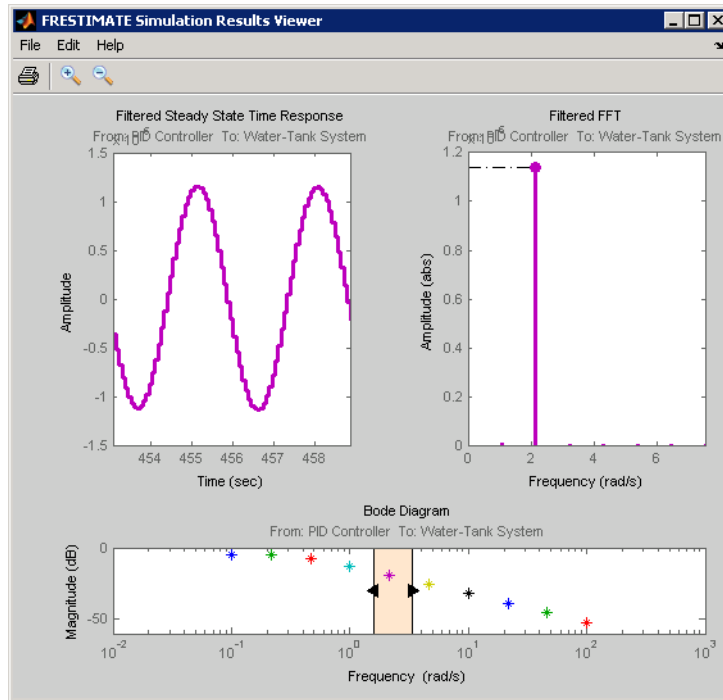
Time Response Not at Steady State

What Does This Mean?

This time response is not at steady state.



This plot shows a steady-state time response.



Because frequency response estimation requires steady-state input and output signals, transients produce inaccurate estimation results.

For sinestream input signals, transients sometimes interfere with the estimation either directly or indirectly through spectral leakage. For chirp input signals, transients slow model response to the instantaneous frequency switching of the chirp signal.

How Do I Fix It?

Try the suggested actions listed the table and repeat the estimation, as described in “Estimating Frequency Response” on page 6-17.

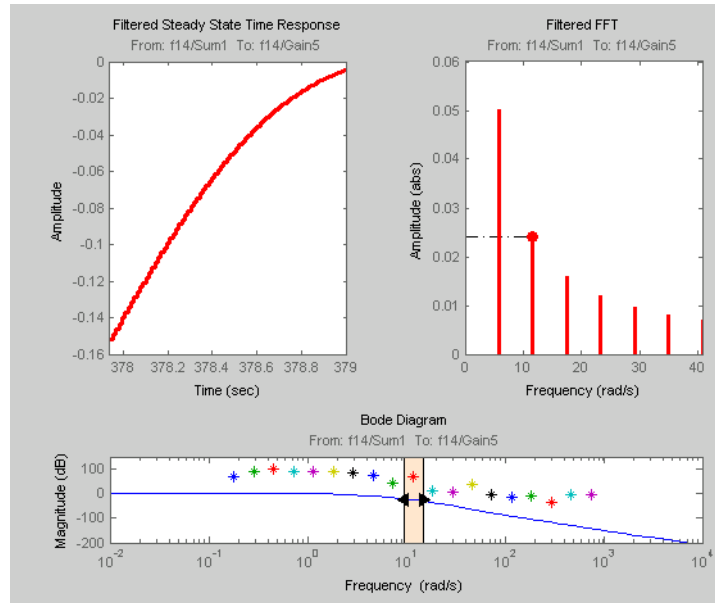
Possible Cause	Action
Model output at the linearization output point is not at steady state.	Disable all source blocks in your model and repeat the estimation using a steady-state operating point. See “Example–Effects of Time-Varying Simulink Signals on Frequency Response Estimation” on page 6-28.
(Sinestream input) Not enough input period for the output to reach steady state.	<ul style="list-style-type: none"> • Increase the number of periods for frequencies that do not reach steady state by changing the NumPeriods and SettlingPeriods. See “Creating the Sinestream Input Signal” on page 6-7. • Check that filtering is enabled during estimation. See “Example–Effects of Filtering on Frequency Response Estimation” on page 6-30.
(Chirp input) Signal is sweeping through the frequency range too quickly.	Increase the simulation time by increasing NumSamples. See “Creating the Chirp Input Signal” on page 6-14.

Example–Effects of Time-Varying Simulink Signals on Frequency Response Estimation

Compare the linear model obtained using exact linearization techniques with the estimated frequency response:

```
mdl = 'f14';
open_system(mdl)
io(1) = linio('f14/Sum1',1)
io(2) = linio('f14/Gain5',1,'out')
sys = linearize(mdl,io);
in = frest.Sinestream(sys);
[sysest,simout] = frestimate(mdl,io,in);
frest.simView(simout,in,sysest,sys)
```

The resulting time response is not at steady state because the Pilot and Wind Gust Disturbance blocks are driving the model away from the operating point. Thus, the linearization results do not match the estimated frequency response.



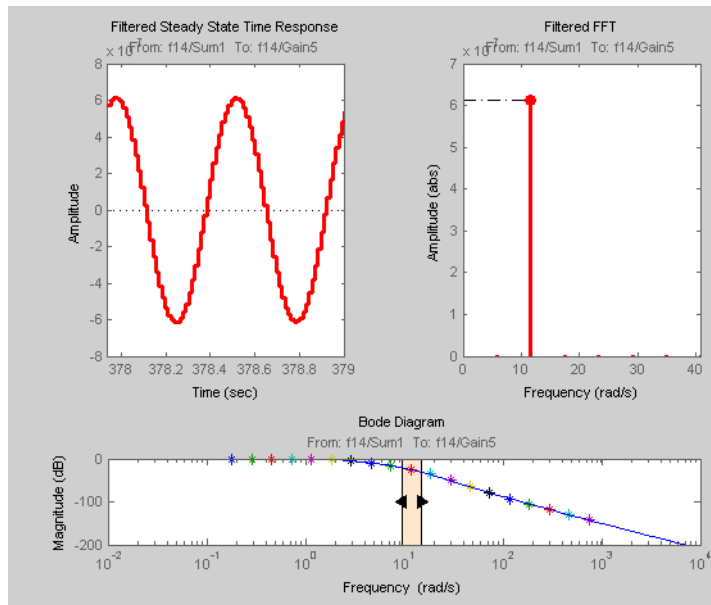
Disable the source blocks by opening the loop at the output of these blocks:

```
io(3) = linio('f14/Pilot',1,'none','on')
io(4) = linio('f14/Dryden Wind Gust Models',1,'none','on')
io(5) = linio('f14/Dryden Wind Gust Models',2,'none','on')
```

Repeat the frequency response estimation:

```
[sysest,simout] = frestimate mdl,io,in;
frest.simView(simout,in,sysest,sys);
```

The resulting frequency response matches the exact linearization results.



Example—Effects of Filtering on Frequency Response Estimation

Open the model, specify linearization I/O points, and define the input signal:

```
magball
io(1) = linio('magball/Desired Height',1);
io(2) = linio('magball/Magnetic Ball Plant',1,'out');
sys = linearize('magball',io);
in = frest.Sinestream(sys);
```

Reduce the number of cycles for the 25th frequency to degrade the quality of the data for this example:

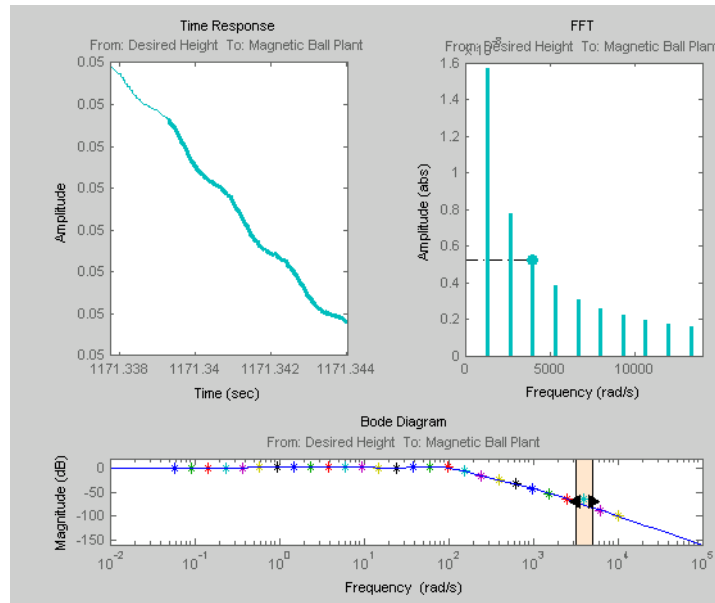
```
numper = in.NumPeriods; settper = in.SettlingPeriods;
numper(25) = 4; settper(25) = 1;
in = set(in,'NumPeriods',numper,...
         'SettlingPeriods',settper);
```

Turn off filtering during estimation and estimate the frequency response:

```
in.ApplyFilteringInFREESTIMATE = 'off';
```

```
[sysest,simout] = frestimate('magball',io,in);
frest.simView(simout,in,sysest,sys)
```

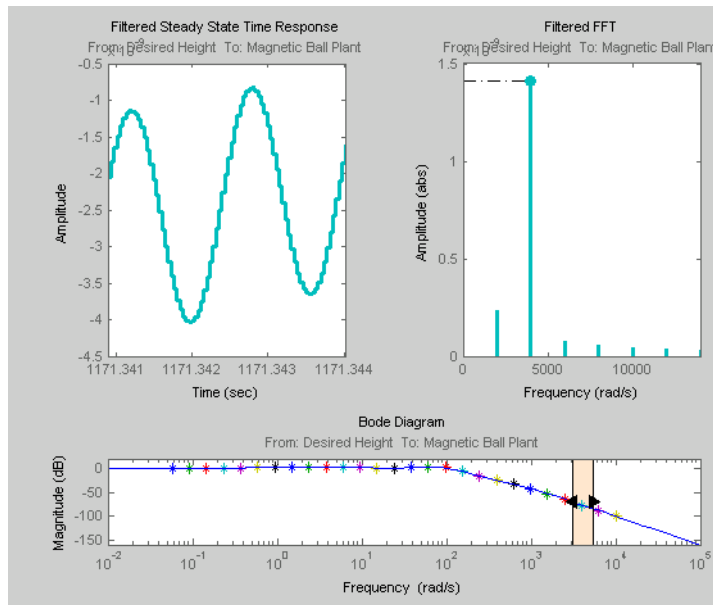
The resulting response deviates from the linearized model. The Simulation Results Viewer shows the time response and its spectrum for the 25th frequency value, which is selected on the Bode plot.



Turn filtering on and repeat the estimation:

```
in.ApplyFilteringInFRESTIMATE = 'on';
[sysest,simout] = frestimate('magball',io,in);
frest.simView(simout,in,sysest,sys)
```

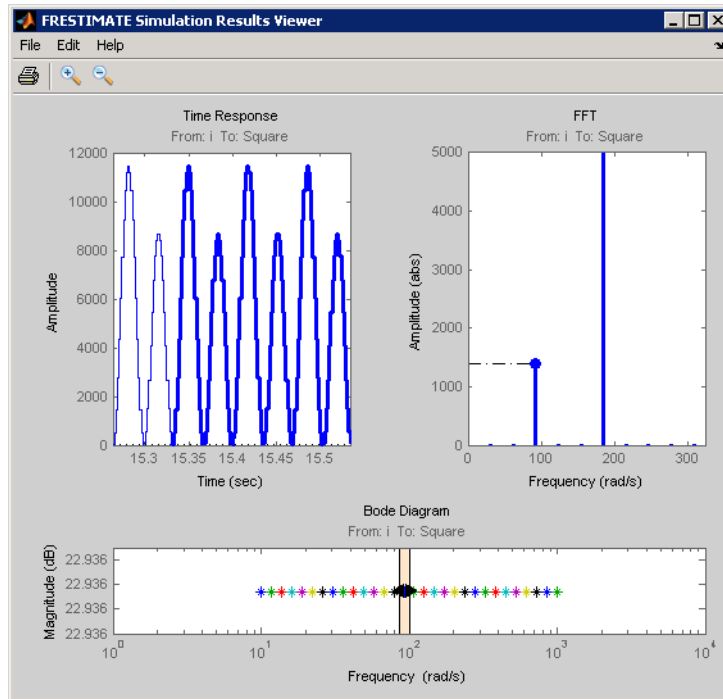
Filtering helps to achieve better agreement between the linearized model and estimated frequency response, as shown on the Bode plot.



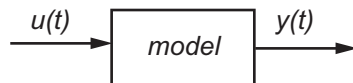
FFT Contains Large Harmonics At Frequencies Other Than The Input Signal Frequency

What Does This Mean?

When the FFT plot shows large amplitudes at frequencies other than the input signal, your model is operating outside the linear range. This is problematic when you want to analyze your linear system response to small perturbations.



For models operating in the linear range, the input amplitude A_1 in $y(t)$ must be larger than the amplitudes of other harmonics, A_2 and A_3 .



$$u(t) = A_1 \sin(\omega_1 + \phi_1)$$

$$y(t) = A_1 \sin(\omega_1 + \phi_1) + A_2 \sin(\omega_2 + \phi_2) + A_3 \sin(\omega_3 + \phi_3) + \dots$$

How Do I Fix It?

Adjust the amplitude of your input signal to decrease the impact of other harmonics, and repeat the estimation, as described in “Estimating Frequency Response” on page 6-17. Typically, you should decrease the input amplitude level to keep the model operating in the linear range.

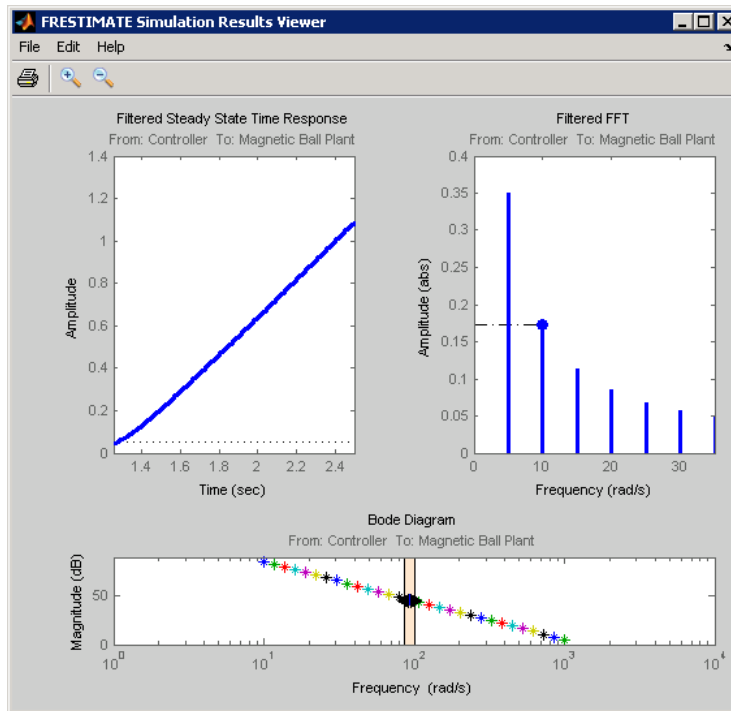
For more information about modifying signal amplitudes, see one of the following reference pages:

- `frest.Sinestream`
- `frest.createFixedTsSinestream`
- `frest.Chirp`

Time Response Grows Without Bound

What Does This Mean?

When the time response grows without bound, the resulting frequency response estimation is inaccurate. Frequency response estimation is only accurate close to the operating point.



How Do I Fix It?

Try the suggested actions listed the table and repeat the estimation, as described in “Estimating Frequency Response” on page 6-17.

Possible Cause	Action
Model is unstable	You cannot estimate the frequency response using <code>frestimate</code> . Instead, use exact linearization to get a linear representation of your model. See Chapter 4, “Exact Linearization Using the GUI” or the <code>linearize</code> reference page.
Stable model is not at steady state.	Disable all source blocks in your model and repeat the estimation using a steady-state operating point, as described in “Estimating Frequency Response” on page 6-17. See “Computing Operating Points from Specifications” on page 3-7.
Stable model captures a growing transient.	If the model captures a growing transient, increase the number of periods in the input signal by changing <code>NumPeriods</code> . Repeat the estimation using a steady-state operating point, as described in “Estimating Frequency Response” on page 6-17.

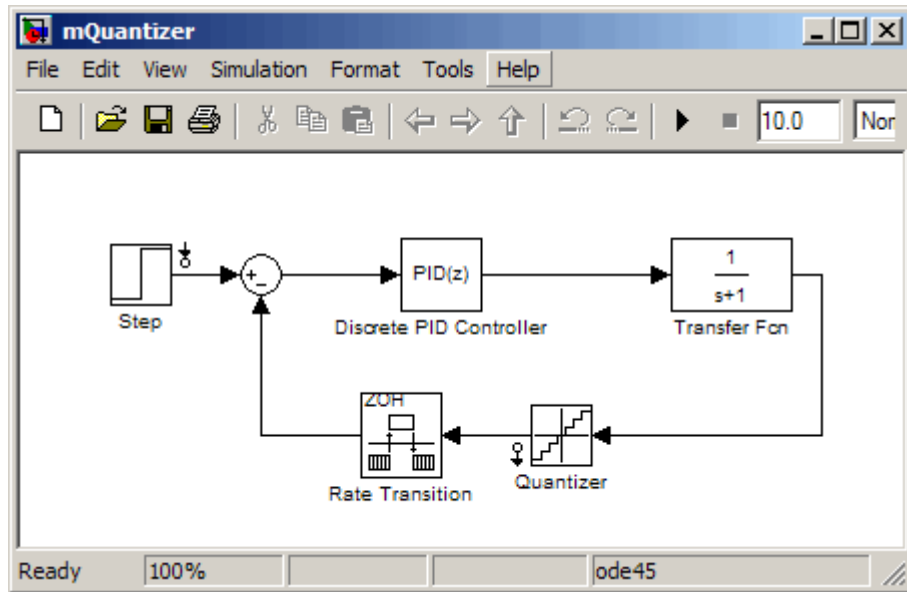
Time Response Is Discontinuous or Zero

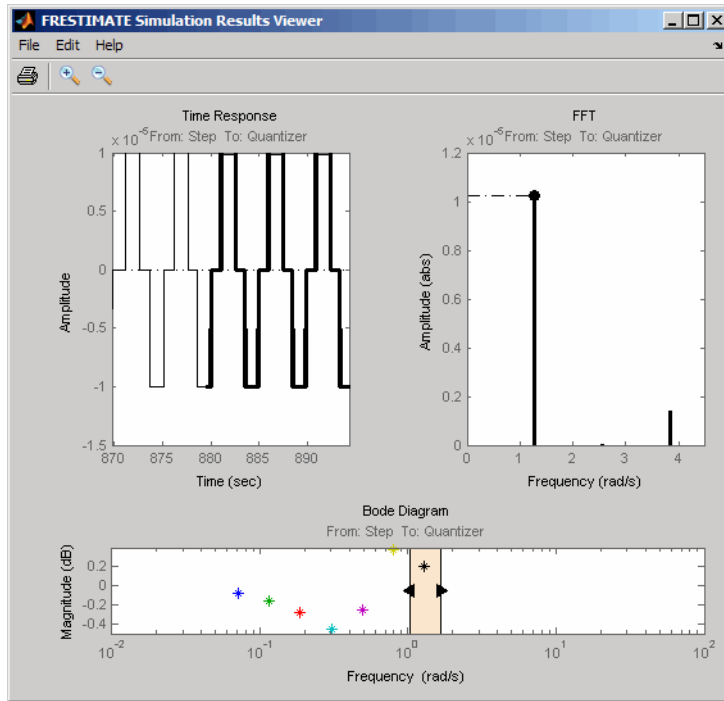
What Does This Mean?

When the time response displays discontinuities or noise, the amplitude of your input signal might be too small to overcome the effects of the discontinuous blocks in your model. Examples of discontinuous blocks include Quantizer, Backlash, and Dead Zones.

If you used a sinestream input signal and estimated with filtering, turn filtering off in the Simulation Results Viewer to see the unfiltered time response.

The following model with a Quantizer block is an example of the impact of an input signal that is too small. When you estimate this model, the unfiltered simulation output includes discontinuities.

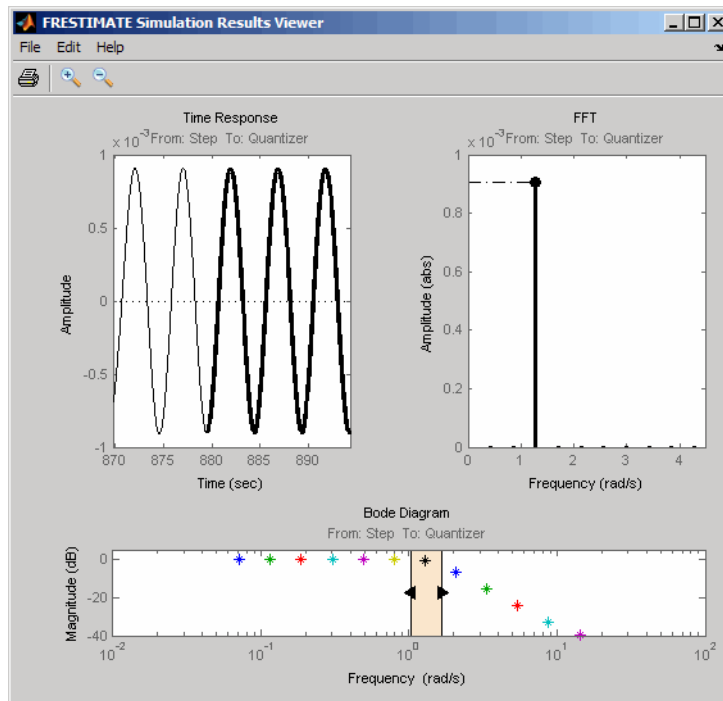




How Do I Fix It?

Increase the amplitude of your input signal and repeat the estimation using frestimate.

With a larger amplitude, the unfiltered simulated output of the model with a Quantizer block is smooth.



For more information about modifying signal amplitudes, see one of the following reference pages:

- `frest.Sinestream`
- `frest.createFixedTsSinestream`
- `frest.Chirp`

Estimating Models With Noise

If your Simulink model contains blocks that model noise, you can do one of the following:

- Disable the blocks that simulate noise by opening the loop at each block output.
- Use `frestimate` to generate input and simulate output signals from your Simulink model for estimation. Then, use the Signal Processing Toolbox™ or System Identification Toolbox software to estimate a model.

Generating Data for Modeling Noise

Use `frestimate` to conveniently simulate the output signal from your Simulink model in response to a specified input signal—without modifying your model.

- 1 Create a random input signal. For example:

```
in = frest.Random('Ts',0.001,'NumSamples',1e4);
```

You can also specify your own custom signal as a `timeseries` object. For example:

```
t = 0:0.001:10;  
y = sin(2*pi*t);  
in_ts = timeseries(y,t);
```

- 2 Simulate the model to obtain the output signal. For example:

```
[sysest,simout] = frestimate(model,op,io,in_ts)
```

The second output argument of `frestimate`, `simout`, is a `Simulink.Timeseries` object that stores the simulated output. `in_ts` is the corresponding input data.

3 Generate `timeseries` objects before using with other MathWorks products:

```
input = generateTimeseries(in_ts);
output = simout{1}.Data;
```

You can use data from `timeseries` objects directly in Signal Processing Toolbox software, or convert these objects to System Identification Toolbox data format.

Example—Estimating a Model With Noise Using Signal Processing Toolbox

Open the Simulink model and specify which portion of the model to linearize:

```
magball
io(1) = linio('magball/Desired Height',1);
io(2) = linio('magball/Magnetic Ball Plant',1,'out');
```

Create a random input signal for simulation:

```
in = frest.Random('Ts',0.001,'NumSamples',1e4);
```

Linearize the model at a steady-state operating point:

```
op = findop('magball',operspec('magball'),...
           linoptions('DisplayReport','off'));
sys = linearize('magball',io,op);
```

Simulate the model to obtain the output at the linearization output point:

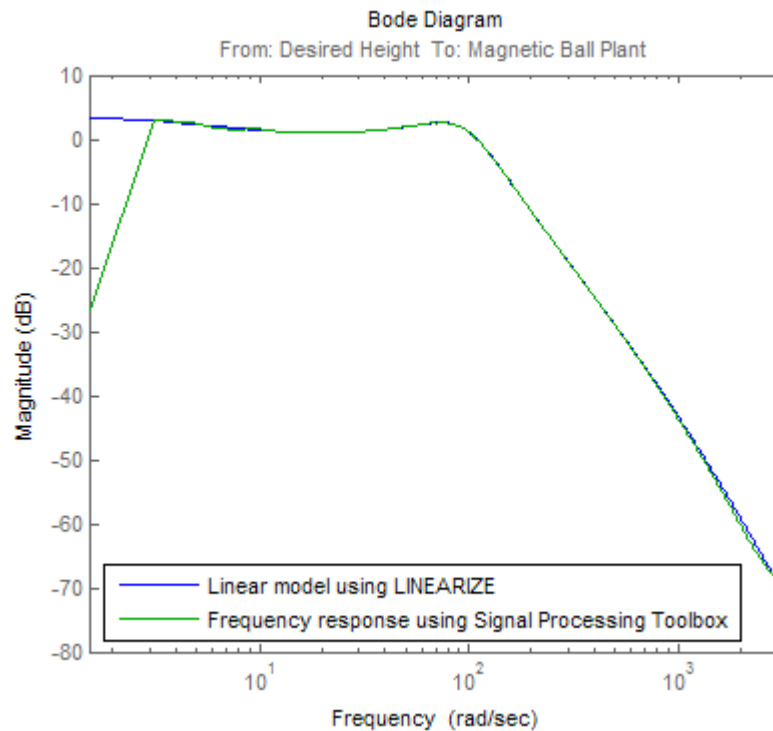
```
[sysest,simout] = frestimate('magball',io,in,op);
```

Estimate a frequency response model using Signal Processing Toolbox software, which includes windowing and averaging:

```
input = generateTimeseries(in);
output = detrend(simout{1}.Data,'constant');
[Txy,F] = tfestimate(input.Data(:),...
                    output,hanning(4000),[],4000,1/in.Ts);
systfest = frd(Txy,2*pi*F);
figure;
```

Compare the results of analytical linearization and `tfestimate`:

```
bodemag(sys,'b',systfest,'g',systfest.Frequency)
legend('Linear model using LINEARIZE',...
      'Frequency response using Signal Processing Toolbox',...
      'Location','SouthWest')
```



The Signal Processing Toolbox command `tfestimate` gives a more accurate estimation than `frestimate`.

Example—Estimating State-Space Model Using System Identification Toolbox

Open the Simulink model and specify which portion of the model to linearize:

```
magball
io(1) = linio('magball/Desired Height',1);
io(2) = linio('magball/Magnetic Ball Plant',1,'out');
```

Compute the steady-state operating point and linearize the model:

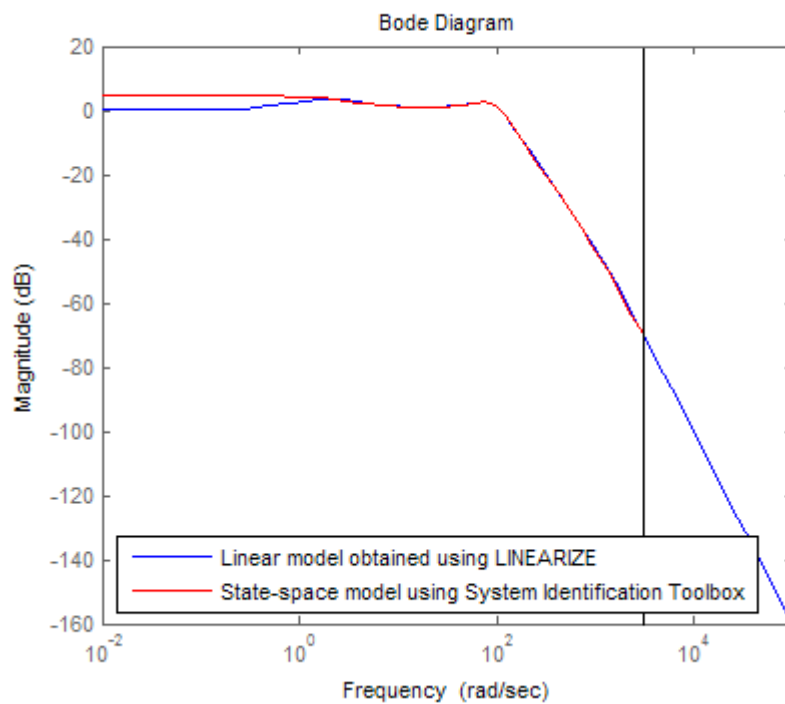
```
op = findop('magball',operspec('magball'),...
           linoptions('DisplayReport','off'));
sys = linearize('magball',io,op);
```

Create a chirp signal and use it to estimate the frequency response:

```
in = frest.Chirp('FreqRange',[1 1000],...
                'Ts',0.001,...
                'NumSamples',1e4);
[~,simout] = frestimate('magball',io,op,in);
```

Use System Identification Toolbox software to estimate a fifth-order state-space model. Compare the results of analytical linearization and the state-state model:

```
input = generateTimeseries(in);
output = simout{1}.Data;
data = iddata(output,input.Data(:),in.Ts);
sys_id = n4sid(detrend(data),5,'cov','none');
bodemag(sys,ss(sys_id('measured')),'r')
legend('Linear model obtained using LINEARIZE',...
       'State-space model using System Identification Toolbox',...
       'Location','SouthWest')
```



Designing Compensators

- “Compensator Design Using Simulink” on page 7-2
- “Automatic PID Tuning” on page 7-4
- “Design and Analysis of Control Systems” on page 7-16

Compensator Design Using Simulink

Simulink Control Design provides two tools to tune Simulink blocks, such as Transfer function and PID Controller blocks:

- PID Tuner
- SISO Design Tool

Use the following table to determine which tool supports what you want to do.

	PID Tuner	SISO Design Tool
Supported Blocks	PID Controller PID Controller 2DOF	Linear blocks
Loop Structure	Single-loop control systems	Single- or multi-loop control systems
Control Design Approach	Simple automatic PID gain tuning by specifying system response time and stability margins	Graphically tune poles and zeros on design plots, such as Bode, root locus, and Nichols Use a PID, LQG, IMC, Robust Control Loop Shaping, and Simulink Design Optimization automated tuning method
Analysis of Control System Performance	Step response for reference tracking and disturbance rejection Open-loop Bode and Nichols charts	Any combination of responses for any input reference or disturbance in your Simulink model using SISO Tool LTIViewer

If you have a nonlinear plant, the software automatically linearizes your model before tuning the controller blocks. You can specify the linearization conditions to ensure that your plant linearizes in the appropriate operating

range. For more information, see Chapter 4, “Exact Linearization Using the GUI”.

Automatic PID Tuning

In this section...
“About Automatic PID Tuning” on page 7-4
“Tuning One-Degree-of-Freedom PID Controllers” on page 7-4
“Tuning Two Degree-of-Freedom PID Controllers” on page 7-14

About Automatic PID Tuning

Use the Simulink Control Design PID Tuner to automatically tune PID gains in a single-loop control system containing a PID Controller block. You specify design requirements, such as response time and phase margin, and the Tuner automatically computes the corresponding gains.

With the tuner, you can achieve a good balance between performance and robustness for both one- and two-degree-of-freedom PID controllers, as described in the following sections:

- “Tuning One-Degree-of-Freedom PID Controllers” on page 7-4
- “Tuning Two Degree-of-Freedom PID Controllers” on page 7-14

Tuning One-Degree-of-Freedom PID Controllers

Use the PID Tuner to tune one-degree-of-freedom PID Controller blocks. With a one-degree-of-freedom PID controller, you can achieve either good set-point tracking or good disturbance rejection when a disturbance model exists. To achieve both good set point tracking and good disturbance rejection, use a two-degree-of-freedom PID Controller 2DOF block.

When you open the tuner, it automatically linearizes the plant and designs an initial controller, as described in “Opening the Tuner and Analyzing the Initial Design” on page 7-5.

You analyze the design to determine if it meets your requirements, and if not you can refine the design using interactive tuning. For instructions, see “Adjusting Response Time To Tune Parameters” on page 7-8 and “Adjusting Bandwidth and Phase Margin To Tune Parameters” on page 7-9.

For an example of tuning a one-degree-of-freedom PID Compensator, see the Automated Tuning of Simulink PID Controller Block demo.

Opening the Tuner and Analyzing the Initial Design


Before you can tune a PID compensator, you must have already:

- Created a Simulink model containing a PID Controller block in a single-loop control system.
The plant in your model can have any order and any time delays.
- Configured the PID block settings, such as type, form, time domain, and sample time.

To open the PID Tuner and view the initial compensator design:

- 1** Open the Simulink model by typing the model name at the MATLAB command prompt.
- 2** Double-click the PID Controller block to open the block dialog box.
- 3** In the block dialog box, click **Tune**.

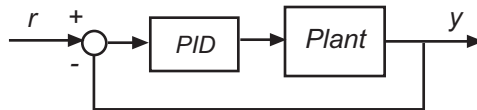
This action

- Automatically linearizes the plant in your model as seen by the controller. This linearization includes everything in the loop except the PID Controller block itself.
 - This linearization occurs at the operating point specified in the model. To linearize at a different operating point, click the icon  in the PID tuner.
 - This linearization ignores nonlinear settings in the PID block, such as antiwindup, output saturation, and external reset.

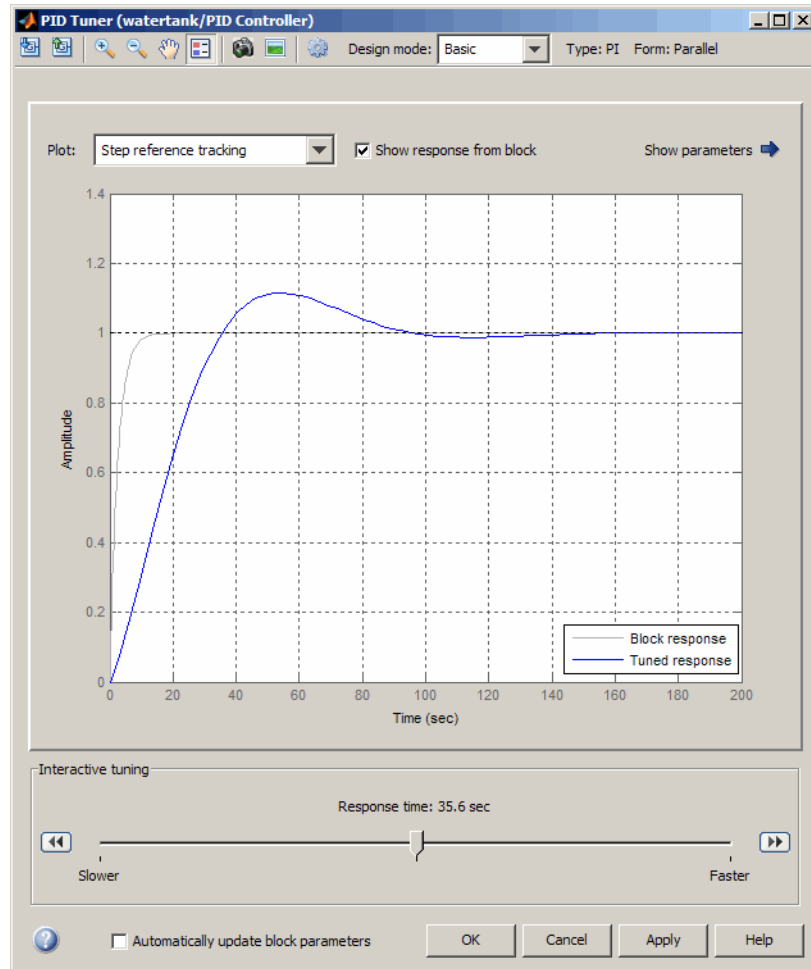
Note If the plant and controller have different time domains, the plant discretizes or converts to continuous time to match the controller. If the plant and controller are both discrete time with different sample times, the plant resamples to match the controller. All conversions use the tustin method.

- Designs an initial controller that achieves a reasonable tradeoff between performance and robustness using a frequency-based design algorithm

This design is for the following loop structure, regardless of the structure in the Simulink model.



- Opens the PID Tuner with the initial compensator design



Tip After the tuner launches, you can close the PID Controller block dialog box.

- 4 Analyze the initial compensator design to determine if it meets your design requirements, as described in “Analyzing Your Design in the PID Tuner” on page 7-11.

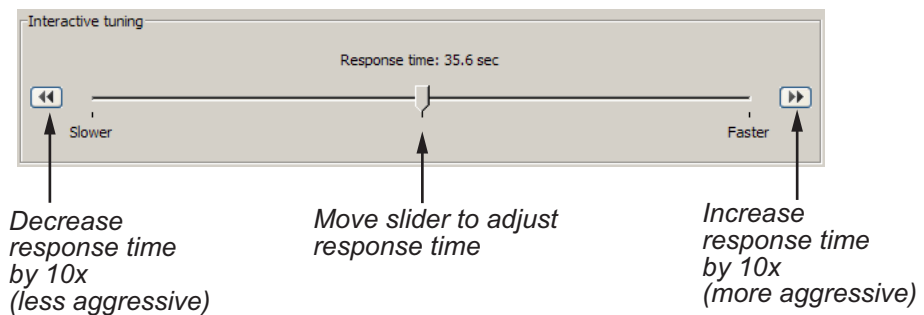
- If the design meets your requirements, go to step 5.
 - If the design does *not* meet your requirements, interactively tune parameters to achieve a good balance between performance and robustness by:
 - “Adjusting Response Time To Tune Parameters” on page 7-8
 - “Adjusting Bandwidth and Phase Margin To Tune Parameters” on page 7-9
- 5** If the initial compensator design meets your requirements, verify that it behaves in a similar way in the nonlinear Simulink model. For instructions, see “Verifying the PID Compensator Design on the Nonlinear Simulink Model” on page 7-13.



Adjusting Response Time To Tune Parameters

Before you can tune parameters, you must have already opened the tuner and determined that the initial design does not meet your requirements. For instructions, see “Opening the Tuner and Analyzing the Initial Design” on page 7-5.

To adjust response time to tune the controller gains:

- 1** In the PID Tuner, select **Basic** for **Design mode**.
- 2** Move the Response time slider to find a PID controller that provides a slower or faster response for your system.



Tip To store a design and continue tuning without losing the design, click the camera icon . To retrieve this design, click the picture icon .

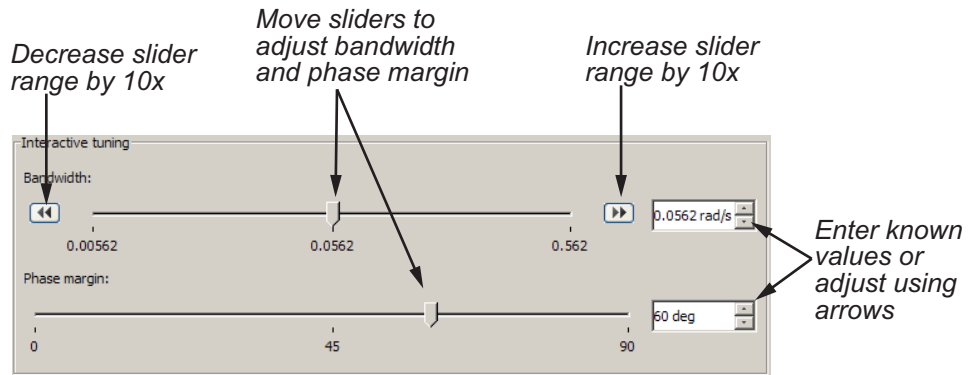
- 3** Analyze the compensator design to determine if it meets your design requirements, as described in “Analyzing Your Design in the PID Tuner” on page 7-11.
 - If the design meets your requirements, go to step 5.
 - If the design does *not* meet your requirements, try adjusting the bandwidth and phase margin. For instructions, see “Adjusting Bandwidth and Phase Margin To Tune Parameters” on page 7-9.
- 4** If the initial compensator design meets your requirements, verify that it behaves in a similar way in the nonlinear Simulink model. For instructions, see “Verifying the PID Compensator Design on the Nonlinear Simulink Model” on page 7-13.

Adjusting Bandwidth and Phase Margin To Tune Parameters

Before you can tune parameters, you must have already opened the tuner and determined that the initial design does not meet your requirements. For instructions, see “Opening the Tuner and Analyzing the Initial Design” on page 7-5. You might have also adjusted the response time to tune parameters, as described in “Adjusting Response Time To Tune Parameters” on page 7-8.



To adjust bandwidth and phase margin to tune the P, I, D, and N parameters:

- 1** In the PID Tuner, select Extended for **Design mode**.
- 2** Adjust the bandwidth and phase margin to find PID controller with a good balance between performance and robustness, respectively. You can do so, in the following ways:
 - Moving the slider bar
 - Entering a known value in the text field
 - Incrementally adjusting the value in the text field using the up and down arrows



Decreasing the bandwidth increases the response time, which makes the controller less aggressive. Increasing the bandwidth decreases the response time, which makes the controller more aggressive. For continuous time, bandwidth is a finite positive real number. For discrete time, bandwidth is a positive real number less than π/T_s , where T_s is the controller's sample time.

Decreasing the phase margin decreases robustness. Increasing the phase margin increases robustness. Phase margin must be an integer between 0 and 90.

Tip To store a design and continue tuning without losing the design, click the camera icon . To retrieve this design, click the picture icon .

- 3 Analyze the compensator design to determine if it meets your design requirements, as described in “Analyzing Your Design in the PID Tuner” on page 7-11.
 - If the design meets your requirements, go to step 4.
 - If you cannot find a compensator to meet your requirements by adjusting bandwidth and phase margin, see “If You Cannot Find a Good Design Using the PID Tuner” on page 7-12.
- 4 If the initial compensator design meets your requirements, verify that it behaves in a similar way in the nonlinear Simulink model. For instructions,

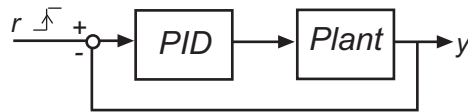
see “Verifying the PID Compensator Design on the Nonlinear Simulink Model” on page 7-13.

Analyzing Your Design in the PID Tuner

To determine if your compensator design meets your requirements, you can analyze the system response in any of these response plots:

- Step reference tracking (default) — For evaluating how the controller tracks a reference signal.

This plot displays the closed loop $\frac{GC}{1+GC}$, where G is the plant and C is the PID controller.

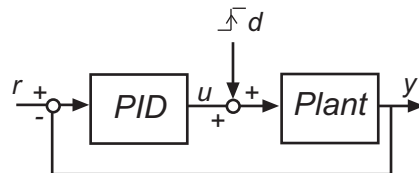


- Step disturbance rejection — For evaluating how the controller rejects a load disturbance.

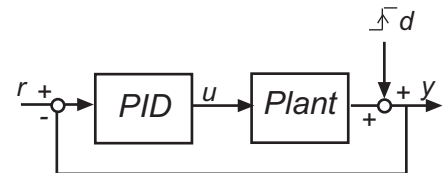
You can set the Location of step disturbance in the PID Tuner Preferences dialog box at the plant input or output. This plot displays the closed loop

as $\frac{G}{1+GC}$ when the disturbance at the plant input and $\frac{1}{1+GC}$ when the disturbance at the plant output.

Disturbance at plant input (default)




Disturbance at plant output

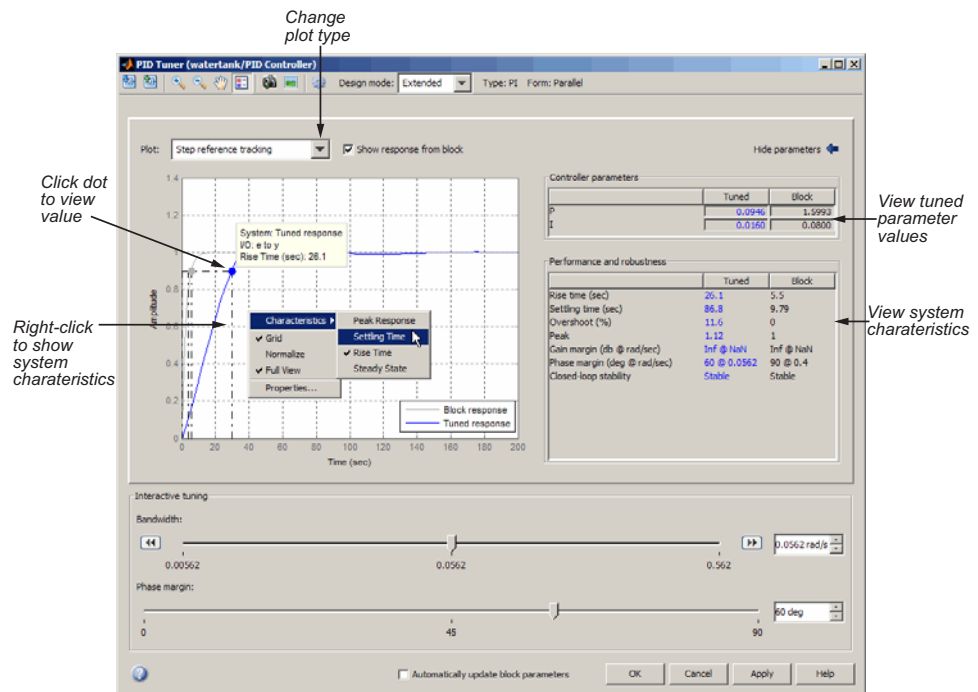



- Open-loop bode plot — For evaluating open-loop frequency response.
- Open-loop nichols chart — For evaluating open-loop frequency response.

To view a particular plot type, select the plot type from the **Plot** drop-down menu.

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

- Directly on the response plot — Use the right-click menu to add characteristics, which appear as blue dots. Then, left-click the dots to open the corresponding data marker.
- In the **Performance and robustness** table — To display this table, click the **Show Parameters** arrow .



To perform further analysis on the plant model, click the icon  to export the plant to the MATLAB workspace as an LTI object.

If You Cannot Find a Good Design Using the PID Tuner. If you cannot find a good design with your current controller type, switch to a different controller type in the PID block dialog box. Then, click **Tune** to design the new controller. For example, if a PI controller cannot stabilize your plant, try a PID controller.

If you cannot find any type of PID controller to meet your requirements, try a different controller structure and tune it using the SISO Design Tool. For information on tuning blocks using the SISO Design Tool, see “Design and Analysis of Control Systems” on page 7-16.

Verifying the PID Compensator Design on the Nonlinear Simulink Model

In the PID Tuner, you tune the compensator using a linear model of your plant. After you find a good compensator design in the PID Tuner, verify that the design behaves as expected in your nonlinear Simulink model.

To verify the compensator design in the nonlinear Simulink model:

- 1** In the PID Tuner, click **Apply** to update the Simulink PID Controller block with the tuned PID parameters.


Tip To automatically update PID block parameters every time you tune the controller in the PID Tuner, select **Automatically update block parameters**.

- 2** Simulate the Simulink model and evaluate whether the simulation output matches the time-domain response in the PID Tuner.

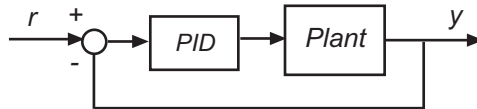
If the Simulation Output Does Not Match the PID Tuner Response.

If your simulation output does not match the PID Tuner response, check whether the Simulink model is at the operating point used in the tuner. If you Simulink model and the PID Tuner use different operating points, you can either:

- Use the PID tuner operating point by initializing the Simulink model with this operating point and evaluating again
- Use the Simulink model operating point by linearizing the model at this operating point and retuning in the PID Tuner

To relinearize, click the icon . Then, in the dialog box, select **Linearize at the operating point currently specified in the Simulink model**.

If the model and PID Tuner use the same operating point, your model may have strong nonlinearities in the plant. Your loop structure may also be different than the loop structure assumed by the PID Tuner.



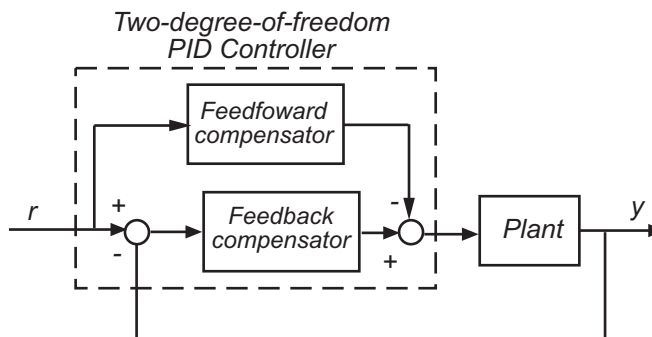
You can either:

- Use the advanced PID settings, such as antiwindup to account for the nonlinearities, and retune using the PID Tuner.
- Add other compensators to your Simulink model, such as a feedforward compensator, and design all compensators simultaneously using the SISO Design Tool.

For information on tuning blocks using the SISO Design Tool, see “Design and Analysis of Control Systems” on page 7-16.

Tuning Two Degree-of-Freedom PID Controllers

Use the PID Tuner to tune two-degree-of-freedom PID Controller 2DOF blocks to achieve both good set point tracking and good disturbance rejection. This compensator, commonly called ISA-PID compensator, contains a feedforward compensator, which is a PD, and a feedback compensator, which is a PID.



To tune a two-degree-of-freedom compensator:

- 1** Tune the P, I, D, and N parameters of the feedback portion of the compensator to meet disturbance rejection requirements using the PID tuner.

You tune this portion of the compensator in the same way that you tune a one-degree-of-freedom PID compensator, as described in “Tuning One-Degree-of-Freedom PID Controllers” on page 7-4.

Ensure that you update the PID block with your design.

- 2** Adjust the set-point weights b and c of the feedforward portion of the compensator to meet the set-point tracking requirements as follows:
 - a** In the PID Controller block dialog box, enter values for the set-point weights **b** and **c** between 0 and 1.

Smaller values generally produce slower reference tracking performance. These settings do not affect loop stability or disturbance rejection.

- b** Evaluate whether the compensator design meets the design requirements by viewing a simulation of the Simulink model.

For an example of tuning a two-degree-of-freedom PID Compensator, see the Design a Simulink PID Controller (2DOF) Block for a Reactor demo.

Design and Analysis of Control Systems

In this section...
“Compensator Design Process Overview” on page 7-16
“Beginning a Compensator Design Task” on page 7-16
“Selecting Blocks to Tune” on page 7-18
“Selecting Closed-Loop Responses to Design” on page 7-21
“Selecting an Operating Point” on page 7-23
“Creating a SISO Design Task” on page 7-26
“Completing the Design” on page 7-37

Compensator Design Process Overview

Compensator design in the Control and Estimation Tools Manager involves the following steps:

- 1 “Selecting Blocks to Tune” on page 7-18
- 2 “Selecting Closed-Loop Responses to Design” on page 7-21
- 3 “Selecting an Operating Point” on page 7-23
- 4 “Creating a SISO Design Task” on page 7-26
- 5 “Completing the Design” on page 7-37

Beginning a Compensator Design Task

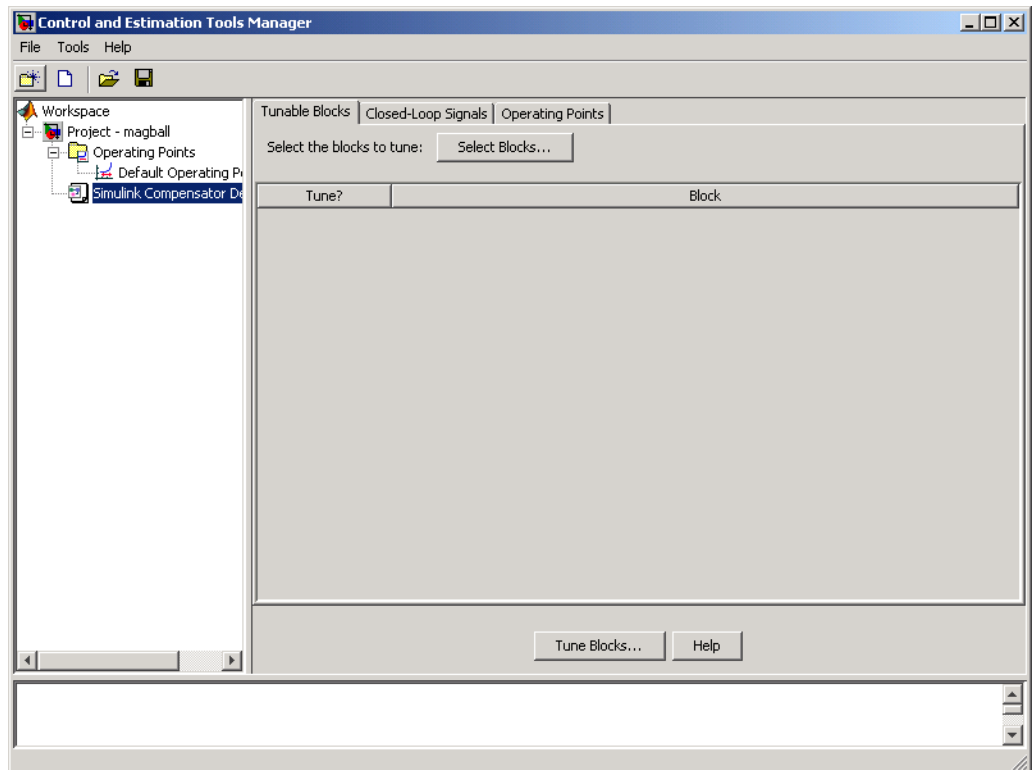
This chapter continues the magball example from “Example Model: The Magnetic Ball System” on page 1-2.

Before you begin this compensator design example, close the Control and Estimation Tools Manager and the magball model, if you have them open, to make sure you are working with a fresh version of the magball model. You do not need to save any projects or any changes to the model.

To begin a new compensator design task for the magball model:

- 1 Enter `magball` at the MATLAB command line to open the magball model.
- 2 Select **Tools > Control Design > Compensator Design** from the magball window.

The Control and Estimation Tools Manager opens and creates a new compensator design task, as shown in the following figure.



The project tree in the left pane of the Control and Estimation Tools Manager now shows a **Simulink Compensator Design Task** node as part of **Project - magball** in addition to the **Operating Points** node. You can select a node within the tree to display its contents in the right pane.

- For information on the **Tunable Blocks** pane within the **Simulink Compensator Design Task** node, refer to “Selecting Blocks to Tune” on page 7-18.
- For information on the **Closed-Loop Signals** pane within the **Simulink Compensator Design Task** node, refer to “Selecting Closed-Loop Responses to Design” on page 7-21.
- For information on the **Operating Points** node or the **Operating Points** pane within the **Simulink Compensator Design Task** node, refer to “Selecting an Operating Point” on page 7-23.

Selecting Blocks to Tune

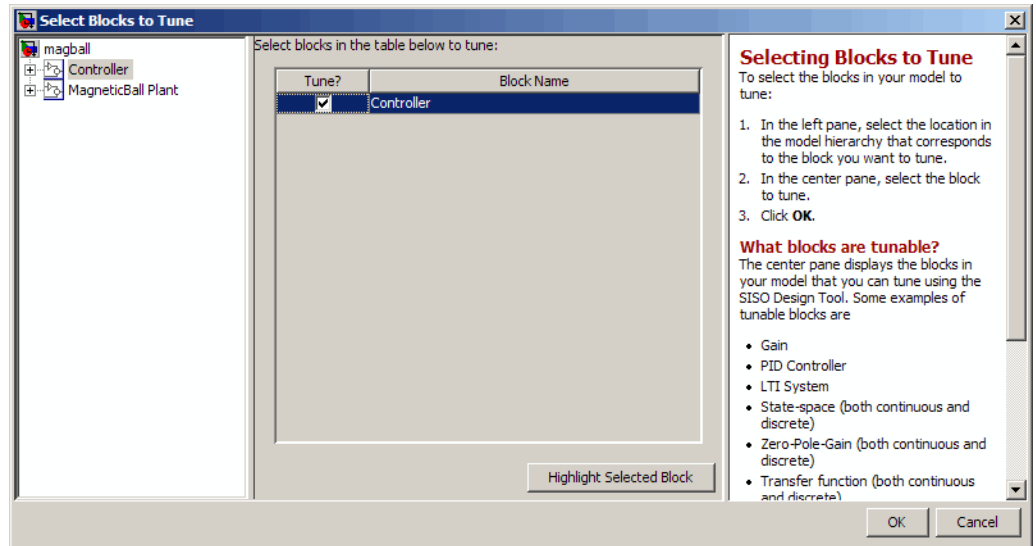
How to Select Blocks to Tune

This section continues the magball example from “Beginning a Compensator Design Task” on page 7-16. At this stage in the example, you have already created a compensator design task.

In this step of the compensator design, you select the blocks in your model to tune from a list of tunable blocks in your model. *Tunable blocks* are blocks that you can tune using the SISO Design Tool to achieve the desired response of your system. Typically, these blocks serve as the compensators in your model.

In this example, you tune the compensator block called Controller inside the Controller subsystem of the magball model. To select this block as the block to tune:

- 1** Select the **Simulink Compensator Design Task** node.
- 2** In the **Tunable Blocks** pane, click **Select Blocks**. The Select Blocks to Tune dialog box opens.
- 3** Select the Controller subsystem in the left pane to display that subsystem’s tunable blocks within the center pane. Within the center pane, select the check box next to the Controller block’s name.



4 Click **OK** to apply your selections, and close the dialog box.

What Blocks Are Tunable?

You can tune parameters in the blocks shown in the following table using Simulink Control Design software. The block input and output signals for tunable blocks must have scalar, double-precision values.

Tunable Blocks	Simulink Library
Gain	Math Operations
LTI System	Control System Toolbox
Discrete Filter	Discrete
PID Controller (one-degree-of-freedom only)	<ul style="list-style-type: none"> • Continuous • Discrete • Simulink Extras Additional Linear

Tunable Blocks	Simulink Library
State-space blocks	<ul style="list-style-type: none"> • Continuous • Discrete • Simulink Extras Additional Linear
Zero-pole blocks	<ul style="list-style-type: none"> • Continuous • Discrete • Simulink Extras Additional Linear
Transfer function blocks	<ul style="list-style-type: none"> • Continuous • Discrete • Simulink Extras Additional Linear

You can also tune the following versions of the blocks listed in the table:

- Blocks with custom configuration functions associated with a masked subsystem
- Blocks discretized using the Simulink Model Discretizer

Note If your model contains Model blocks with normal-mode model references to other models, you can select tunable blocks in the referenced models for compensator design.

Creating Custom Configuration Functions

When you have masked subsystems that you want to tune in your model, they will not automatically appear in the list of tunable blocks. For them to appear in the list, you need to create a custom configuration function for the masked subsystem. The custom configuration function serves the following functions:

- It informs the Simulink Control Design software that you want this block to be available for tuning.

- It determines how you want the SISO Design Task to treat the block; it describes the relationship between the block mask parameters and the poles and zeros of the transfer function.

To learn how to create a custom configuration function, see the Simulink Control Design demo “Tuning Custom Masked Subsystems”.

Selecting Closed-Loop Responses to Design

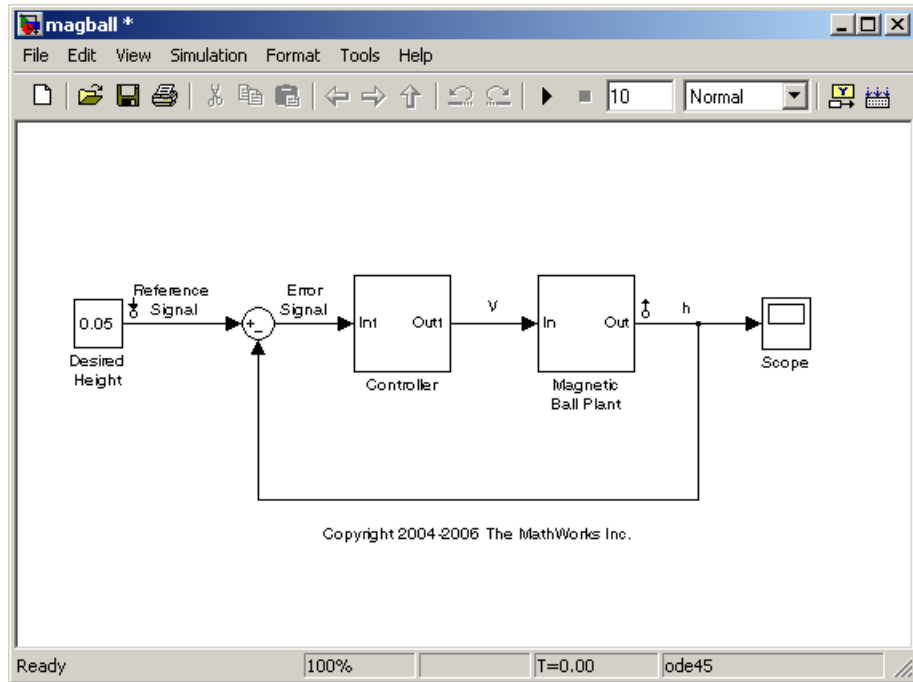
This section continues the magball example from “Selecting Blocks to Tune” on page 7-18. At this stage in the example a compensator design task has been created, and tunable blocks have been selected.

In this step of the compensator design task, you will select the closed loops whose responses you want to design in your model. A closed-loop system is defined by an input point, such as a reference or disturbance signal, and an output point, such as a measured output or actuator signal. The **Simulink Compensator Design Task** uses linearization points on the signal lines of the model to determine the closed-loop systems. For more information on linearization points, see “Selecting Inputs and Outputs for the Linearized Model” on page 4-45.

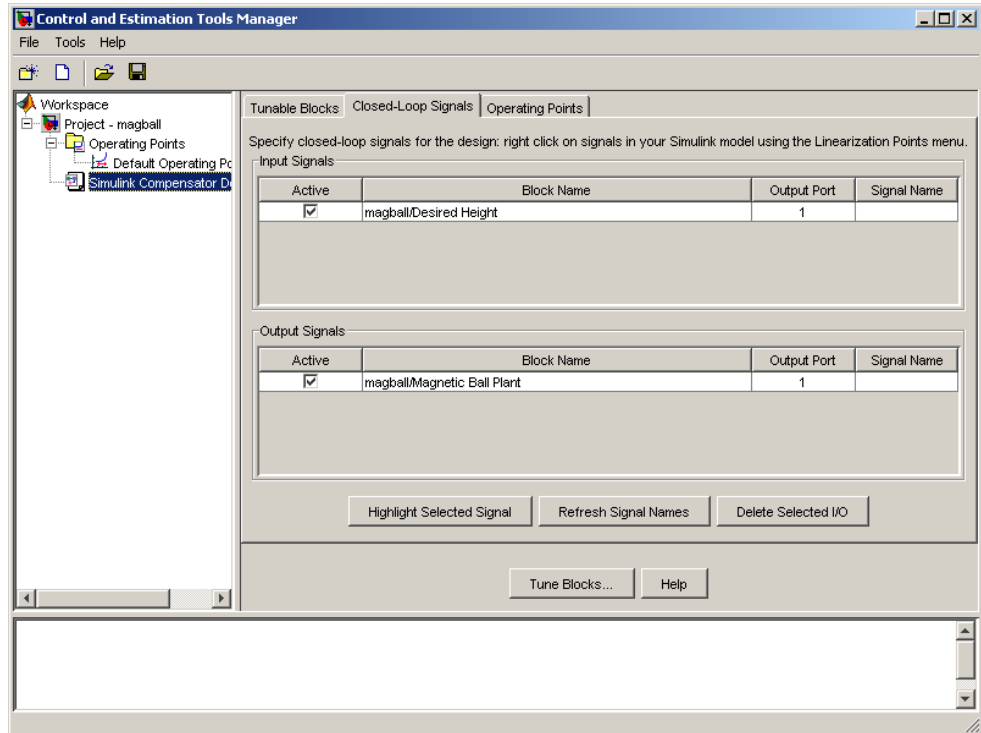
In this example you will design the response of the closed-loop system from the reference signal to the output of the plant model. To set up linearization points to define this closed-loop system, perform the following steps:

- 1** On the magball model diagram, position the mouse on the Reference signal between the Desired Height block and the Sum block. Right-click and select **Linearization Points > Input Point** from the menu to add an input point.
- 2** Position the mouse on the signal line at the output of the Magnetic Ball Plant block. Right-click and select **Linearization Points > Output Point** from the menu to add an output point.

The magball model should now appear as follows:



Within the Control and Estimation Tools Manager, click the **Closed-Loop Signals** tab of the **Simulink Compensator Design Task** node to view the input and output points in the model.



Within this pane you can view the input and output signals in the model and use the **Active** column to select the ones you want to use to define closed-loop systems for compensator design.

Selecting an Operating Point

This section continues the magball example from “Selecting Closed-Loop Responses to Design” on page 7-21. At this stage in the example, a compensator design task has been created, tunable blocks have been selected, and closed-loop signals have been selected.

In this step of the compensator design task, you will select the operating point that you want to use in the compensator design. The Simulink Control Design software uses the operating point when it linearizes the model before creating a SISO Design Task.

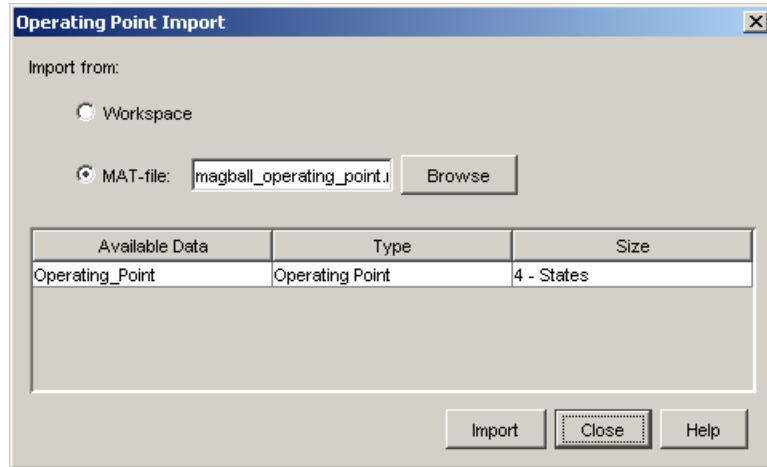
Note A compensator designed for the linearized model is likely to control the behavior of the nonlinear model only in a small region around the operating point that the model was linearized at. Therefore it is important that the linearization of the model is accurate and the selection of the operating point about which the system is linearized is an important step in the compensator design process.

In Chapter 2, “Operating Point Analysis Using the GUI”, you created operating points for this model, exported one to the MATLAB workspace, and saved it in a MAT-file. This example imports the operating point that you saved. If you did not already compute and save an operating point for the magball model, you can import an operating point that was installed along with the Simulink Control Design demos.

To import an operating point for compensator design, perform the following steps:

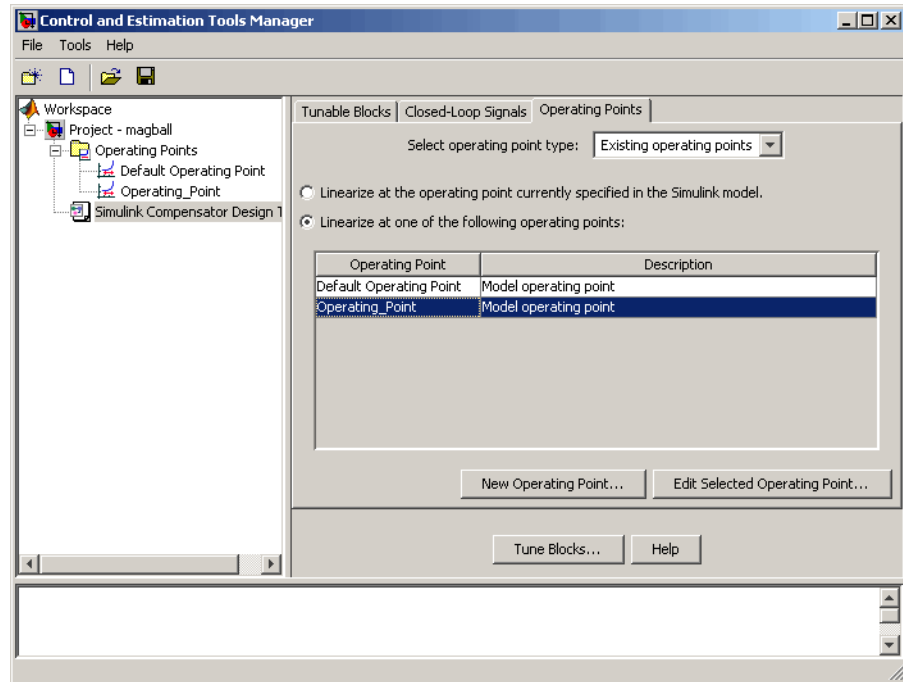
- 1** Select the **Operating Points** node in the Control and Estimation Tools Manager.
- 2** Click the **Import** button, in the bottom-right corner of the Control and Estimation Tools Manager.
- 3** In the Operating Point Import dialog box, select **MAT-file** as the location to import from.
- 4** Click **Browse** and locate the file `magball_operating_point.mat` that you previously saved. If you did not previously save an operating point, browse to `matlabroot/toolbox/slcontrol/slctrldemos/magball_operating_point.mat`.
- 5** Click **Open** to return to the Operating Point Import dialog box.

The Operating Point Import dialog box now shows all the operating points available within the selected MAT-file. In this case just a single operating point is contained in the MAT-file.



- 6 Select this operating point and click **Import** to import it into the Control and Estimation Tools Manager.

Click the **Operating Points** tab in the **Simulink Compensator Design Task** node to select an operating point for the compensator design. For this example, you should use the operating point that you just imported, called **Operating_Point**. To specify this operating point, first select the **Linearize at one of the following operating points** option button. Then select **Operating_Point** in the list, as shown in the following figure.



For information on performing compensator design at specified simulation times and events, see “Linearizing at an Operating Point” on page 4-57.

Creating a SISO Design Task

- “What is a SISO Design Task?” on page 7-26
- “Configuring Design Plots” on page 7-27
- “Configuring Analysis Plots” on page 7-30
- “Control Design Linearization Options” on page 7-35
- “Designing Compensators for Plants with Time Delays” on page 7-36

What is a SISO Design Task?

This section continues the magball example from “Selecting an Operating Point” on page 7-23. At this stage in the example, a compensator design task

has been created, and tunable blocks, closed-loop signals, and an operating point have been selected.

In this step of the compensator design task, you will create and configure a **SISO Design Task** in the Control and Estimation Tools Manager. The **SISO Design Task** includes several tools for tuning the response of SISO systems:

- A graphical editing environment in the SISO Design Tool window that contains design plots such as root-locus, and Bode diagrams
- An LTI Viewer window where you can view time and frequency analysis plots of the system
- A compensator editor where you can directly edit the block mask parameters or the poles and zeros of compensators in your system
- A tool that automatically generates compensators using PID, internal model control (IMC), or linear-quadratic-Gaussian (LQG) methods (uses the Control System Toolbox software)
- Optimization-based tuning methods that automatically tunes the system to satisfy design requirements (available when you have the Simulink Design Optimization product)

The Design Configuration Wizard guides you through the selection of the open- and closed-loop systems you want to design and the configuration of the design and analysis plots you want to use in the **SISO Design Task**. To launch the wizard, click the **Tune Blocks** button in the Simulink Compensator Design Task node. The wizard opens in a separate window.

The first page of the wizard provides an overview of the design configuration process and lists some issues to consider when selecting design and analysis plots. Click **Next** to continue to step 1 of the design configuration process on the second page of the wizard.

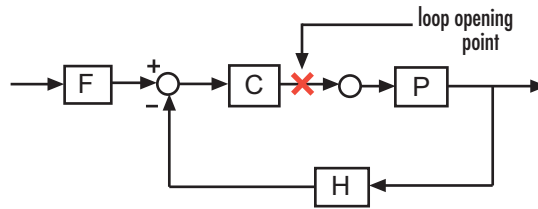
Configuring Design Plots

In step 1, select the open- and closed-loop systems that you want to design in your model, and up to six corresponding design plots you want to use.

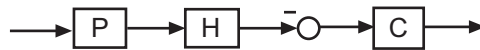
Open-loop design allows you to design the response of a closed feedback loop in your model by artificially opening the loop and designing the response of

this *open-loop* system. The open-loop design plots use rules of linear control theory to determine the dynamics of the closed-loop system from those of the open-loop system. Open-loop design is typically used to tune compensators that lie inside feedback loops.

A set of default open-loop systems is created for your model, shown in the lower half of the wizard. To create these open-loop systems, the software artificially opens the feedback loop at the output signal of each tunable block (at the X in the following figure) and unwraps the closed-loop system to create the corresponding open-loop system.



The unwrapped open-loop system, which is $-CPH$, is shown in the following figure. The open-loop design plots show the negative of the unwrapped open-loop, which is CPH . This configuration allows you to design controllers using a negative feedback architecture.



Note that elements that are outside the feedback loop, such as the prefilter **F**, are not seen in the open-loop system.

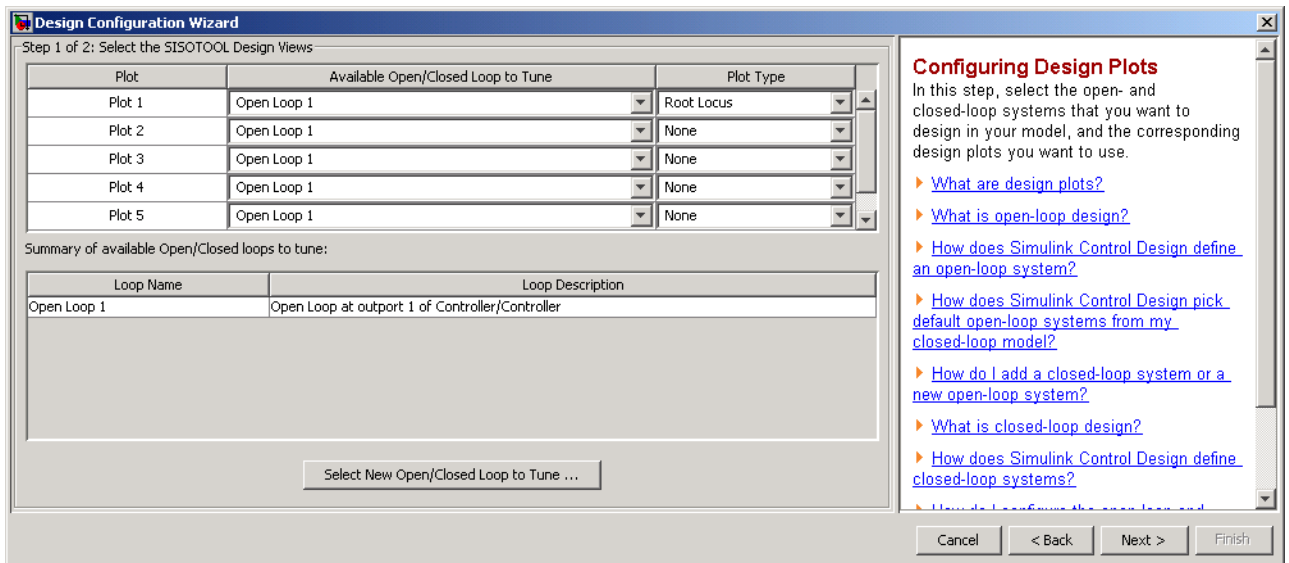
In this example, you will tune the response of **Open Loop 1** which is defined by a loop opening at the output of the Controller block. This open-loop system contains the plant model and the controller. To design this system, select **Open Loop 1** from the menu next to **Plot 1** in the wizard.

Next, select a design plot to use for this open-loop system. Design plots are interactive plots within the SISO Design Tool. You can use them to graphically tune parameters and manually move, add, or remove poles and zeros of the tunable blocks to tune and design the dynamics of open- and closed-loop systems in your model. The following table shows the design plots, along with their uses, available in the SISO Design Tool.

Type of Design Plot	Available Plots in the SISO Design Tool	Use to tune blocks that act as
Open-loop	Root Locus, Nichols, Open-loop bode	Feedback elements
Closed-loop	Closed-loop bode	Feedforward or prefilter elements

You can also use the design plots to specify requirements for stability, performance, or both to use in using optimization-based automated tuning.

For this example, select **Root Locus** from the menu next to **Plot 1** to use this plot type as the design plot for **Open Loop 1**. Step 1 of the wizard should now look similar to the following figure.



Click **Next** to proceed to step 2 of the wizard.

Configuring Analysis Plots

In this step, select the closed-loop responses that you want to view while designing your model, and the corresponding analysis plots you want to use to view them.

Analysis plots are plots that show the responses or dynamics of a closed or open loop systems or tunable blocks in your model. Although you cannot directly edit the analysis plots by graphically moving gains, poles, zeros, etc., changes that you make in the design plots, compensator editor, or automated design tools will affect the responses in the analysis plots. Possible analysis plots include

- Step response
- Impulse response
- Bode and Bode magnitude
- Nyquist
- Nichols
- Pole/Zero

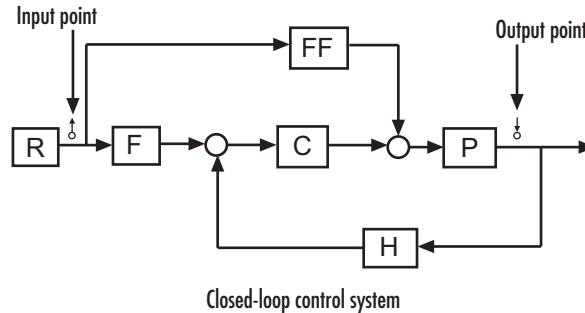
You can use analysis plots to

- Analyze closed-loop, open-loop, and tuned block responses in your control system.
- Define stability and performance requirements for optimization-based automated tuning.

For this example, select **Step** from the menu for **Plot 1** to create a step response analysis plot.

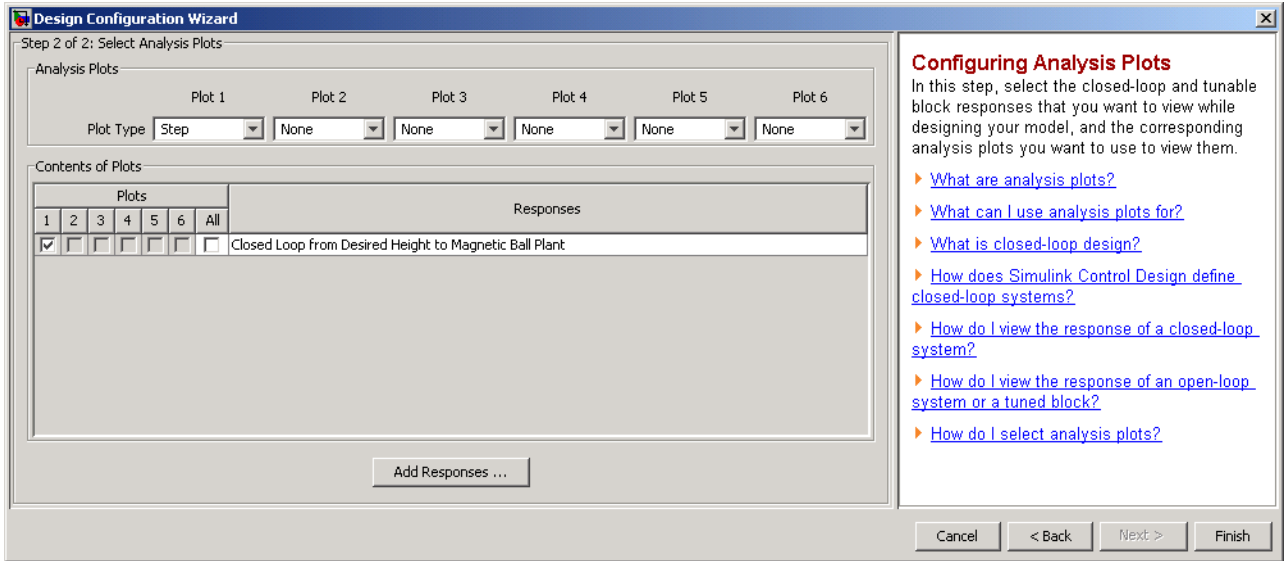
Next, select the closed-loop system that you want to display in this plot. A closed-loop system is a system that has not had any feedback loops opened for open-loop design. It typically defines the system whose response you want to control and it lies between the input and output signals of interest, for example between a reference signal and the plant output signal.

Linearization input and output points placed on signal lines in your model define closed-loop systems. The closed-loop system includes all blocks in the path between the input and the output.

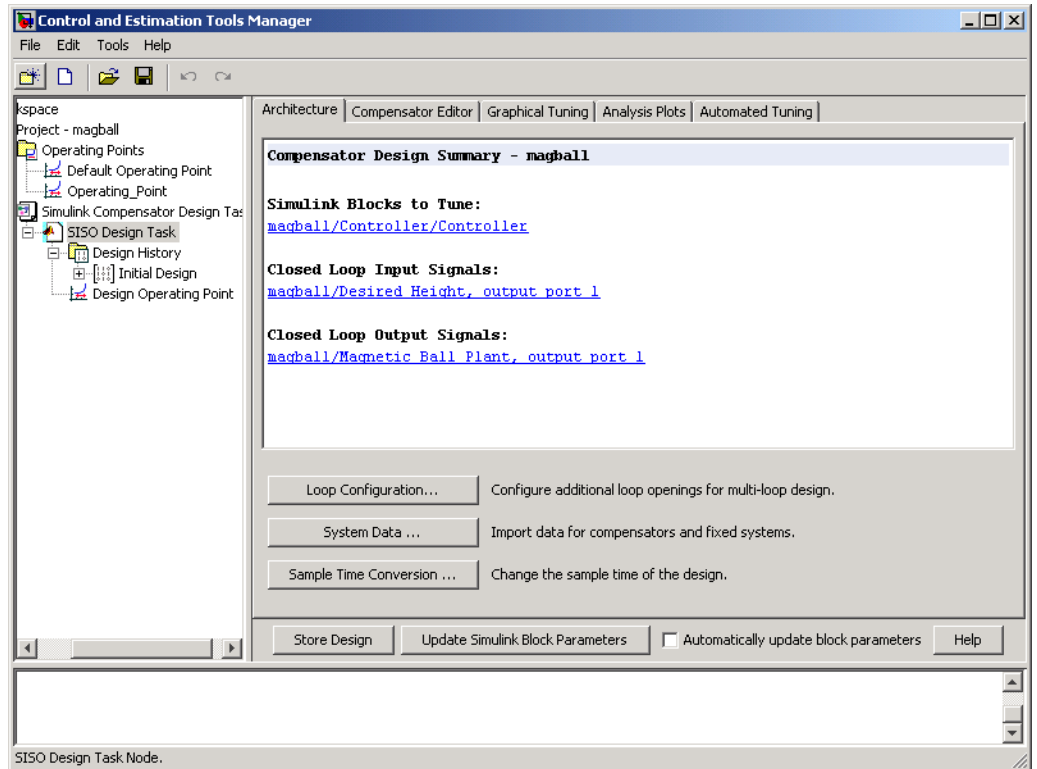


The software automatically displays a list of up to four closed-loop systems in your model, based on the input and output points on the signal lines. In this example, only one closed-loop system appears in the wizard, the closed-loop from the Desired Height signal to the output of the Magnetic Ball Plant Model, because the system only has one input and one output point. You can add additional closed-loop responses, as well as open-loop and tunable block responses. To add a new response, click the **Add Responses** button and complete the Select a New Response to Analyze dialog box.

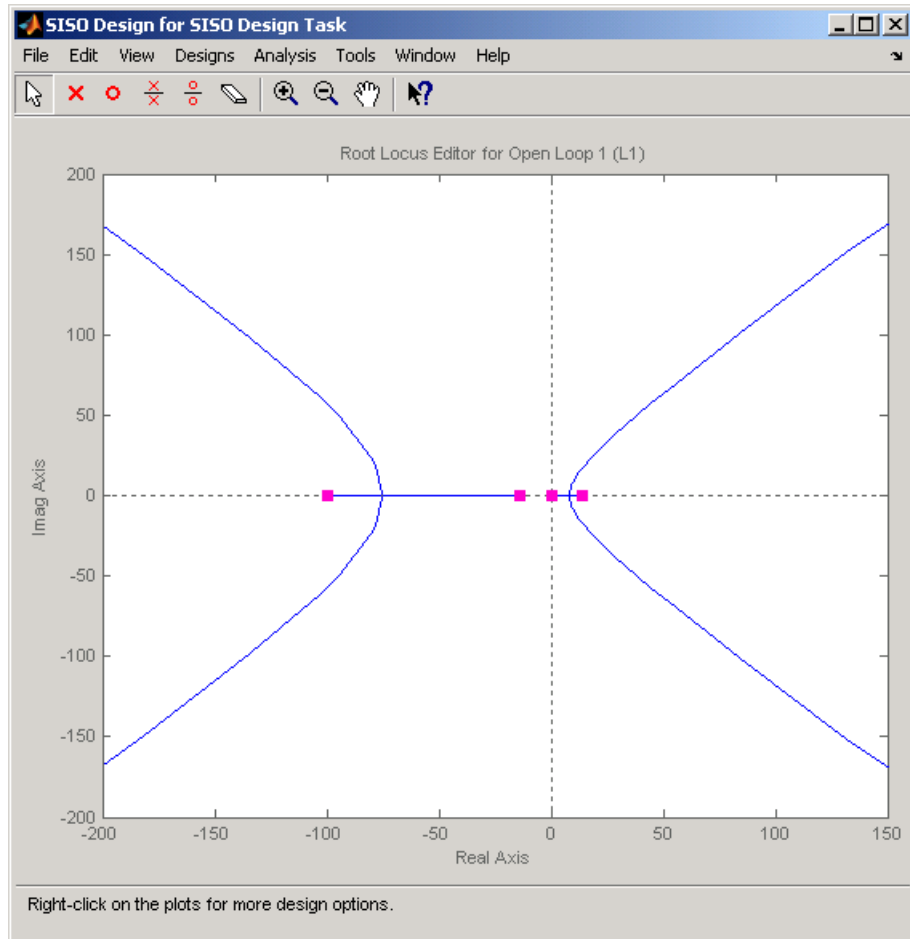
To display the current closed-loop system in the step response plot of **Plot 1**, select the check box under **Plot 1** to the left of the closed-loop system. Step 2 of the wizard should now look similar to the following figure.



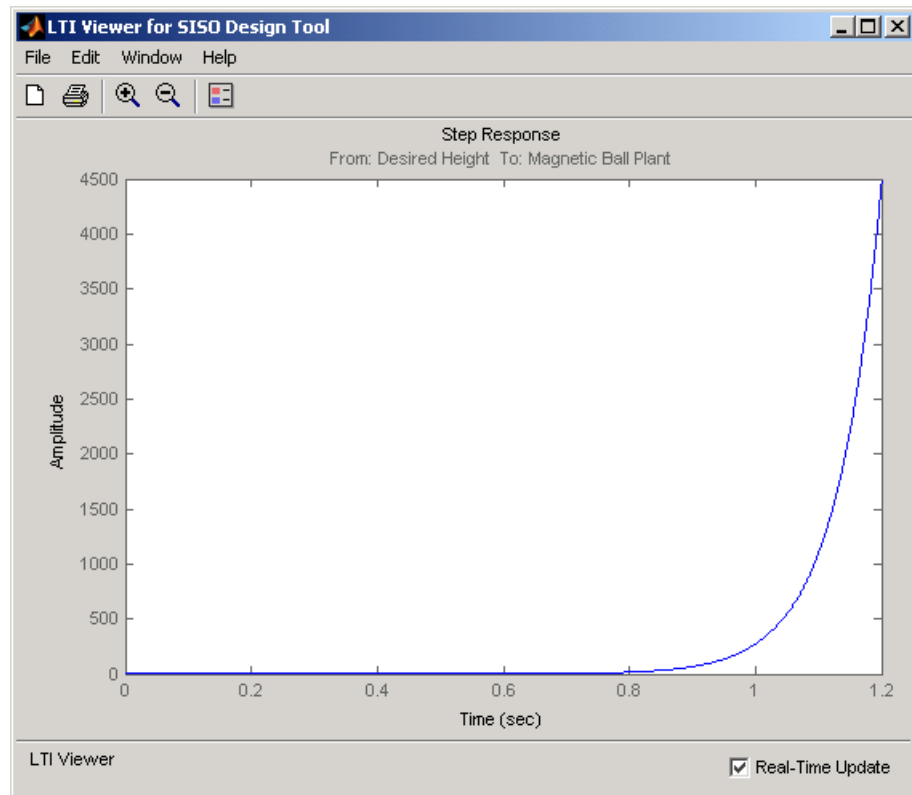
Click **Finish** to complete the wizard and create the **SISO Design Task** underneath the **Simulink Compensator Design Task** node within the Control and Estimation Tools Manager, as shown in the following figure.



The **SISO Design Task** also includes the design plots you configured in the Design Configuration Wizard. They appear within the SISO Design Tool window, as shown in the following figure.

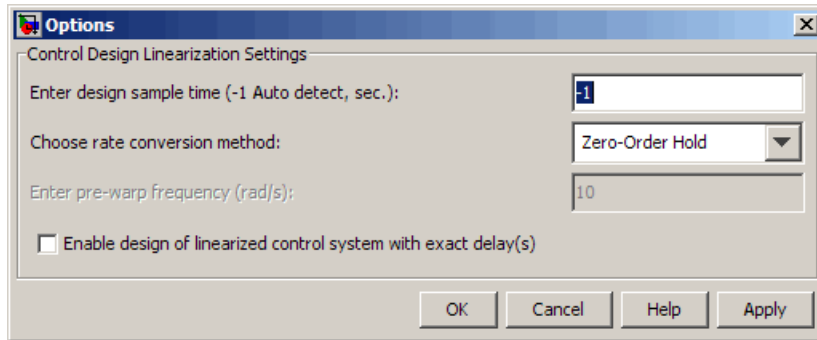


In addition, the **SISO Design Task** also includes the analysis plots you configured in the Design Configuration Wizard. They appear within the LTI Viewer window, as shown in the following figure.



Control Design Linearization Options

To modify or adjust the settings used to linearize a model when creating a SISO Design Task, click the **Simulink Compensator Design Task** node, and then select **Tools > Options**. The Options dialog box opens.



Specify the linearization sample time and rate conversion method. If, for the **Rate conversion method** parameter, you specify Tustin W/Prewarping, you must also specify a pre-warp frequency.

Designing Compensators for Plants with Time Delays

You can design compensators for plants with time delays using the tools in the SISO Design Task. These tools automatically create a linear model of your plant. Within this model, you can represent time delays in two ways—using Padé approximation or exact delay.

To represent time delays in the linear plant model using...	You must...
Padé approximation representations	Specify the Padé order in the Block Parameters window for each Simulink blocks with delays.
Exact delay representations	Open the Simulink Compensator Design Task, and select Tools > Options . Then, in the Options dialog box, select Enable design of linearized control systems with exact delay(s) .

Note Some tools do not support exact time delays and automatically compute a Padé approximation for delays. In this case, you receive a notification. The software uses the Padé order specified in SISO Tool Preferences and ignores the Padé order specified in your block. For more information, see “Time Delays Pane”.

For more information on the linearizing models with time delays, see “Linearizing Models with Time Delays” on page 4-15. For more information on the tools available for compensator design, see “Tools for Compensator Design” on page 7-37.

Completing the Design

- “Tools for Compensator Design” on page 7-37
- “Storing and Retrieving Designs” on page 7-42
- “Writing the Design to the Simulink Model” on page 7-44
- “Compare and Contrast the SISO Design Task and Enhanced SISO Design Task” on page 7-46
- “Design Operating Point Node” on page 7-49
- “SISO Tool Options” on page 7-49

Tools for Compensator Design

This section continues the magball example from “Creating a SISO Design Task” on page 7-26. At this stage in the example, a compensator design task has been created, tunable blocks, closed-loop signals, and an operating point have been selected, design and analysis plots have been created, and a **SISO Design Task** node has been created in the Control and Estimation Tools Manager.

In this step of the compensator design task, you will complete the design of the compensator in the magball model, using the **SISO Design Task** node. For a more detailed discussion of the **SISO Design Task** node, refer to the Control System Toolbox documentation.

The **SISO Design Task** node contains five panes with various tools for designing the compensators in your system.

- **Architecture:**
 - Configure loops for multi-loop design by opening signals to remove the effects of other feedback loops.
 - Import compensators into your system.
 - Convert the sample time of the system or switch between different sample times to design different compensators.
- **Compensator Editor:**
 - Directly edit the poles, zeros, and gains of the compensator.
 - Add or remove poles and zeros to the compensators.
- **Graphical Tuning:**
 - Configure design plots in the SISO Design Tool.
 - Use design plots to graphically manipulate the response of the system.
- **Analysis Plots:**
 - Configure analysis plots in the LTI Viewer.
 - Use analysis plots to view the response of open- or closed-loop systems.
- **Automated Tuning:** Design compensators using one of several automated methods.
 - Automatically generate compensators using PID, internal model control (IMC), or linear-quadratic-Gaussian (LQG) methods (uses Control System Toolbox software).
 - Use optimization-based methods that automatically tune the system to satisfy design requirements (available when you have the Simulink Design Optimization product).

You can use any of these design methods, or a combination of methods, to design the compensators for your system. A suitable final design for the Controller of the magball model is:

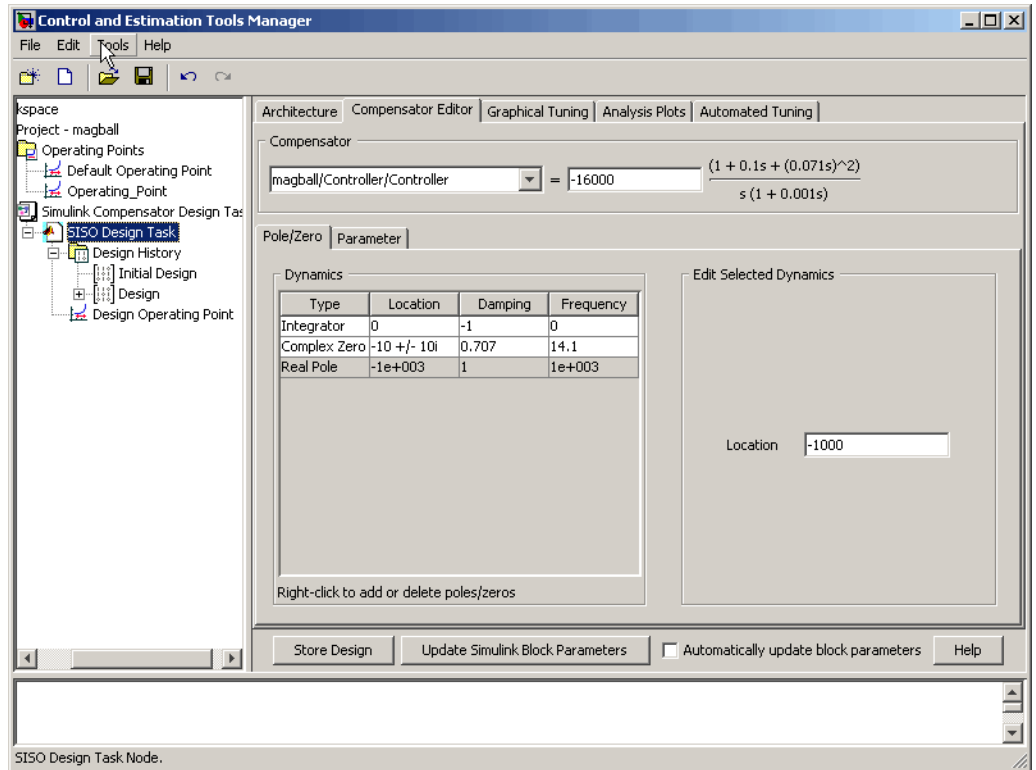
- Gain: -16000

- Integrator at the origin
- Complex zeros at $-10 \pm 10i$
- Real pole at -1000

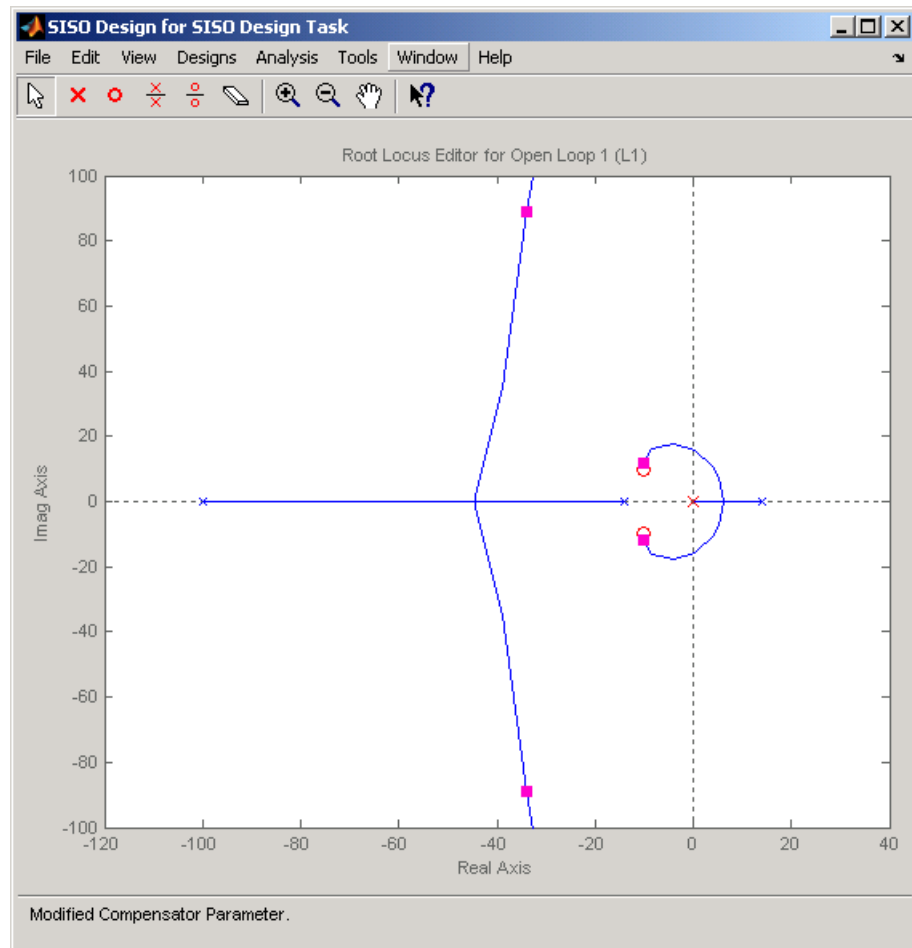
You can use the **Compensator Editor** in the **SISO Design Task** node to specify these settings. The initial design contains an integrator at the origin. Specify the remaining settings as follows:

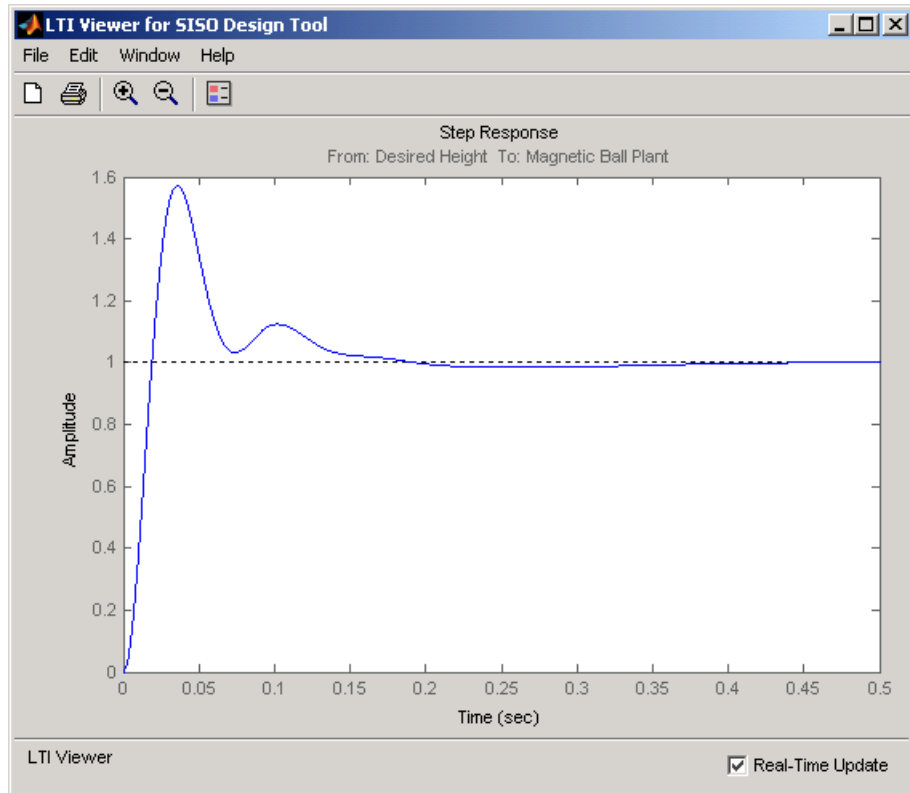
- Gain — Enter -16000 in the text box to the right of the equal sign in the **Compensator** area.
- Complex zeros — In the **Dynamics** table, right-click and then select **Add Pole/Zero > Complex Zero**. Select the new complex zero in the **Dynamics** table. In the **Edit Selected Dynamics** table:
 - Enter -10 in the **Real Part** field.
 - Enter 10 in the **Imaginary Part** field.
- Real pole — In the **Dynamics** table, right-click and then select **Add Pole/Zero > Real Pole**. Select the new real pole in the **Dynamics** table. In the **Edit Selected Dynamics** table:
 - Enter -1000 in the **Location** field.

The Control and Estimation Tools Manager should now appear as follows:



With these settings, the root-locus diagram and step-response plot should look similar to the following figure.





Storing and Retrieving Designs

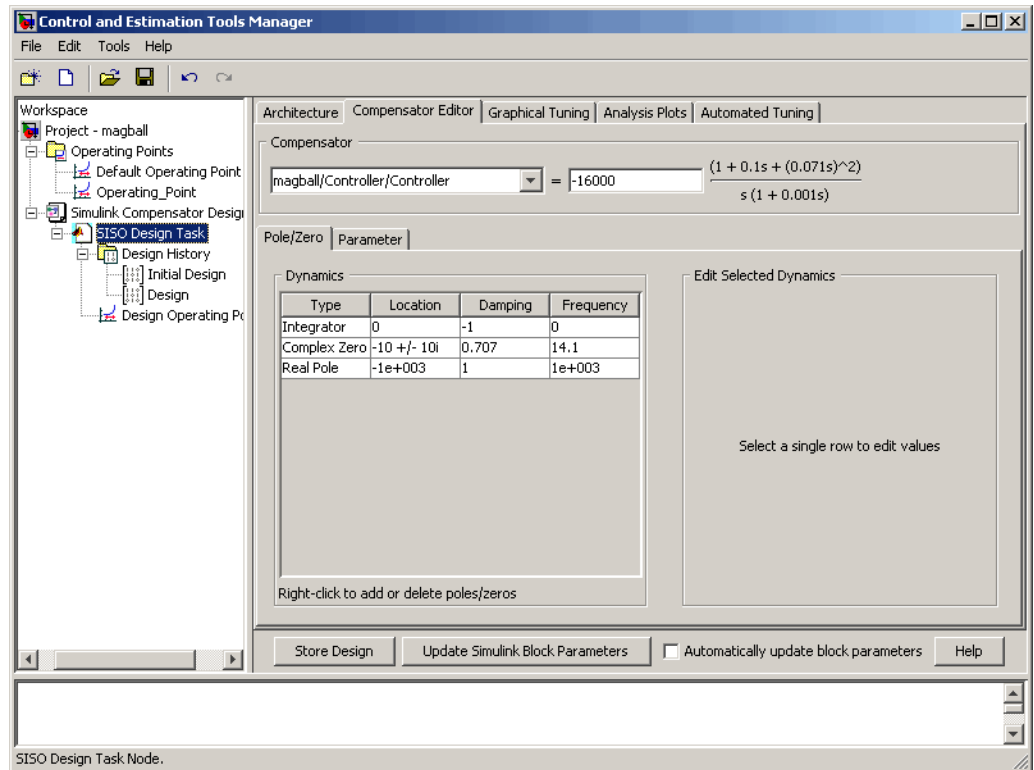
When you design a compensator within a **Simulink Compensator Design Task**, you can store the current design. You can retrieve the stored design at any time. This storage and retrieval capability lets you continue designing and still be able to return to a previously saved version of the design.

This section continues the example from “Completing the Design” on page 7-37. At this stage in the example, a compensator has been designed to control the system. To store the design within the **SISO Design Task** node, perform the following steps:

- 1 Select the **SISO Design Task** node in the Control and Estimation Tools Manager.

2 Underneath the **SISO Design Task** panes, click the **Store Design** button.

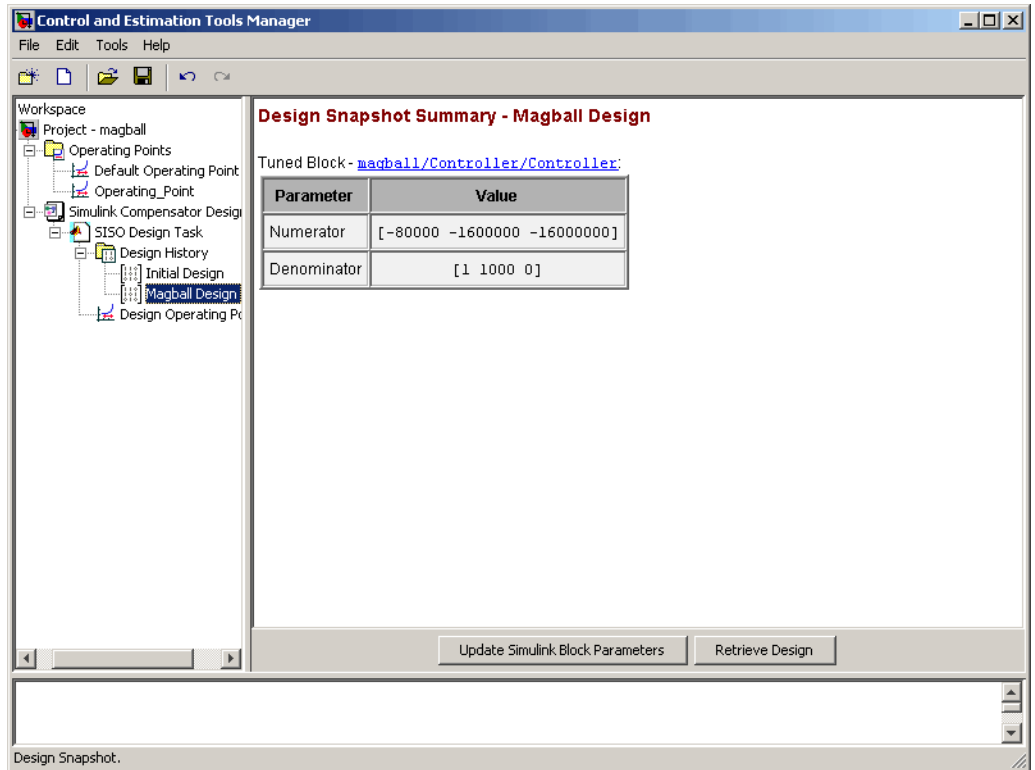
Clicking this button adds the current design to the **Design History** node as shown in the following figure. The default name for the design is **Design**.



To rename the design to something more descriptive:

- 1** Right-click the **Design** node underneath the **Design History** node.
- 2** Select **Rename** from the right-click menu.
- 3** Enter a name for your design. For this example, call the design **Magball Design**.

The Control and Estimation Tools Manager should now appear as follows:



Note After you store a compensator design, you can continue to refine it. If you make undesired changes, you can retrieve the stored design by selecting it in the **Design History** node and then clicking the **Retrieve Design** button.

Writing the Design to the Simulink Model

When designing a compensator within a **Simulink Compensator Design Task** node, you can write the compensator design to the Simulink model. This is useful when

- You want to see how the current design performs in the full nonlinear model.

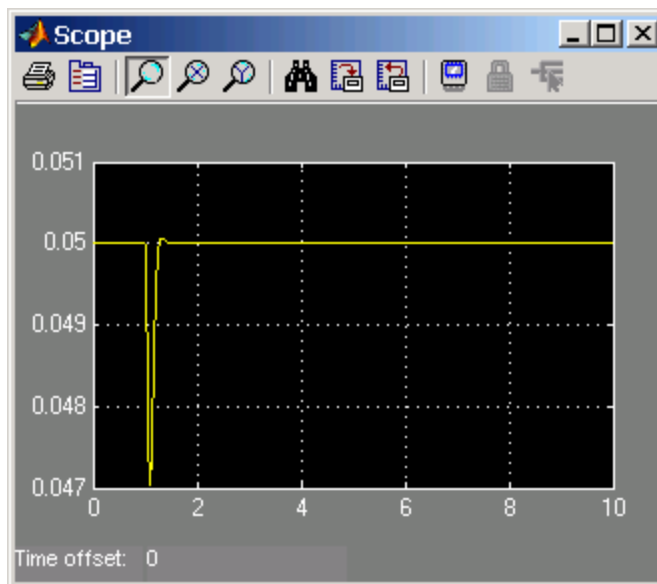
- You have completed the design and you want to update the model with the newly designed parameters.

This section continues the example from “Storing and Retrieving Designs” on page 7-42. At this stage in the example a compensator has been designed to control the system and the design has been stored within the **SISO Design Task** node. To write the stored design to the magball model, perform the following steps:

- 1** Select the **Magball Design** node under the **Design History** node in the Control and Estimation Tools Manager.
- 2** Click the **Update Simulink Block Parameters** button.

You can now simulate the magball model containing the newly designed Controller block. For more information on simulating models, see “Running Simulations” in the Simulink documentation.

After simulation, the Scope block of the magball model should look similar to the following figure.

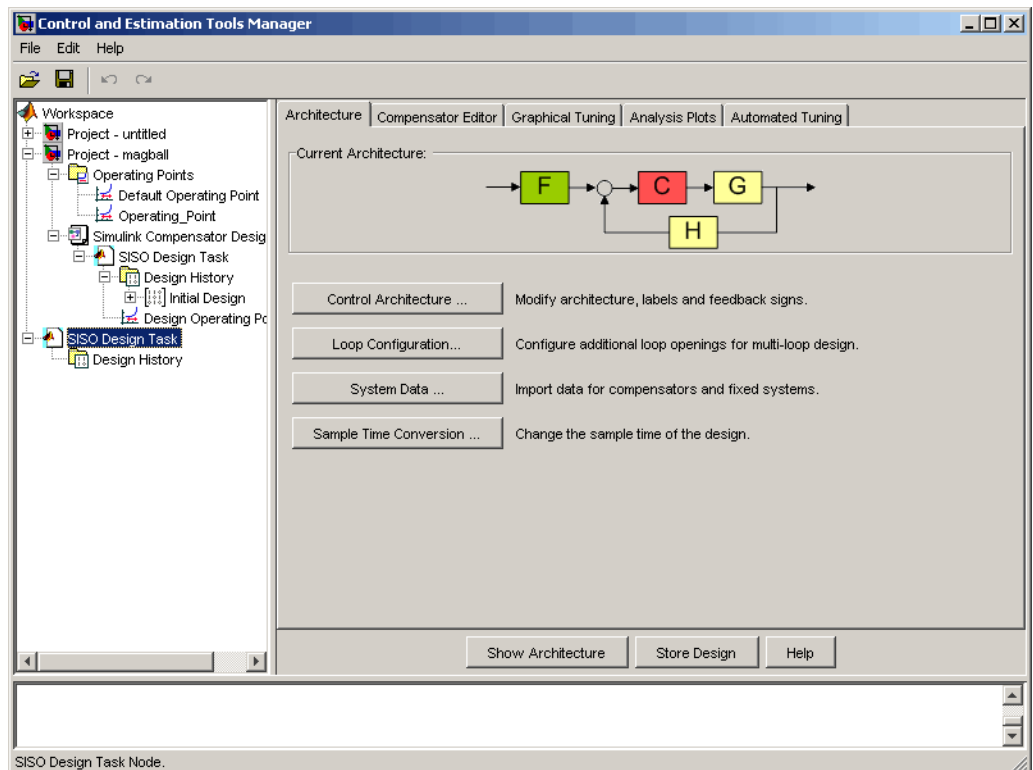


As you can see, the system is now stable and the height of the magnetic ball settles at the desired height of 0.05 m.

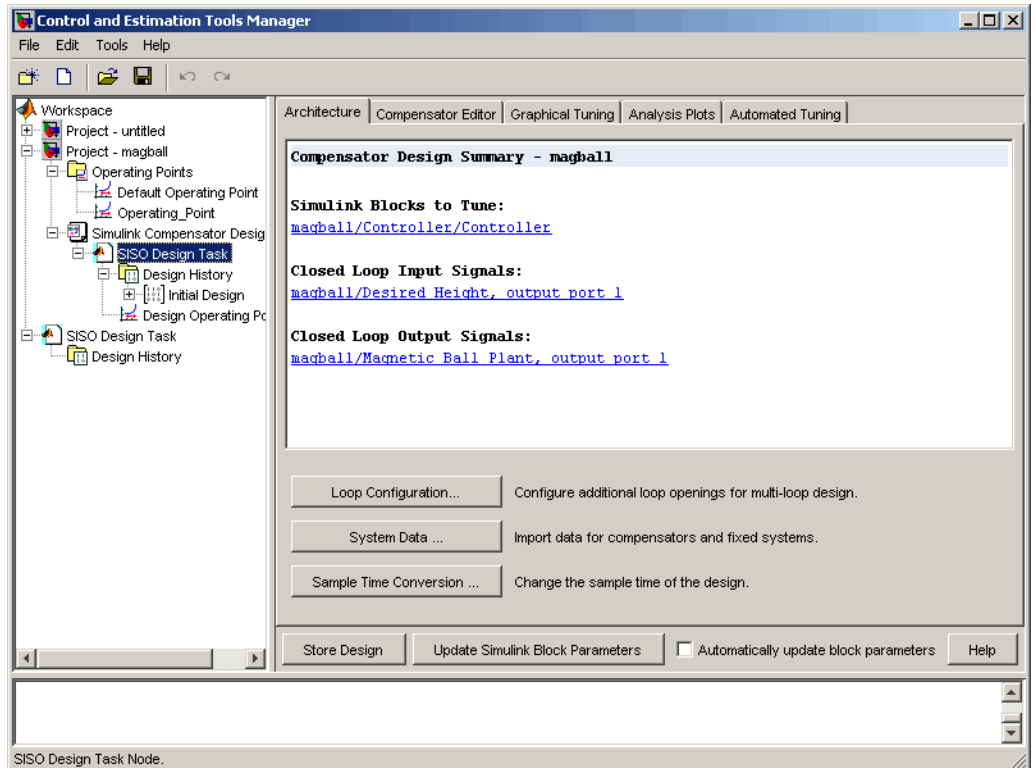
Compare and Contrast the SISO Design Task and Enhanced SISO Design Task

The SISO Design Task is a graphical user interface (GUI) that simplifies the task of designing controllers. This section describes the similarities and differences between the SISO Design Task, which is available in the Control System Toolbox product, and the enhanced SISO Design Task, which is available with the Simulink Control Design product.

The following figure shows the SISO Design Task as it appears in the Control and Estimation Tools Manager.



The following figure shows the enhanced SISO Design Task as it appears under the **Simulink Compensator Design Task** node in the Control and Estimation Tools Manager.



The following table summarizes the similarities and differences between the SISO Design Task and the enhanced SISO Design Task:

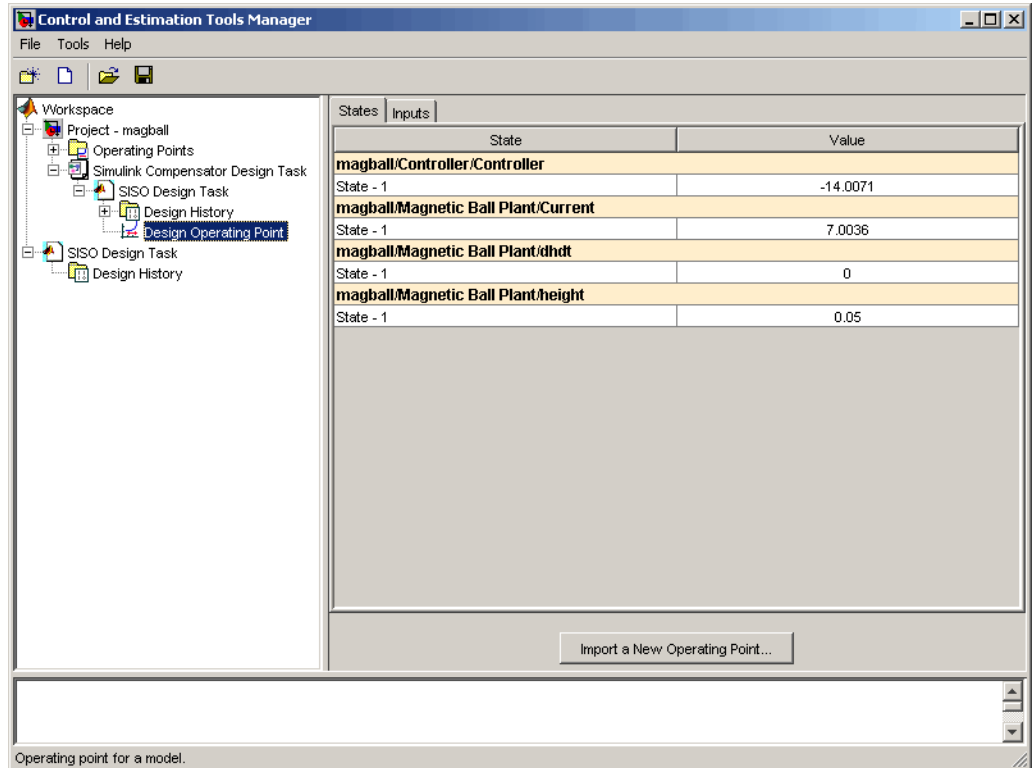
Similarities	Differences
<ul style="list-style-type: none"> • Similar layout • Graphical Tuning, Analysis Plots, and Automated Tuning panes have the same functionality. For more information about these tabs, see “Tools for Compensator Design” on page 7-37. 	<ul style="list-style-type: none"> • Architecture tab — The SISO Tool lets you change the architecture of your system. In contrast, once you create a SISO Design Task you cannot add or delete blocks from your model. Also, the Architecture tab in the SISO Design Task node summarizes the Simulink Blocks to Tune, Closed Loop Input Signals, and Closed Loop Output Signals. • Compensator Editor tab — The SISO Design Tool lets you tune the poles and zeros of your system. The enhanced SISO Design Tool lets you tune the poles, zeros, and parameters of your system. For more information, see the Simulink Control Design demo “Tuning Simulink Blocks in the Compensator Editor”. • When you are satisfied with your system’s performance, the enhanced SISO Design Tool lets you click Update Simulink Block Parameters to write the parameters back to your Simulink model.

For additional information, see:

- “Creating a SISO Design Task” on page 7-26
- “Designing Compensators” in the Control System Toolbox getting started documentation
- “SISO Design Tool” in the Control System Toolbox getting started documentation

Design Operating Point Node

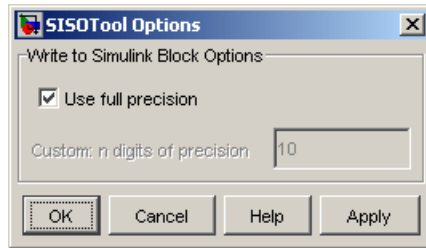
The **Design Operating Point** node is located inside the **Design History** node of the Control and Estimation Tools Manager.



The **States** pane describes the operating point the GUI used to linearize the model. When creating the **SISO Design Task** node, you can use this pane to import a different operating point and to populate the **SISO Design Task** node with a linear model evaluated at the new operating point.

SISO Tool Options

To modify the precision of the numbers calculated by SISO Tool, click the **SISO Design Task** node, and then select **Tools > Options**. The SISOTool Options dialog box opens.



If you select the **Use full precision** check box, the SISO Tool uses the full double-precision data type when writing back to the Simulink block dialog box. If you clear this check box, use **Custom: n digits of precision** to specify the precision you want.

For additional information, see “Creating a SISO Design Task” on page 7-26.

Function Reference

Linearization Analysis I/Os (p. 8-1)	Functions for creating and setting linearization analysis I/Os
Operating Points (p. 8-2)	Functions for creating and working with operating points
Linearization (p. 8-3)	Functions for linearizing Simulink models
Frequency Response Estimation (p. 8-3)	Functions for estimating frequency response models

Linearization Analysis I/Os

<code>get</code>	Properties of linearization I/Os and operating points
<code>getlinio</code>	Linearization I/O settings for Simulink model
<code>linio</code>	Construct linearization I/O settings for Simulink model
<code>set</code>	Set properties of linearization I/Os and operating points
<code>setlinio</code>	Assign I/O settings to Simulink model

Operating Points

<code>addoutputspec</code>	Add output specification to operating point specification
<code>copy</code>	Copy operating point or operating point specification
<code>findop</code>	Find operating points from specifications or simulation
<code>get</code>	Properties of linearization I/Os and operating points
<code>getinputstruct</code>	Input structure from operating point
<code>getstatestruct</code>	State structure from operating point
<code>getxu</code>	States and inputs from operating points
<code>initopspec</code>	Initialize operating point specification values
<code>operpoint</code>	Create operating point for Simulink model
<code>operspec</code>	Create operating point specifications for Simulink model
<code>set</code>	Set properties of linearization I/Os and operating points
<code>setxu</code>	Set states and inputs in operating points
<code>update</code>	Update operating point object with structural changes in model

Linearization

<code>findop</code>	Find operating points from specifications or simulation
<code>getlinio</code>	Linearization I/O settings for Simulink model
<code>getlinplant</code>	Compute open-loop plant model from Simulink diagram
<code>linearize</code>	Create linearized model from Simulink model
<code>linio</code>	Construct linearization I/O settings for Simulink model
<code>linlft</code>	Linearize a model while removing the contribution of specified blocks
<code>linlftfold</code>	Combine linearization results from specified blocks and a model
<code>linoptions</code>	Set options for linearization and finding operating points
<code>operpoint</code>	Create operating point for Simulink model
<code>operspec</code>	Create operating point specifications for Simulink model

Frequency Response Estimation

<code>frest.Chirp</code>	Swept-frequency cosine signal
<code>frest.createFixedTsSinestream</code>	Sinestream input signal with fixed sample time
<code>frest.createStep</code>	Step input signal
<code>frest.Random</code>	Random input signal for simulation

<code>frest.simCompare</code>	Plot time-domain simulation of nonlinear and linear models
<code>frest.simView</code>	Plot frequency response model in time- and frequency-domain
<code>frest.Sinestream</code>	Signal containing series of sine waves
<code>frestimate</code>	Frequency response estimation of Simulink models
<code>generateTimeseries</code>	Generate time-domain data for input signal

Functions — Alphabetical List

addoutputspec

Purpose Add output specification to operating point specification

Syntax `opnew=addoutputspec(op,'block',portnumber)`

Graphical Interface As an alternative to the `addoutputspec` function, add output specifications with the Simulink Control Design GUI. See “Constraining Outputs” on page 2-27.

Description `opnew=addoutputspec(op,'block',portnumber)` adds an output specification for a Simulink model to an existing operating point specification, `op`, created with `operspec`. The signal being constrained by the output specification is indicated by the name of the block, `'block'`, and the port number, `portnumber`, that it originates from.

You can edit the output specification within the new operating point specification object, `opnew`, to include the actual constraints or specifications for the signal. Use the new operating point specification object with the function `findop` to find operating points for the model.

This function automatically compiles the Simulink model, given in the property `Model` of `op`, to find the block’s output portwidth.

Example Create an operating point specification for the model `magball`.

```
op=operspec('magball')
```

This specification returns the object `op`. Note that there are no outputs in this model and no outputs in the object `op`.

```
Operating Specificaton for the Model magball.  
(Time-Varying Components Evaluated at time t=0)
```

```
States:
```

```
-----  
(1.) magball/Controller/Controller  
      spec: dx = 0, initial guess:          0  
      spec: dx = 0, initial guess:          0  
(2.) magball/Magnetic Ball Plant/Current
```



```

        spec: dx = 0, initial guess:          7
(3.) magball/Magnetic Ball Plant/dhdt
        spec: dx = 0, initial guess:          0
(4.) magball/Magnetic Ball Plant/height
        spec: dx = 0, initial guess:          0.05

```

Inputs: None

Outputs: None

To add an output specification to the signal between the Controller block and the Magnetic Ball Plant block, use the function `addoutputspec`.

```
newop=addoutputspec(op, 'magball/Controller',1)
```

This function adds the output specification is added to the operating point specification object.

```

Operating Specificaton for the Model magball.
(Time-Varying Components Evaluated at time t=0)

```

States:

```

(1.) magball/Controller/Controller
    spec: dx = 0, initial guess:          0
        spec: dx = 0, initial guess:          0
(2.) magball/Magnetic Ball Plant/Current
    spec: dx = 0, initial guess:          7
(3.) magball/Magnetic Ball Plant/dhdt
    spec: dx = 0, initial guess:          0
(4.) magball/Magnetic Ball Plant/height
    spec: dx = 0, initial guess:          0.05

```

Inputs: None

Outputs:

addoutputspec

```
(1.) magball/Controller
    spec: none
```

Edit the output specification to constrain this signal to be 14.

```
newop.Outputs(1).Known=1, newop.Outputs(1).y=14
```

The final output specification is displayed.

```
Operating Specificaton for the Model magball.
(Time-Varying Components Evaluated at time t=0)
```

States:

```
-----
(1.) magball/Controller/Controller
    spec: dx = 0, initial guess:      0
    spec: dx = 0, initial guess:      0
(2.) magball/Magnetic Ball Plant/Current
    spec: dx = 0, initial guess:      7
(3.) magball/Magnetic Ball Plant/dhdt
    spec: dx = 0, initial guess:      0
(4.) magball/Magnetic Ball Plant/height
    spec: dx = 0, initial guess:      0.05
```

Inputs: None

Outputs:

```
-----
(1.) magball/Controller
    spec: y = 14
```

See Also

findop, operspec, operpoint

Purpose Copy operating point or operating point specification

Syntax
`op_point2=copy(op_point1)`
`op_spec2=copy(op_spec1)`

Description
`op_point2=copy(op_point1)` returns a copy of the operating point object `op_point1`. You can create `op_point1` with the function `operpoint`.
`op_spec2=copy(op_spec1)` returns a copy of the operating point specification object `op_spec1`. You can create `op_spec1` with the function `operspec`.

Note The command `op_point2=op_point1` does not create a copy of `op_point1` but instead creates a pointer to `op_point1`. In this case, any changes made to `op_point2` are also made to `op_point1`.

Examples Create an operating point object for the model, `magball`.

```
opp=operpoint('magball')
```

The operating point is displayed.

```
Operating Point for the Model magball.  
(Time-Varying Components Evaluated at time t=0)
```

```
States:
```

```
-----
```

- ```
(1.) magball/Controller/Controller
 x: 0
 x: 0
(2.) magball/Magnetic Ball Plant/Current
 x: 7
(3.) magball/Magnetic Ball Plant/dhdt
 x: 0
(4.) magball/Magnetic Ball Plant/height
```

x: 0.05

Inputs: None

Create a copy of this object, opp.

```
new_opp=copy(opp)
```

An exact copy of the object is displayed.

```
Operating Point for the Model magball.
(Time-Varying Components Evaluated at time t=0)
```

States:

-----

(1.) magball/Controller/Controller

x: 0

x: 0

(2.) magball/Magnetic Ball Plant/Current

x: 7

(3.) magball/Magnetic Ball Plant/dhdt

x: 0

(4.) magball/Magnetic Ball Plant/height

x: 0.05

Inputs: None

## See Also

operpoint, operspec

|                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|----------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>             | Find operating points from specifications or simulation                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>Syntax</b>              | <pre>[op_point,op_report]=findop('model',op_spec) [op_point,op_report]=findop('model',op_spec,options) op_point=findop('model',times)</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Graphical Interface</b> | As an alternative to the <code>findop</code> function, create operating points from specifications or simulation within the <b>Operating Points</b> node of the Simulink Control Design GUI. For more information on creating operating points, see “Computing Operating Points from Specifications” on page 3-7 and “Extracting Values from Simulation” on page 3-16.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>Remarks</b>             | Finding operating points from specifications using the <code>findop</code> function is the same as trimming, or performing trim analysis. Use the <code>findop</code> function instead of the Simulink <code>trim</code> function when you work with Simulink Control Design operating point objects and specification objects.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>Description</b>         | <p><code>[op_point,op_report]=findop('model',op_spec)</code> finds an operating point, <code>op_point</code>, of the model, 'model', from specifications given in <code>op_spec</code>.</p> <p><code>[op_point,op_report]=findop('model',op_spec,options)</code> finds an operating point, <code>op_point</code>, of the model, 'model', from specifications given in <code>op_spec</code>. Several options for the optimization are specified in the <code>options</code> object, which you can create with the function <code>linoptions</code>.</p> <p>The input to <code>findop</code>, <code>op_spec</code>, is an operating point specification object. Create this object with the function <code>operspec</code>. Specifications on the operating points, such as minimum and maximum values, initial guesses, and known values, are specified by editing <code>op_spec</code> directly or by using <code>get</code> and <code>set</code>. To find equilibrium, or steady-state, operating points, set the <code>SteadyState</code> property of the states and inputs in <code>op_spec</code> to 1. The <code>findop</code> function uses optimization to find operating points that closely meet the specifications in <code>op_spec</code>. By default, <code>findop</code> uses the optimizer <code>graddescent_elim</code>. To use a different optimizer, change the value of <code>OptimizerType</code> in <code>options</code> using the <code>linoptions</code> function.</p> |

A report object, `op_report`, gives information on how closely `findop` meets the specifications. The function `findop` displays the report automatically, even if the output is suppressed with a semicolon. To turn off the display of the report, set `DisplayReport` to 'off' in options using the function `linoptions`.

`op_point=findop('model',times)` runs a simulation of the model, 'model', and extracts operating points from the simulation at the *snapshot* times given in the vector, `times`. An operating point object, `op_point`, is returned.

---

**Note** For all syntaxes, `findop` automatically uses the following properties in the Simulink model:

- `BufferReuse` = 'off'
- `RTWInlineParameters` = 'on'
- `BlockReductionOpt` = 'off'

Simulink restores the original property values after finding the operating point.

---

The output of `findop` is always an operating point object. Use this object with the function `linearize` to create linearized models of Simulink models. The operating point object has the following properties:

- “Model” on page 9-8
- “States” on page 9-9
- “Inputs” on page 9-9
- “Time” on page 9-10

## Model

`Model` specifies the name of the Simulink model to which this operating point object refers.

## States

`States` describes the operating points of states in the Simulink model. The `States` property is a vector of state objects that contains the operating point values of the states. There is one state object per block that has a state in the Simulink model. The `States` object has the following properties:

|                                |                                                                                                                |
|--------------------------------|----------------------------------------------------------------------------------------------------------------|
| <code>Nx</code>                | Number of states in the block. This property is read-only.                                                     |
| <code>Block</code>             | Block with which the states are associated.                                                                    |
| <code>x</code>                 | Vector containing the values of states in the block.                                                           |
| <code>Ts</code>                | Vector containing the sample time and offset for the state.                                                    |
| <code>SampleType</code>        | Set this value to <code>CSTATE</code> , for a continuous state, or <code>DSTATE</code> , for a discrete state. |
| <code>inReferencedModel</code> | Set this value to 1, when the state is inside a referenced model, or 0, when it is not.                        |
| <code>Description</code>       | Text string describing the block.                                                                              |

## Inputs

`Inputs` is a vector of input objects that contains the input levels at the operating point. There is one input object per root-level inport block in the Simulink model. The `Inputs` object has the following properties:

|                        |                                                        |
|------------------------|--------------------------------------------------------|
| <code>Block</code>     | Inport block with which the input vector is associated |
| <code>PortWidth</code> | Width of the corresponding inport                      |

|                |                                                          |
|----------------|----------------------------------------------------------|
| <code>u</code> | Vector containing the input level at the operating point |
| Description    | Text string describing the input                         |

## Time

`Time` specifies the time at which any time-varying functions in the model are evaluated.

The operating point report object, returned when finding operating points from specifications, has the following properties:

- `Model`
- `Inputs`
- `Outputs`
- `States`
- `Time`
- `TerminationString`
- `OptimizationOutput`

Of these properties, `Model`, `Inputs`, `Outputs`, `States`, and `Time` contain the same information as the operating point specification object, with the addition of `dx` values for the `States` and `yspec` values, or desired `y` values, for the `Outputs`. The `TerminationString` contains the message that `findop` displays after terminating the optimization. The `OptimizationOutput` property contains the same properties returned in the output variable of the Optimization Toolbox functions `fmincon`, `fminsearch`, and `lsqnonlin`. See the Optimization Toolbox documentation for more information. If you do not have Optimization Toolbox software, you can access the documentation at:

<http://www.mathworks.com/access/helpdesk/help/toolbox/optim/optim.shtml>



## Examples

### Example 1

Create an operating point specification object for the model `magball` with the `operspec` function.

```
op_spec=operspec('magball');
```

Edit the operating point specification object to reflect any specifications on the operating points such as minimum and maximum values, initial guesses, and known values. This example uses the default specifications in which `SteadyState` is set to 1 for all states, specifying that an equilibrium operating point is desired.

Find the equilibrium operating points with the `findop` function.

```
op_point=findop('magball',op_spec)
```

This function returns an operating point object, `op_point`.

```
Operating Point for the Model magball.
(Time-Varying Components Evaluated at time t=0)
```

```
States:
```

```

```

- (1.) magball/Controller/Controller  
x: 0  
x: -2.56e-006
- (2.) magball/Magnetic Ball Plant/Current  
x: 7
- (3.) magball/Magnetic Ball Plant/dhdt  
x: 0
- (4.) magball/Magnetic Ball Plant/height  
x: 0.05

```
Inputs: None
```

The MATLAB window displays the name of the model, the time at which any time-varying functions in the model are evaluated, the names of blocks containing states, and the operating point values of the

states. In this example, there are four blocks that contain states in the model and four entries in the `States` object. The first entry contains two states. MATLAB also displays the `Inputs` field although there are no inputs in this model. To view the properties of `op_point` in more detail, use the `get` function.

MATLAB also displays the operating point report object.

```
Operating Point Search Report for the Model magball.
(Time-Varying Components Evaluated at time t=0)
```

```
Operating condition specifications were successfully met.
```

```
States:
```

```

```

```
(1.) magball/Controller/Controller
 x: 0 dx: 0 (0)
 x: -2.56e-006 dx: 0 (0)
(2.) magball/Magnetic Ball Plant/Current
 x: 7 dx: 0 (0)
(3.) magball/Magnetic Ball Plant/dhdt
 x: 0 dx: -1.78e-015 (0)
(4.) magball/Magnetic Ball Plant/height
 x: 0.05 dx: 0 (0)
```

```
Inputs: None
```

```
Outputs: None
```

In addition to the operating point values, the report shows how closely the specifications were met. In the preceding report, the `dx` values are all small and close to the desired `dx` values of 0 indicating that an equilibrium or steady-state value was found.

## Example 2

To extract an operating point from a simulation at the times 10 and 20 using `findop`, enter the following:

```
op_point=findop('magball',[10,20])
```

This function returns the message:

```
There is more than one operating point. Select an element
in the vector of operating points to display.
```

To display the first operating point, enter the command

```
op_point(1)
```

This command should display:

```
Operating Point for the Model magball.
(Time-Varying Components Evaluated at time t=10)
```

```
States:
```

```

```

```
(1.) magball/Controller/Controller
 x: -4.82e-010
 x: -2.56e-006
(2.) magball/Magnetic Ball Plant/Current
 x: 7
(3.) magball/Magnetic Ball Plant/dhdt
 x: 2.6e-006
(4.) magball/Magnetic Ball Plant/height
 x: 0.05
```

```
Inputs: None
```

To display the second operating point, enter:

```
op_point(2)
```

This function returns:

```
Operating Point for the Model magball.
(Time-Varying Components Evaluated at time t=20)
```

States:

-----

- (1.) magball/Controller/Controller  
x: -5.5e-010  
x: -2.56e-006
- (2.) magball/Magnetic Ball Plant/Current  
x: 7
- (3.) magball/Magnetic Ball Plant/dhdt  
x: 2.54e-006
- (4.) magball/Magnetic Ball Plant/height  
x: 0.05

Inputs: None

## See Also

operspec, linearize

**Purpose** Swept-frequency cosine signal

**Syntax**

```
input = frest.Chirp(sys)
input = frest.Chirp('OptionName',OptionValue)
```

**Description** `input = frest.Chirp(sys)` creates a swept-frequency cosine input signal based on the dynamics of a linear system `sys`.

`input = frest.Chirp('OptionName',OptionValue)` creates a swept-frequency cosine input signal using the options specified by comma-separated name/value pairs.

To view a plot of your input signal, type `plot(input)`. To obtain a timeseries for your input signal, use the `generateTimeseries` command.

**Inputs** `sys`

Linear system for creating a chirp signal that matches the dynamic characteristics of this system. You can specify the linear system based on known dynamics using `tf`, `zpk`, or `ss`. You can also obtain the linear system by linearizing a nonlinear system.

The resulting chirp signal automatically sets these options based on the linear system:

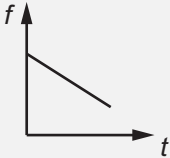
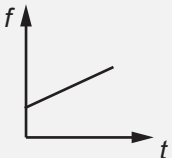
- `'FreqRange'` are the frequencies at which the linear system has interesting dynamics.
- `'Ts'` is set to avoid aliasing such that the Nyquist frequency of the signal is five times the upper end of the frequency range.
- `'NumSamples'` is set such that the frequency response estimation includes the lower end of the frequency range.

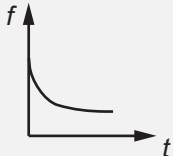
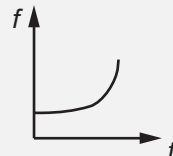
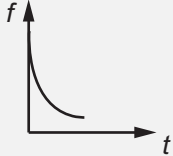
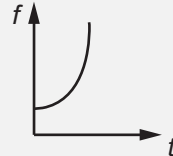
Other chirp options have default values.

*'OptionName',OptionValue*

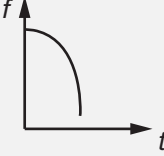
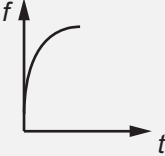
Signal characteristics, specified as comma-separated pairs of option name string and the option value.

| Option Name | Option Value                                                                                                                                                                                                                    |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 'Amplitude' | Signal amplitude.<br><br><b>Default:</b> 1e-5                                                                                                                                                                                   |
| 'FreqRange' | Signal frequencies, specified as either: <ul style="list-style-type: none"> <li>• Two-element vector, for example [w1 w2]</li> <li>• Two-element cell array, for example {w1 w2}</li> </ul><br><b>Default:</b> [1,1000]         |
| 'FreqUnits' | Frequency units: <ul style="list-style-type: none"> <li>• 'rad/s'—Radians per second</li> <li>• 'Hz'—Hertz</li> </ul><br>Changing frequency units does not impact frequency response estimation.<br><br><b>Default:</b> 'rad/s' |
| 'Ts'        | Sample time of the chirp signal in seconds. The default setting avoids aliasing.<br><br><b>Default:</b> $\frac{2\pi}{5 * \max(\text{FreqRange})}$                                                                               |

| Option Name   | Option Value                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|---------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 'NumSamples'  | <p>Number of samples in the chirp signal. Default setting ensures that the estimation includes the lower end of the frequency range.</p> <p><b>Default:</b> <math>\frac{2\pi}{T_0 * \min(\text{FreqRange})}</math></p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 'SweepMethod' | <p>Method for evolution of instantaneous frequency:</p> <ul style="list-style-type: none"> <li>'linear' (default)—Specifies the instantaneous frequency sweep <math>f_i(t)</math>:           <math display="block">f_i(t) = f_0 + \beta t \text{ where } \beta = (f_1 - f_0) / t_f</math> <p><math>\beta</math> ensures that the signal maintains the desired frequency breakpoint <math>f_1</math> at final time <math>t_f</math>.</p> <div style="display: flex; justify-content: space-around; align-items: center;"> <div style="text-align: center;"> <p><b>f1 &gt; f2</b></p>  </div> <div style="text-align: center;"> <p><b>f1 &lt; f2</b></p>  </div> </div> </li> <li>'logarithmic'—Specifies the instantaneous frequency sweep <math>f_i(t)</math> given by           <math display="block">f_i(t) = f_0 \times \beta^t \text{ where } \beta = \left(\frac{f_1}{f_0}\right)^{\frac{1}{t_f}}</math> </li> </ul> |

| Option Name | Option Value                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|             | <div style="display: flex; justify-content: space-around; align-items: flex-start;"> <div style="text-align: center;"> <p><b>f1 &gt; f2</b></p>  </div> <div style="text-align: center;"> <p><b>f1 &lt; f2</b></p>  </div> </div> <ul style="list-style-type: none"> <li>'quadratic'—Specifies the instantaneous frequency sweep <math>f_i(t)</math>:           <math display="block">f_i(t) = f_0 + \beta t^2 \text{ where } \beta = (f_1 - f_0) / t_i^2</math>           Also specify the shape of the quadratic using the 'Shape' option.         </li> </ul> |
| 'Shape'     | <p>Use when you set 'SweepMethod' to 'quadratic' to describe the shape of the parabola in the positive frequency axis:</p> <ul style="list-style-type: none"> <li>'concave'—Concave quadratic sweeping shape.</li> </ul> <div style="display: flex; justify-content: space-around; align-items: flex-start;"> <div style="text-align: center;"> <p><b>f1 &gt; f2</b></p>  </div> <div style="text-align: center;"> <p><b>f1 &lt; f2</b></p>  </div> </div> <ul style="list-style-type: none"> <li>'convex'—Convex quadratic sweeping shape.</li> </ul>       |



| Option Name    | Option Value                                                                                                                                                                                                                                                                                                                                                                                                       |
|----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                | <div style="display: flex; justify-content: space-around; align-items: center;"> <div style="text-align: center;"> <p><math>f_1 &gt; f_2</math></p>  </div> <div style="text-align: center;"> <p><math>f_1 &lt; f_2</math></p>  </div> </div> |
| 'InitialPhase' | <p>Initial phase of the Chirp signal in degrees.</p> <p><b>Default:</b> 270</p>                                                                                                                                                                                                                                                                                                                                    |

## Examples

Create a chirp input signal:

```
input = frest.Chirp('Amplitude',1e-3,'FreqRange',[10 500],'NumSamples',20000)
```

## See Also

frest.Sinestream | frest.Random | frestimate | generateTimeseries

## How To

- “Creating Input Signals for Estimation” on page 6-7
- Chapter 6, “Frequency Response Estimation of Simulink Models”
- “Validating Exact Linearization Results” on page 4-76

# frest.createFixedTsSinestream

---

**Purpose** Sinestream input signal with fixed sample time

**Syntax**

```
input = frest.createFixedTsSinestream(ts)
input = frest.createFixedTsSinestream(ts,{wmin wmax})
input = frest.createFixedTsSinestream(ts,w)
input = frest.createFixedTsSinestream(ts,sys)
input = frest.createFixedTsSinestream(ts,sys,{wmin wmax})
input = frest.createFixedTsSinestream(ts,sys,w)
```

**Description** `input = frest.createFixedTsSinestream(ts)` creates series of sine waves using a fixed sample time `ts` in seconds. Use when your Simulink model has linearization input I/Os on signals with discrete sample times.

`input = frest.createFixedTsSinestream(ts,{wmin wmax})` creates series of sine waves with frequencies between `wmin` and `wmax` in rad/s.

`input = frest.createFixedTsSinestream(ts,w)` creates series of sine waves with frequencies `w`, specified as a vector of frequency values

in rad/s. The values of `w` must satisfy  $w = \frac{2\pi N}{ts}$  for integer  $N$  such that the sample rate `ts` is an integer multiple of each element of `w`.

`input = frest.createFixedTsSinestream(ts,sys)` creates series of sine waves with a fixed sample time `ts` based on the dynamics of a linear system `sys`.

`input = frest.createFixedTsSinestream(ts,sys,{wmin wmax})` creates series of sine waves with a fixed sample time `ts` based on the dynamics of a linear system such that the frequencies are between `wmin` and `wmax`.

`input = frest.createFixedTsSinestream(ts,sys,w)` creates series of sine waves at frequencies `w` in rad/s. The values of `w` must satisfy

$w = \frac{2\pi N}{ts}$  for integer  $N$ . In this case, the sample rate corresponding to `ts` is an integer multiple of each element in `w`.

## Examples

Create a sinusoidal input signal with the following characteristics:

- Sample time of 0.02 sec
- Frequencies of the sinusoidal signal are between 1 rad/s and 10 rad/s

```
input = frest.createFixedTsSinestream(0.02, {1, 10});
```

## See Also

frest.Sinestream | frestimate

## How To

- “Creating Input Signals for Estimation” on page 6-7
- Chapter 6, “Frequency Response Estimation of Simulink Models”
- “Validating Exact Linearization Results” on page 4-76

# frest.createStep

---

**Purpose** Step input signal

**Syntax** `input = frest.createStep('OptionName',OptionValue)`

**Description** `input = frest.createStep('OptionName',OptionValue)` creates a step input signal as a MATLAB timeseries object using the options specified by comma-separated name/value pairs.

Plot your input signal using `plot(input)`.

**Inputs** `'OptionName',OptionValue`

Signal characteristics, specified as comma-separated pairs of option name string and the option value.

| Option Name | Option Value                                                                                         |
|-------------|------------------------------------------------------------------------------------------------------|
| 'Ts'        | Sample time of the step input in seconds.<br><b>Default:</b> 1e-3                                    |
| 'StepTime'  | Time in seconds when the output jumps from 0 to the StepSize parameter.<br><b>Default:</b> 1         |
| 'StepSize'  | Value of the step signal after time reaches and exceeds the StepTime parameter.<br><b>Default:</b> 1 |
| 'FinalTime' | Final time of the step input signal in seconds.<br><b>Default:</b> 10                                |

## Examples

Create step signal:

```
input = frest.createStep('StepTime',3,'StepSize',2)
```

## See Also

[frest.simCompare](#) | [frestimate](#)

## How To

- “Creating Input Signals for Estimation” on page 6-7
- Chapter 6, “Frequency Response Estimation of Simulink Models”
- “Validating Exact Linearization Results” on page 4-76

# frest.Random

---

**Purpose** Random input signal for simulation

**Syntax**  
`input = frest.Random('OptionName',OptionValue)`  
`input = frest.Random(sys)`

**Description** `input = frest.Random('OptionName',OptionValue)` creates the Random input signal using the options specified by comma-separated name/value pairs.

`input = frest.Random(sys)` creates a Random input signal based on the dynamics of a linear system `sys`.

To view a plot of your input signal, type `plot(input)`. To obtain a timeseries for your input signal, use the `generateTimeseries` command.

**Inputs** `sys`

Linear system for creating a random signal that matches the dynamic characteristics of this system. You can specify the linear system based on known dynamics using `tf`, `zpk`, or `ss`. You can also obtain the linear system by linearizing a nonlinear system.

The resulting random signal automatically sets these options based on the linear system:

- `Ts` is set such that the Nyquist frequency of the signal is five times the upper end of the frequency range to avoid aliasing issues.
- `NumSamples` is set such that the frequency response estimation includes the lower end of the frequency range.

Other random options have default values.

*'OptionName',OptionValue*

Signal characteristics, specified as comma-separated pairs of option name string and the option value.

| Option Name  | Option Value                                                                                                                                                                                                                                                                                                                                                                       |
|--------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 'Amplitude'  | Signal amplitude.<br><b>Default:</b> 1e-5                                                                                                                                                                                                                                                                                                                                          |
| 'Ts'         | Sample time of the chirp signal in seconds.<br><b>Default:</b> 1e-3                                                                                                                                                                                                                                                                                                                |
| 'NumSamples' | Number of samples in the Random signal.<br><b>Default:</b> 1e4                                                                                                                                                                                                                                                                                                                     |
| 'Stream'     | Random number stream you create using the MATLAB command <code>RandStream</code> . The state of the stream you specify stores with the input signal. This stored state allows the software to return the same result every time you use <code>generateTimeseries</code> and <code>frestimate</code> with the input signal.<br><b>Default:</b> Default stream of the MATLAB session |

## Examples

Create a Random input signal with 1000 samples taken at 100 Hz and amplitude of 0.02:

```
input = frest.Random('Amplitude',0.02,'Ts',1/100,'NumSamples',1000);
```

Create a Random input signal using multiplicative lagged Fibonacci generator random stream:

```
% Specify the random number stream
stream = RandStream('mlfg6331_64','Seed',0);
```

# frest.Random

---

```
% Create the input signal
input = frest.Random('Stream',stream);
```

## See Also

[frest.Sinestream](#) | [frest.Random](#) | [frestimate](#) | [generateTimeseries](#)

## How To

- “Creating Input Signals for Estimation” on page 6-7
- Chapter 6, “Frequency Response Estimation of Simulink Models”
- “Validating Exact Linearization Results” on page 4-76



## Purpose

Plot time-domain simulation of nonlinear and linear models

## Syntax

```
frest.simCompare(simout,sys,input)
frest.simCompare(simout,sys,input,x0)
```

## Description

`frest.simCompare(simout,sys,input)` plots both

- Simulation output, `simout`, of the nonlinear Simulink model  
You obtain the output from the `frestimate` command.
- Simulation output of the linear model `sys` for the input signal `input`  
The linear simulation results are offset by the initial output values in the `simout` data.

`frest.simCompare(simout,sys,input,x0)` plots the frequency response simulation output and the simulation output of the linear model with initial state `x0`. Because you specify the initial state, the linear simulation result is *not* offset by the initial output values in the `simout` data.

## Examples

Compare a time-domain simulation of the Simulink `watertank` model and its linear model representation:

```
% Create input signal for simulation
input = frest.createStep('FinalTime',100);

% Open the Simulink model
watertank

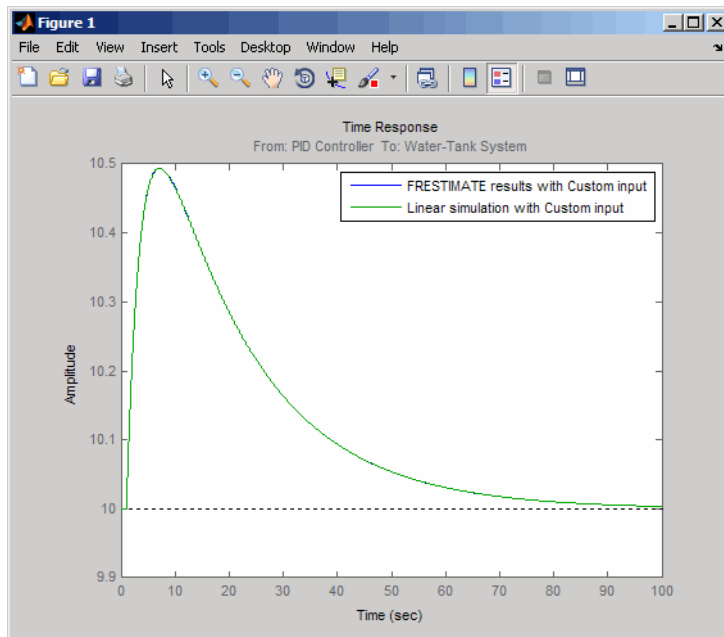
% Specify the operating point for the estimation
watertank_spec = operspec('watertank');
op = findop('watertank',watertank_spec)

% Specify portion of model to estimate
io(1)=linio('watertank/PID Controller',1,'in');
io(2)=linio('watertank/Water-Tank System',1,'out');
```

# frest.simCompare

```
% Estimate the frequency response of the magball model
[sys, simout] = frestimate('watertank', op, io, input)
sys = linearize('watertank', op, io)
frest.simCompare(simout, sys, input)
```

The software returns the following plot.



## See Also

frestimate | frest.simView

## How To

- “Validating Exact Linearization Results” on page 4-76

## Purpose

Plot frequency response model in time- and frequency-domain

## Syntax

```
frest.simView(simout,input,syseset)
frest.simView(simout,input,syseset,sys)
```

## Description

`frest.simView(simout,input,syseset)` plots the following frequency response estimation results:

- Time-domain simulation `simout` of the Simulink model
- FFT of time-domain simulation `simout`
- Bode of estimated system `syseset`

This Bode plot is available when you create the input signal using `frest.Sinestream` or `frest.Chirp`. In this plot, you can interactively select frequencies or a frequency range for viewing the results in all three plots.

You obtain `simout` and `syseset` from the `frestimate` command using the input signal `input`.

`frest.simView(simout,input,syseset,sys)` includes the linear system `sys` in the Bode plot when you create the input signal using `frest.Sinestream` or `frest.Chirp`. Use this syntax to compare the linear system to the frequency response estimation results.

## Examples

Estimate the closed-loop of the `watertank` Simulink model and analyze the results:

```
% Open the Simulink model
watertank

% Specify portion of model to linearize and estimate
io(1)=linio('watertank/PID Controller',1,'in');
io(2)=linio('watertank/Water-Tank System',1,'out');

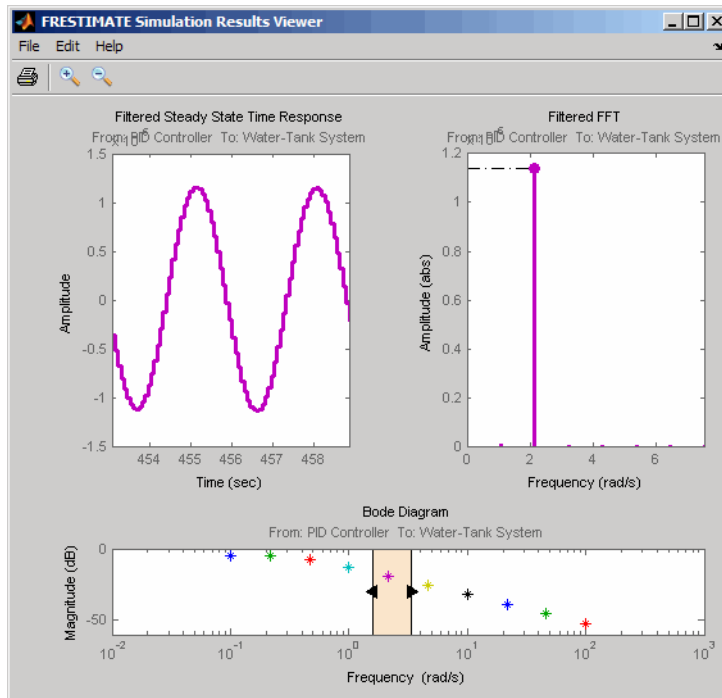
% Specify the operating point for the linearization and estimation
watertank_spec = operspec('watertank');
```

```
op = findop('watertank',watertank_spec);

% Create input signal for simulation
input = frest.Sinestream('Frequency',logspace(-1,2,10));

% Estimate the frequency response of the magball model
[sysest,simout] = frestimate('watertank',op,io,input);

% Analyze the estimation results
frest.simView(simout,input,sysest)
```



## See Also

frestimate | frest.simCompare

## How To

- “Analyzing Estimated Frequency Response” on page 6-20

- Chapter 6, “Frequency Response Estimation of Simulink Models”
- “Validating Exact Linearization Results” on page 4-76

# frest.Sinestream

---

**Purpose** Signal containing series of sine waves

**Syntax**  
`input = frest.Sinestream(sys)`  
`input = frest.Sinestream('OptionName',OptionValue)`

**Description** `input = frest.Sinestream(sys)` creates a signal with a series of sinusoids based on the dynamics of a linear system `sys`.

`input = frest.Sinestream('OptionName',OptionValue)` creates a signal with a series of sinusoids, where each sinusoid frequency lasts for a specified number of periods, using the options specified by comma-separated name/value pairs.

To view a plot of your input signal, type `plot(input)`. To obtain a timeseries for your input signal, use the `generateTimeseries` command.

**Inputs** `sys`

Linear system for creating a sinestream signal that matches the dynamic characteristics of this system. You can specify the linear system based on known dynamics using `tf`, `zpk`, or `ss`. You can also obtain the linear system by linearizing a nonlinear system.

The resulting sinestream signal automatically sets these options based on the linear system:

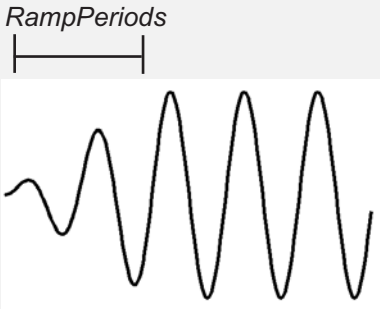
- 'Frequency' are the frequencies at which the linear system has interesting dynamics.
- 'SettlingPeriods' is the number of periods it takes the system to reach steady state at each frequency in 'Frequency'.
- 'NumPeriods' is (3 + SettlingPeriods) to ensure that each frequency excites the system at maximum amplitude for at least three periods.
- For discrete systems only, 'SamplesPerPeriod' is set such that all frequencies have the same sample time as the linear system.

Other sinestream options have default values.

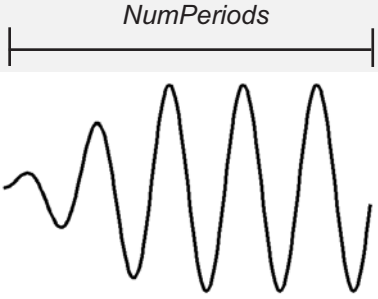
*'OptionName', OptionValue*

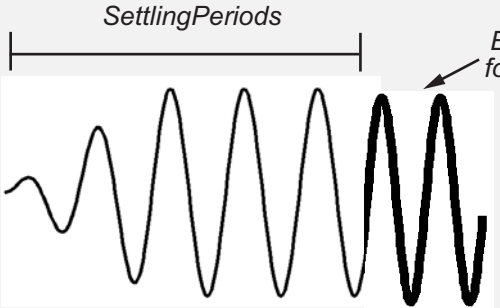
Signal characteristics, specified as comma-separated pairs of option name string and the option value.

| Option Name        | Option Value                                                                                                                                                                                                                                                                                     |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 'Frequency'        | Signal frequencies, specified as either a scalar or a vector of frequency values.<br><br><b>Default:</b> <code>logspace(1,3,50)</code>                                                                                                                                                           |
| 'Amplitude'        | Signal amplitude at each frequency, specified as either: <ul style="list-style-type: none"> <li>• Scalar when all frequencies have the same value</li> <li>• Vector of values when the value is different across frequencies</li> </ul> <b>Default:</b> <code>1e-5</code>                        |
| 'SamplesPerPeriod' | Number of samples for each period for each signal frequency, specified as either: <ul style="list-style-type: none"> <li>• Scalar when all frequencies have the same value</li> <li>• Vector of values when the value is different for each frequency</li> </ul> <b>Default:</b> <code>40</code> |

| Option Name   | Option Value                                                                                                                                                                                                                                                                                                                                                                                                                          |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 'FreqUnits'   | <p>Frequency units:</p> <ul style="list-style-type: none"><li>• 'rad/s'—Radians per second</li><li>• 'Hz'— Hertz</li></ul> <p>Changing frequency units does not impact frequency response estimation.</p> <p><b>Default:</b> 'rad/s'</p>                                                                                                                                                                                              |
| 'RampPeriods' | <p>Number of periods for ramping up the amplitude of each sine wave to its maximum value, specified as either:</p> <ul style="list-style-type: none"><li>• Scalar when all frequencies have the same value</li><li>• Vector of values when the value is different for each frequency</li></ul> <p><b>Default:</b> 0</p> <p><i>RampPeriods</i></p>  |



| Option Name  | Option Value                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|--------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 'NumPeriods' | <p>Number of periods each sine wave is at maximum amplitude, specified as either:</p> <ul style="list-style-type: none"> <li>• Scalar when all frequencies have the same value</li> <li>• Vector of values when the value is different for each frequency</li> </ul> <p>Use this option to ensure a smooth response when your input amplitude changes.</p> <p><b>Default:</b><br/> <math>\max(3 \text{ RampPeriods} + \text{SettlingPeriods}, 2)</math></p>  |

| Option Name       | Option Value                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 'SettlingPeriods' | <p data-bbox="783 317 1270 473">Number of periods corresponding to the transient portion of the simulated response at a specific frequency, before the system reaches steady state, specified as either:</p> <ul data-bbox="788 479 1274 618" style="list-style-type: none"><li data-bbox="788 479 1274 539">• Scalar when all frequencies have the same value</li><li data-bbox="788 557 1274 618">• Vector of values when the value is different for each frequency</li></ul> <p data-bbox="783 656 1240 748">Before performing the estimation, <code>frestimate</code> discards this number of periods from the output signals.</p> <p data-bbox="783 782 921 808"><b>Default:</b> 1</p>  |

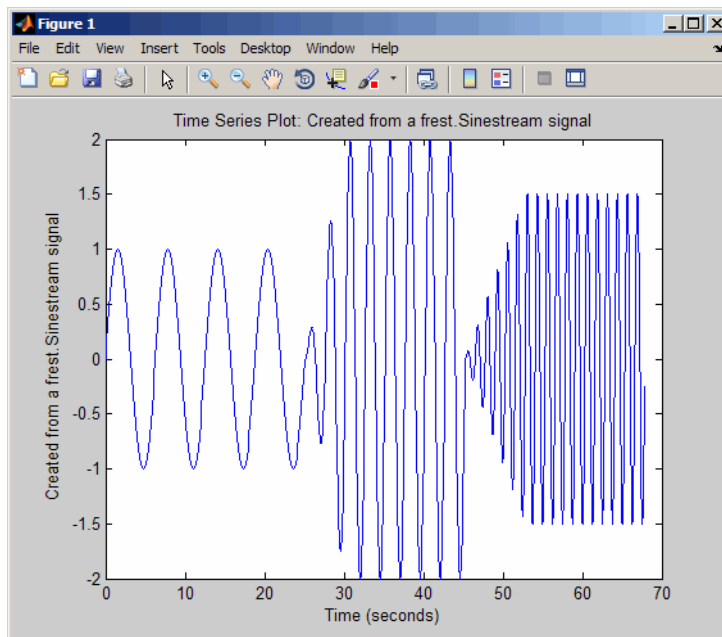
| Option Name                   | Option Value                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|-------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 'ApplyFilteringInFREESTIMATE' | <p>Frequency-selective FIR filtering of the input signal before estimating the frequency response using <code>frestimate</code>.</p> <ul style="list-style-type: none"> <li>• 'on' (default)</li> <li>• 'off'</li> </ul> <p>For more information, see the <code>frestimate</code> algorithm.</p>                                                                                                                                                                          |
| 'SimulationOrder'             | <p>When frequency response estimation occurs relative to output simulation at specified input frequencies.</p> <ul style="list-style-type: none"> <li>• 'Sequential' (default)—Simulates output at all input frequencies before estimating the frequency response.</li> <li>• 'OneAtATime'—Simulates the output and estimates the frequency response for each input frequency, on at a time. Use this option when your Simulink model has a fixed-step solver.</li> </ul> |

## Examples

Create a `sinstream` signal at different frequencies, amplitudes, and ramp-up periods:

```
% Create the input signal
input = frest.Sinestream('Frequency',[1 2.5 5],...
 'Amplitude',[1 2 1.5],...
 'NumPeriods',[4 6 12],...
 'RampPeriods',[0 2 6]);

% Plot the input signal
plot(input)
```



Create a sinusoidal input signal with the following characteristics:

- 50 frequencies spaced logarithmically between 10 Hz and 1000 Hz
- All frequencies have amplitude of  $1e-3$
- Sampled with a frequency 10 times the frequency of the signal (meaning ten samples per period)

```
% Create the input signal
input = frest.Sinestream('Amplitude',1e-3,'Frequency',logspace(1,3,50),...
 'SamplesPerPeriod',10,'FreqUnits','Hz');
```

## See Also

`frest.Chirp` | `frest.Random` | `frest.estimate` | `generateTimeseries` | `frest.createFixedTsSinestream`

## **How To**

- “Creating Input Signals for Estimation” on page 6-7
- Chapter 6, “Frequency Response Estimation of Simulink Models”
- “Validating Exact Linearization Results” on page 4-76

# frestimate

---

**Purpose** Frequency response estimation of Simulink models

**Syntax**

```
sysest = frestimate(model,io,input)
sysest = frestimate(model,op,io,input)
[sysest,simout] = frestimate(model,op,io,input)
```

**Description** `sysest = frestimate(model,io,input)` estimates frequency response model `sysest`. `model` is a string that specifies the name of your Simulink model. `input` can be a `sinestream`, `chirp`, or random signal, or a Simulink timeseries object. `io` specifies the linearization I/O object, which you create using `getlinio` or `linio`. I/O points cannot be on bus signals. The estimation occurs at the operating point specified in the Simulink model.

`sysest = frestimate(model,op,io,input)` initializes the model at the operating point `op` before estimating the frequency response model. Create `op` using either `operpoint` or `findop`.

`[sysest,simout] = frestimate(model,op,io,input)` estimates frequency response model and returns the simulated output `simout`. This output is a cell array of Simulink timeseries objects with dimensions `m-by-n`. `m` is the number of linearization output points, and `n` is the number of input channels.

**Examples** Estimating frequency response for a Simulink model:

```
% Create input signal for simulation:
input = frest.Sinestream('Frequency',logspace(-3,2,30));

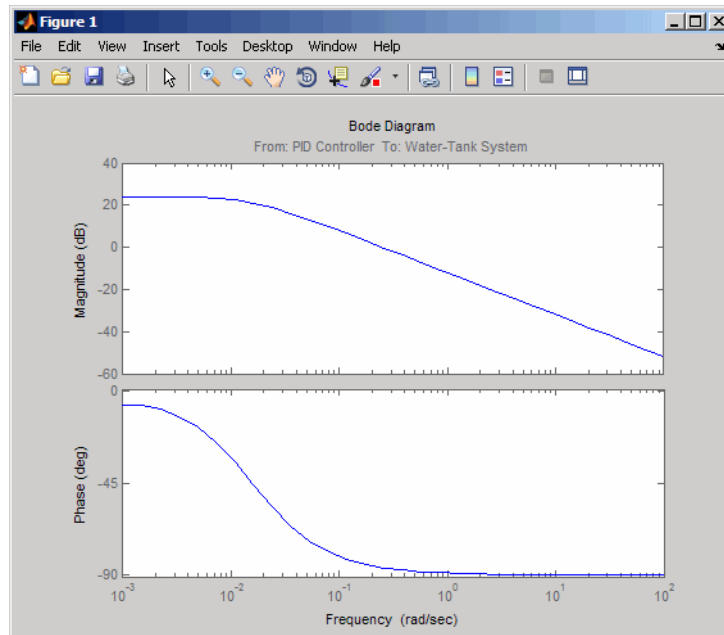
% Open the Simulink model:
watertank

% Specify portion of model to estimate:
io(1)=linio('watertank/PID Controller',1,'in');
io(2)=linio('watertank/Water-Tank System',1,'out','on');

% Specify the steady state operating point for the estimation.
watertank_spec =operspec('watertank');
```

```
op = findop('watertank',watertank_spec);

% Estimate frequency response of specified blocks:
sysest = frestimate('watertank',op,io,input);
bode(sysest)
```



Validate exact linearization results using estimated frequency response of a Simulink model:

```
% Open the Simulink model:
watertank

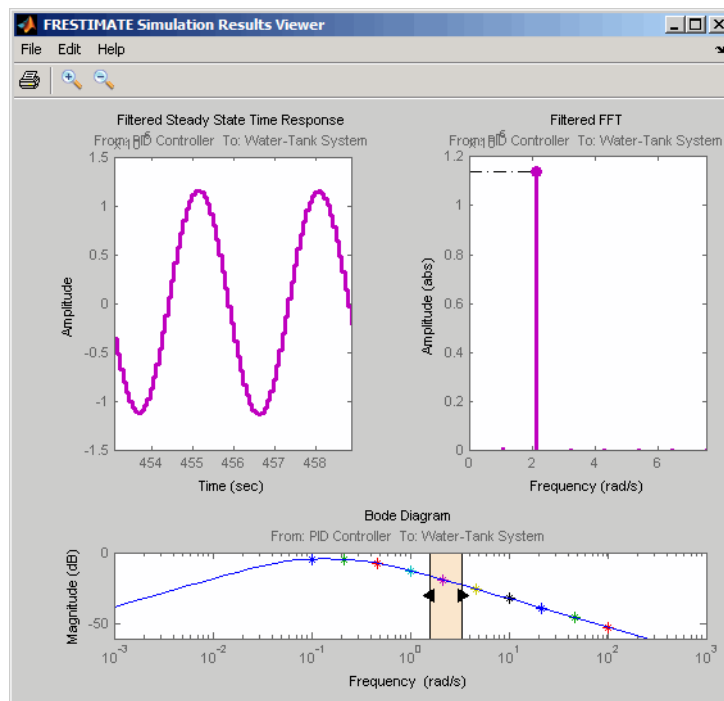
% Specify portion of model to estimate:
io(1)=linio('watertank/PID Controller',1,'in');
io(2)=linio('watertank/Water-Tank System',1,'out');
```

```
% Specify operating point for linearization and estimation:
watertank_spec = operspec('watertank');
op = findop('watertank',watertank_spec);

% Linearize the model:
sys = linearize('watertank',op,io);

% Estimate the frequency response of the magball model
input = frest.Sinestream('Frequency',logspace(-1,2,10));
[sysest,simout] = frestimate('watertank',op,io,input);

% Compare linearization and estimation results in frequency domain:
frest.simView(simout,input,sysest,sys)
```



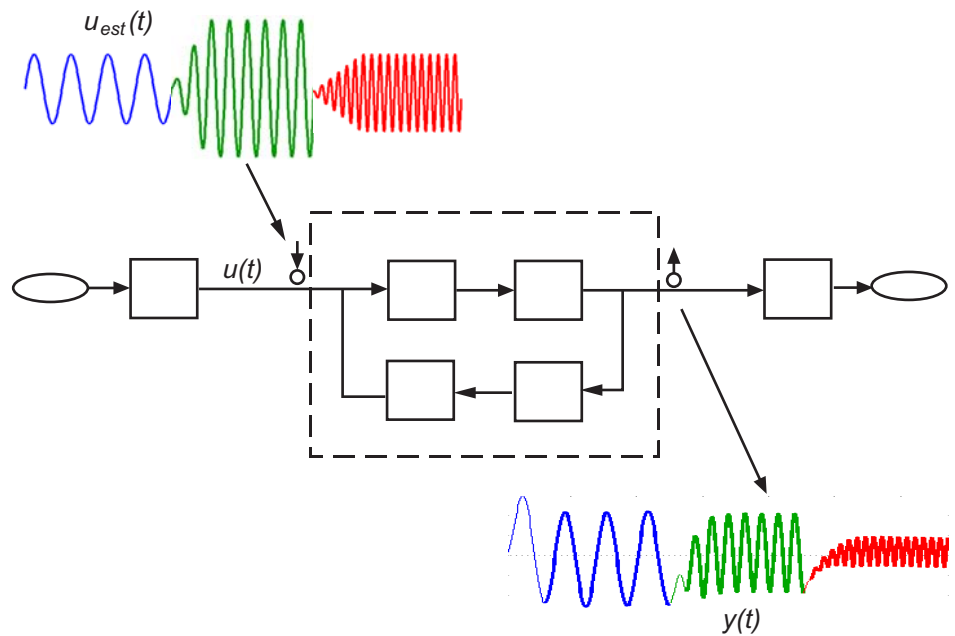


## Algorithm

frestimate performs the following operations when you use the sinestream signal:

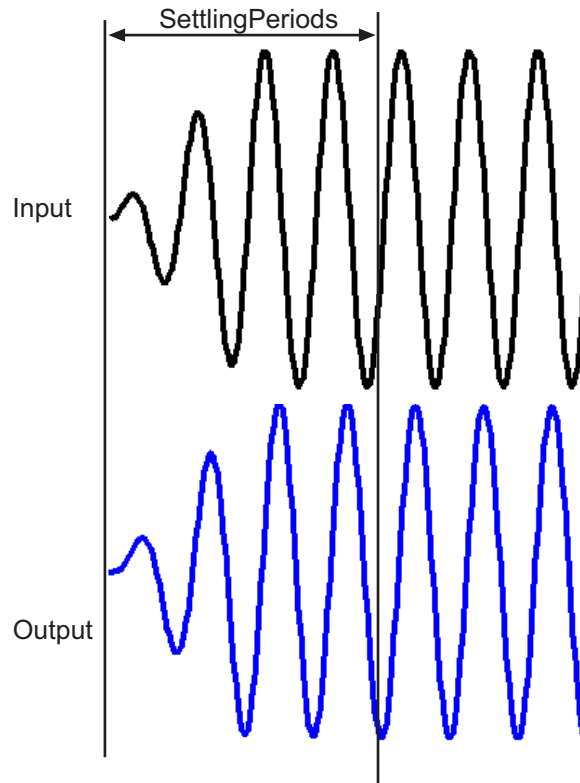
- 1 Injects the sinestream input signal you design,  $u_{est}(t)$ , at the linearization input point.
- 2 Simulates the output at the linearization output point.

frestimate adds the signal you design to existing Simulink signals at the linearization input point.



- Discards the `SettlingPeriods` portion of the output (and the corresponding input) at each frequency.

The simulated output at each frequency has a transient portion and steady state portion. `SettlingPeriods` corresponds to the transient components of the output and input signals. The periods following `SettlingPeriods` are considered to be at steady state.



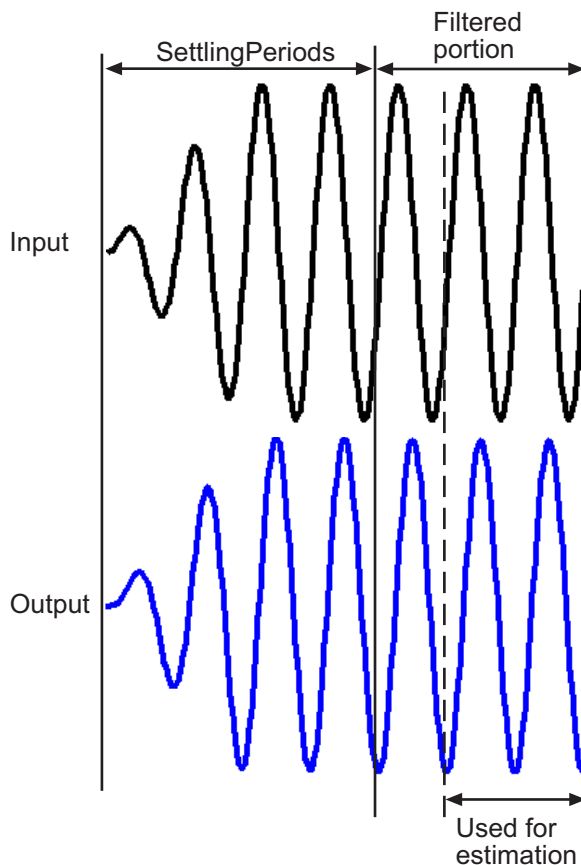
- Filters the remaining portion of the output and the corresponding input signals at each input frequency using a bandpass filter. Because most models are not at steady state, the response might

contain low-frequency transient behavior. Filtering typically improves the accuracy of your model by removing the effects of frequencies other than the input frequencies, which are problematic when sampling and analyzing data of finite length. These effects are called *spectral leakage*.

# frestimate

---

Any transients associated with filtering are only in the first period of the filtered steady-state output. After filtering, `frestimate` discards the first period of the input and output signals. `frestimate` uses a finite impulse response (FIR) filter, whose order matches the number of samples in a period.



- 5 Estimates the frequency response of the processed signal by computing the ratio of the fast Fourier transform of the filtered

steady-state portion of the output signal  $y_{est}(t)$  and the fast Fourier transform of the filtered input signal  $u_{est}(t)$ :

$$\text{Frequency Response Model} = \frac{\text{fft of } y_{est}(t)}{\text{fft of } u_{est}(t)}$$

To compute the response at each frequency, `frestimate` uses only the simulation output at that frequency.

**See Also**

`frest.Sinestream` | `frest.Chirp` | `frest.Random` | `frest.simView`

**How To**

- Chapter 6, “Frequency Response Estimation of Simulink Models”
- “Validating Exact Linearization Results” on page 4-76

# generateTimeseries

---

**Purpose** Generate time-domain data for input signal

**Syntax** `ts = generateTimeseries(input)`

**Description** `ts = generateTimeseries(input)` creates a MATLAB `timeseries` object `ts` from the input signal `input`. `input` can be a `sinestream`, `chirp`, or random signal. For `chirp` and random signals, that time vector of `ts` has equally spaced time values, ranging from 0 to `Ts (NumSamples - 1)`.

**Examples** Create `timeseries` object for chirp signal:

```
input = frest.Chirp('Amplitude',1e-3,'FreqRange',...
 [10 500],'NumSamples',20000);
ts = generateTimeseries(input)
```

**See Also** `frestimate` | `frest.Sinestream` | `frest.Chirp` | `frest.Random`

**Purpose**

Properties of linearization I/Os and operating points

**Syntax**

```
get(ob)
get(ob, 'PropertyName')
ob.PropertyName
```

**Graphical Interface**

As an alternative to the `get` function, view properties of linearization I/Os and operating points with the Simulink Control Design GUI. For more information, see “Inspecting Analysis I/Os” on page 4-52 and Chapter 2, “Operating Point Analysis Using the GUI”.

**Description**

`get(ob)` displays all properties and corresponding values of the object, `ob`, which can be a linearization I/O object, an operating point object, or an operating point specification object. Create `ob` using `findop`, `getlinio`, `linio`, `operpoint`, or `operspec`.

`get(ob, 'PropertyName')` returns the value of the property, `PropertyName`, within the object, `ob`. The object, `ob`, can be a linearization I/O object, an operating point object, or an operating point specification object. Create `ob` using `findop`, `getlinio`, `linio`, `operpoint`, or `operspec`.

`ob.PropertyName` is an alternative notation for displaying the value of the property, `PropertyName`, of the object, `ob`. The object, `ob`, can be a linearization I/O object, an operating point object, or an operating point specification object. Create `ob` using `findop`, `getlinio`, `linio`, `operpoint`, or `operspec`.

**Examples**

Create an operating point object, `op`, for the Simulink model, `magball`.

```
op=operpoint('magball');
```

Get a list of all object properties using the `get` function with the object name as the only input.

```
get(op)
```

This returns the properties of `op` and their current values.

```
Model: 'magball'
States: [4x1 opcond.StatePoint]
Inputs: []
Time: 0
```

To view the value of a particular property of `op`, supply the property name as an argument to `get`. For example, to view the name of the model associated with the operating point object, type:

```
V=get(op, 'Model')
```

which returns

```
V =
magball
```

Because `op` is a structure, you can also view any properties or fields using dot-notation, as in this example.

```
W=op.States
```

This notation returns a vector of objects containing information about the states in the operating point.

```
(1.) magball/Controller/Controller
 x: 0
 x: 0
(2.) magball/Magnetic Ball Plant/Current
 x: 7
(3.) magball/Magnetic Ball Plant/dhdt
 x: 0
(4.) magball/Magnetic Ball Plant/height
 x: 0.05
```

Use `get` to view details of `W`. For example:

```
get(W(2), 'x')
```



returns

```
ans =
 7.0036
```

**See Also**

findop, getlinio, linio, operpoint, operspec, set

# getinputstruct

---

**Purpose** Input structure from operating point

**Syntax** `in_struct = getinputstruct(op_point)`

**Description** `in_struct = getinputstruct(op_point)` extracts a structure of input values, `in_struct`, from the operating point object, `op_point`. The structure, `in_struct`, uses the same format as Simulink software which allows you to set initial values for inputs in the model within the **Data Import/Export** pane of the Configuration Parameters dialog box.

**Example** Create an operating point object for the f14 model:

```
op_f14=operpoint('f14');
```

Extract an input structure from the operating point object:

```
inputs_f14=getinputstruct(op_f14)
```

This extraction returns

```
inputs_f14 =
 time: 0
 signals: [1x1 struct]
```

To view the values of the inputs within this structure, use dot-notation to access the `values` field:

```
inputs_f14.signals.values
```

In this case, the value of the input is 0.

**See Also** `getstatestruct`, `getxu`, `operpoint`

**Purpose** Linearization I/O settings for Simulink model

**Syntax** `io = getlinio('sys')`

**Graphical Interface** As an alternative to the `getlinio` function, view linearization I/Os in the **Analysis I/Os** pane of the **Linearization Task** node within the Simulink Control Design GUI. See “Inspecting Analysis I/Os” on page 4-52.

**Description** `io = getlinio('sys')` finds all linearization annotations in the Simulink model, `sys`, and returns a vector of objects, `io`. Each object represents a linearization annotation in the model and is associated with an output port of a Simulink block. Before running `getlinio`, use the right-click menu to insert the linearization annotations, or I/Os, on the signal lines of the model diagram.

Each object within the vector, `io`, has the following properties:

|            |                                                                                                 |
|------------|-------------------------------------------------------------------------------------------------|
| Active     | Set this value to 'on', when the I/O is used for linearization, and 'off' otherwise             |
| Block      | Name of the block the with which I/O is associated                                              |
| OpenLoop   | Set this value to 'on', when the feedback loop at the I/O is open, and 'off', when it is closed |
| PortNumber | Integer referring to the output port with which the I/O is associated                           |

|             |                                                                                                                                                                                                                                                                                                      |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Type        | Choose one of the following linearization I/O types: <ul style="list-style-type: none"><li>• 'in': linearization input point</li><li>• 'out': linearization output point</li><li>• 'outin': linearization output then input point</li><li>• 'inout': linearization input then output point</li></ul> |
| Description | String description of the I/O object                                                                                                                                                                                                                                                                 |

You can edit this I/O object to change its properties. Alternatively, you can change the properties of `io` using the `set` function. To upload an edited I/O object to the Simulink model diagram, use the `setlinio` function. Use I/O objects with the function `linearize` to create linear models.

## Example

Before creating a vector of I/O objects using `getlinio`, you must add linearization annotations representing the I/Os, such as input points or output points, to a Simulink model.

Open the Simulink model `magball` by typing

```
magball
```

at the MATLAB prompt. Right-click the signal line between the Magnetic Ball Plant and the Controller. Select **Linearization Points > Input Point** from the menu to place an input point on this signal line. A small arrow pointing toward a small circle just above the signal line represents the input point. Right-click the signal line after the Magnetic Ball Plant. Select **Linearization Points > Output Point** from the menu to place an output point on this signal line. A small arrow pointing away from a small circle just above the signal line represents the output point.

To create a vector of I/O objects for this model, type:

```
io=getlinio('magball')
```

This syntax returns a formatted display of the linearization I/Os.

```
Linearization I/Os:
```

```

```

```
Block magball/Controller, Port 1 is marked with the following
properties:
```

- No Loop Opening
- An Input Perturbation

```
Block magball/Magnetic Ball Plant, Port 1 is marked with the
following properties:
```

- An Output Measurement
- No Loop Opening

There are two entries in the vector, `io`, representing the two linearization annotations previously set in the model diagram. MATLAB displays:

- the name of the block associated with the I/O
- the port number associated with the I/O
- the type of IO (input perturbation or output measurement referring to an input point or output point respectively)
- whether the IO is also a loop opening

By default, the I/Os have no loop openings. Display the properties of each I/O object in more detail using the `get` function.

## See Also

`get`, `linearize`, `linio`, `set`, `setlinio`

# getlinplant

---

**Purpose** Compute open-loop plant model from Simulink diagram

**Syntax**  
`[sysp,sysc] = getlinplant(block,op)`  
`[sysp,sysc] = getlinplant(block,op,options)`

**Description** `[sysp,sysc] = getlinplant(block,op)` Computes the open-loop plant seen by a Simulink block labeled `block` (where `block` specifies the full path to the block). The plant model, `sysp`, and linearized block, `sysc`, are linearized at the operating point `op`.

`[sysp,sysc] = getlinplant(block,op,options)` Computes the open-loop plant seen by a Simulink block labeled `block`, using the linearization options specified in `options`.

**Example** To compute the open-loop model seen by the Controller block in the Simulink model `magball`, first create an operating point object using the function `findop`. In this case, you find the operating point from simulation of the model.

```
op=findop('magball',20);
```

Next, compute the open-loop model seen by the block `magball/Controller`, with the `getlinplant` function.

```
[sysp,sysc]=getlinplant('magball/Controller',op)
```

The output variable `sysp` gives the open-loop plant model as follows:

```
a =
 magball/Magn magball/Magn magball/Magn
magball/Magn -100 0 0
magball/Magn -2.798 0 195.7
magball/Magn 0 1 0
```

```
b =
 magball/Cont
magball/Magn 50
magball/Magn 0
```

```
magball/Magn 0
c =
 magball/Magn magball/Magn magball/Magn
Controller (0 0 -1
d =
 magball/Cont
Controller (0
```

Continuous-time model.

**See Also**

findop, linoptions, operpoint, operspec

# getstatestruct

---

**Purpose** State structure from operating point

**Syntax** `x_struct = getstatestruct(op_point)`

**Description** `x_struct = getstatestruct(op_point)` extracts a structure of state values, `x_struct`, from the operating point object, `op_point`. The structure, `x_struct`, uses the same format as Simulink software which allows you to set initial values for states in the model within the **Data Import/Export** pane of the Configuration Parameters dialog box.

**Example** Create an operating point object for the `magball` model:

```
op_magball=operpoint('magball');
```

Extract a state structure from the operating point object:

```
states_magball=getstatestruct(op_magball)
```

This extraction returns

```
states_magball =

 time: 0
 signals: [1x4 struct]
```

To view the values of the states within this structure, use dot-notation to access the `values` field:

```
states_magball.signals.values
```

This dot-notation returns

```
ans =

 0
 0
```



```
ans =
 7.0036
```

```
ans =
 0
```

```
ans =
 0.0500
```

**See Also** `getinputstruct`, `getxu`, `operpoint`

**Purpose** States and inputs from operating points

**Syntax**

```
x = getxu(op_point)
[x,u] = getxu(op_point)
[x,u,xstruct] = getxu(op_point)
```

**Description** `x = getxu(op_point)` extracts a vector of state values, `x`, from the operating point object, `op_point`. The ordering of states in `x` is the same as that used by Simulink software.

`[x,u] = getxu(op_point)` extracts a vector of state values, `x`, and a vector of input values, `u`, from the operating point object, `op_point`. States in `x` and inputs in `u` are ordered in the same way as for Simulink.

`[x,u,xstruct] = getxu(op_point)` extracts a vector of state values, `x`, a vector of input values, `u`, and a structure of state values, `xstruct`, from the operating point object, `op_point`. The structure of state values, `xstruct`, has the same format as that returned from a Simulink simulation. States in `x` and `xstruct` and inputs in `u` are ordered in the same way as for Simulink.

**Example** Create an operating point object for the `magball` model by typing:

```
op=operpoint('magball');
```

To view the states within this operating point, type:

```
op.States
```

which returns

```
(1.) magball/Controller/Controller
 x: 0
 x: 0
(2.) magball/Magnetic Ball Plant/Current
 x: 7
(3.) magball/Magnetic Ball Plant/dhdt
 x: 0
```

```
(4.) magball/Magnetic Ball Plant/height
x: 0.05
```

To extract a vector of state values, with the states in an ordering that is compatible with Simulink, along with inputs and a state structure, type:

```
[x,u,xstruct]=getxu(op)
```

This syntax returns:

```
x =
 0.0500
 0
 0
 7.0036
 0

u =
 []

xstruct =
 time: 0
 signals: [1x4 struct]
```

View xstruct in more detail by typing:

```
xstruct.signals
```

This syntax displays:

```
1x4 struct array with fields:
 values
 dimensions
 label
 blockname
```

View each component of the structure individually. For example:

```
xstruct.signals(1).values
```

```
ans =
```

```
 0
 0
```

or

```
xstruct.signals(2).values
```

```
ans =
```

```
 7.0036
```

You can import these vectors and structures into Simulink as initial conditions or input vectors or use them with `setxu`, to set state and input values in another operating point.

## See Also

`operpoint`, `operspec`

|                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|----------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>             | Initialize operating point specification values                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>Syntax</b>              | <pre>opnew=initopspec(opspec,oppoint) opnew=initopspec(opspec,x,u) opnew=initopspec(opspec,xstruct,u)</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Graphical Interface</b> | As an alternative to the <code>initopspec</code> function, initialize operating point specification values in the <b>Create Operating Points</b> pane in the <b>Operating Points</b> node within the Simulink Control Design GUI. See “Computing Operating Points from Specifications” on page 2-10.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>Description</b>         | <p><code>opnew=initopspec(opspec,oppoint)</code> initializes the operating point specification object, <code>opspec</code>, with the values contained in the operating point object, <code>oppoint</code>. The function returns a new operating point specification object, <code>opnew</code>. Create <code>opspec</code> with the function <code>operspec</code>. Create <code>oppoint</code> with the function <code>operpoint</code> or <code>findop</code>.</p> <p><code>opnew=initopspec(opspec,x,u)</code> initializes the operating point specification object, <code>opspec</code>, with the values contained in the state vector, <code>x</code>, and the input vector, <code>u</code>. The function returns a new operating point specification object, <code>opnew</code>. Create <code>opspec</code> with the function <code>operspec</code>. You can use the function <code>getxu</code> to create <code>x</code> and <code>u</code> with the correct ordering.</p> <p><code>opnew=initopspec(opspec,xstruct,u)</code> initializes the operating point specification object, <code>opspec</code>, with the values contained in the state structure, <code>xstruct</code>, and the input vector, <code>u</code>. The function returns a new operating point specification object, <code>opnew</code>. Create <code>opspec</code> with the function <code>operspec</code>. You can use the function <code>getstatestruct</code> or <code>getxu</code> to create <code>xstruct</code> and the function <code>getxu</code> to create <code>u</code> with the correct ordering. Alternatively, you can save <code>xstruct</code> to the MATLAB workspace after a simulation of the model. See the Simulink documentation for more information on these structures.</p> |
| <b>Example</b>             | Create an operating point using <code>findop</code> by simulating the <code>magball</code> model and extracting the operating point after 20 time units.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |

```
oppoint=findop('magball',20)
```

This syntax returns the following operating point:

```
Operating Point for the Model magball.
(Time-Varying Components Evaluated at time t=20)
```

```
States:
```

```

```

```
(1.) magball/Controller/Controller
 x: 5.28e-009
 x: -2.56e-006
(2.) magball/Magnetic Ball Plant/Current
 x: 6.99
(3.) magball/Magnetic Ball Plant/dhdt
 x: -2.62e-005
(4.) magball/Magnetic Ball Plant/height
 x: 0.05
```

```
Inputs: None
```

Use these operating point values as initial values in an operating point specification object.

```
opspec=operspec('magball');
newopspec=initopspec(opspec,oppoint)
```

The new operating point specification object is displayed.

```
Operating Specificaton for the Model magball.
(Time-Varying Components Evaluated at time t=0)
```

```
States:
```

```

```

```
(1.) magball/Controller/Controller
 spec: dx = 0, initial guess: 5.28e-009
 spec: dx = 0, initial guess: -2.56e-006
```

```
(1.) magball/Magnetic Ball Plant/Current
 spec: dx = 0, initial guess: 6.99
(1.) magball/Magnetic Ball Plant/dhdt
 spec: dx = 0, initial guess: -2.62e-005
(1.) magball/Magnetic Ball Plant/height
 spec: dx = 0, initial guess: 0.05
```

Inputs: None

Outputs: None

You can now use this object to find operating points by optimization.

**See Also**

findop, getstatestruct, getxu, operpoint, operspec

# linearize

---

**Purpose** Create linearized model from Simulink model

**Syntax**

```
lin=linearize('sys',io)
lin=linearize('sys',op,io)
lin=linearize('sys',op,io,options)
lin_block=linearize('sys',op,'blockname')
lin=linearize('sys',op)
lin=linearize('sys',op,options)
[lin,op] = linearize('sys',snapshottimes);
lin = linearize('sys',blocksubs)
lin = linearize('sys',blocksubs,io)
lin = linearize('sys',blocksubs,io,op)
lin = linearize('sys','StateOrder',stateorder)
```

**Graphical Alternative** As an alternative to the linearize function, create linearized models using the **Linearization Task** node of the Simulink Control Design GUI. See Chapter 4, “Exact Linearization Using the GUI”.

**Description** `lin=linearize('sys',io)` takes a model name, 'sys', and an I/O object, io, as inputs and returns a linear time-invariant state-space model, lin. The linearization I/O object is created with the function `getlinio` or `linio`. io must be associated with the Simulink model, sys. The linearization occurs at the operating point specified in the Simulink model.

`lin=linearize('sys',op,io)` takes a model name, 'sys', an operating point object, op, and an I/O object, io, as inputs and returns a linear time-invariant state-space model, lin. The operating point object is created with the function `operpoint` or `findop`. The linearization I/O object is created with the function `getlinio` or `linio`. Both op and io must be associated with the same Simulink model, sys.

`lin=linearize('sys',op,io,options)` takes a model name, 'sys', an operating point object, op, an I/O object, io, and a linearization options object, options, as inputs. It returns a linear time-invariant state-space model, lin. The operating point object is created with the function `operpoint` or `findop`. The linearization I/O object is



created with the function `getlinio` or `linio`. Both `op` and `io` must be associated with the same Simulink model, `sys`. The linearization options object is created with the function `linoptions` and contains several options for linearization.

`lin_block=linearize('sys',op,'blockname')` takes a model name, `'sys'`, an operating point object, `op`, and the name of a block in the model, `'blockname'`, as inputs and returns `lin_block`, a linear time-invariant state-space model of the named block. The operating point object is created with the function `operpoint` or `findop`. Both `op` and `'blockname'` must be associated with the same Simulink model, `sys`. You can also supply a fourth argument, `options`, to provide options for the linearization. Create options with the function `linoptions`.

`lin=linearize('sys',op)` creates a linearized model, `lin`, of the system `'sys'` at the operating point, `op`. Root-level inport and outport blocks in `sys` are used as inputs and outputs for linearization. The operating point object, `op`, is created with the function `operpoint` or `findop`. You can also supply a third argument, `options`, to provide options for the linearization. Create options with the function `linoptions`.

`lin=linearize('sys',op,options)` is the form of the `linearize` function that is used with numerical-perturbation linearization. The function returns a linear time-invariant state-space model, `lin`, of the entire model, `sys`. The operating point object, `op`, is created with the function `operpoint` or `findop`. The `LinearizationAlgorithm` option must be set to `'numericalpert'` within `options` for numerical-perturbation linearization to be used. Create the variable `options` with the `linoptions` function. The function uses inport and outport blocks in the model as inputs and outputs for linearization.

`[lin,op] = linearize('sys',snapshottimes);` creates operating points for the linearization by simulating the model, `'sys'`, and taking snapshots of the system's states and inputs at the times given in the vector `snapshottimes`. The function returns `lin`, a set of linear time-invariant state-space models evaluated and `op`, the set of operating point objects used in the linearization. You can specify input and output points for linearization by providing an additional argument such as

a linearization I/O object created with `getlinio` or `linio`, or a block name. If an I/O object or block name is not supplied the linearization uses root-level inport and outport blocks in the model. You can also supply an additional argument, `options`, to provide options for the linearization. Create options with the function `linoptions`.

`lin = linearize('sys',blocksubs)` takes a Simulink model named `sys`, and a `nx1` structure `blocksubs`, which specifies the blocks in a model with a desired linearization, and returns a linear state-space model, `lin`. The structure `blocksubs` contains two fields:

- 'Block' is a string specifying the Simulink block to replace.
- 'Value' is a structure with the following fields that specifies the block linearization.
  - `Specification` is a string specifying a MATLAB expression or function for the block's linearization.
  - `Type` is the type of specification, which is either 'Expression' or 'Function'.
  - `ParameterNames` is a comma separated list of the name of parameters to evaluate in the scope of the block. You only specify this field for 'Function' specifications. The evaluated values are available inside the function.
  - `ParameterValues` is a comma separated list containing parameters to evaluate in the scope of the block. You only specify this field for 'Function' specifications.

`lin = linearize('sys',blocksubs,io)` takes a Simulink model named `sys`, a `nx1` structure `blocksubs`, and an I/O object, `io`, as inputs and returns a linear state-space model, `lin`.

`lin = linearize('sys',blocksubs,io,op)` takes a Simulink model named `sys`, a `nx1` structure `blocksubs`, an I/O object, `io`, and operating point `op` as inputs and returns a linear state-space model, `lin`. You can

specify `op` as an operating point you create using `operpoint` or `findop`, or as a vector of simulation times at which to compute `lin`.

`lin = linearize('sys', 'StateOrder', stateorder)` takes the Simulink model 'sys' and creates a linear-time-invariant state-space model, `lin`, whose states are in a specified order. Specify the state order in the cell array `stateorder` by entering the names of the blocks containing states in the model 'sys'. For all blocks, you can enter block names as the full block path. For continuous blocks, you can alternatively enter block names as the user-defined unique state name.

---

**Note** For all syntaxes, `linearize` automatically uses the following properties in the Simulink model:

- `BufferReuse = 'off'`
- `RTWInlineParameters = 'on'`
- `BlockReductionOpt = 'off'`

Simulink restores the original property values after creating the linearized model.

---

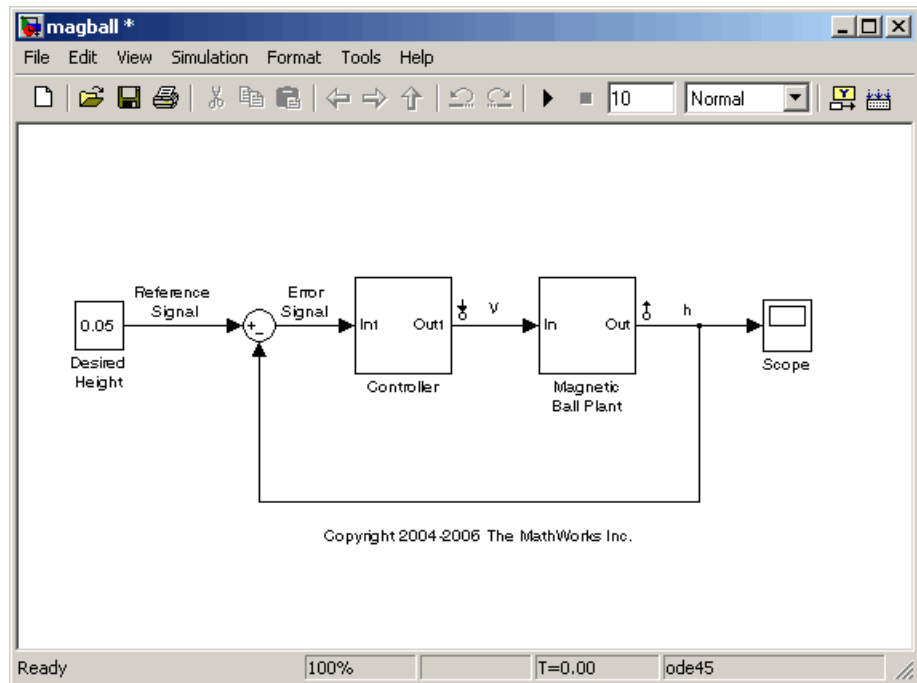
## Algorithms

The function `linoptions` sets the linearization algorithm options and then passes them to the function `linearize` as an optional argument.

## Examples

Open the Simulink model, `magball`, and insert linearization annotations as shown in the following figure.

# linearize



Create an I/O object based on the linearization annotations, create an operating point specification object for the model, and then find the operating point using `findop`.

```
io=getlinio('magball');
op=operspec('magball');
op=findop('magball',op);
```

Compute a linear model of the `magball` system, based on the linearization I/Os, `io`, and defined about the operating point, `op`, with the command

```
lin=linearize('magball',op,io)
```

which returns

```

a =
 Controller Current dhdt height
Controller 0 0 0 -1
Current -50 -100 0 0
dhdt 0 -2.801 0 196.2
height 0 0 1 0

```

```

b =
 magball/Cont
Controller 0
Current 50
dhdt 0
height 0

```

```

c =
 Controller Current dhdt height
Magnetic Bal 0 0 0 1

```

```

d =
 magball/Cont
Magnetic Bal 0

```

Continuous-time model.

The matrices, a, b, c, and d are the state-space matrices of the linear system given by the following equations:

$$\dot{x}(t) = ax(t) + bu(t)$$

$$y(t) = cx(t) + du(t)$$

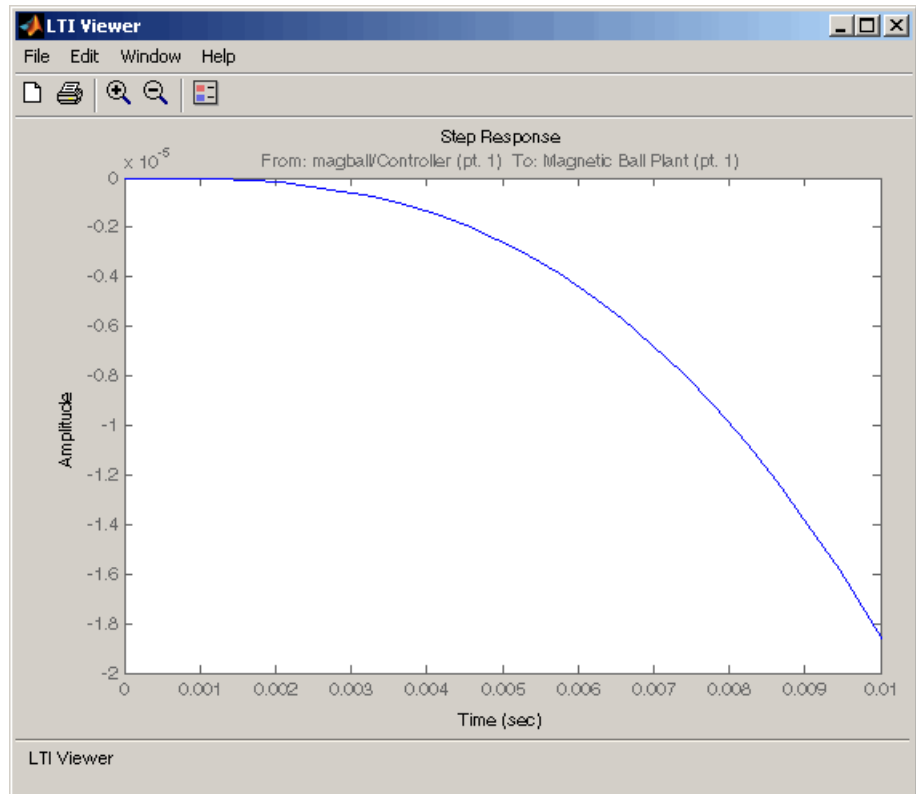
where  $x(t)$  is a vector of states and  $u(t)$  is a vector of inputs to the system.

You can view the linearized model, `lin`, with the LTI Viewer, by typing:

```
ltiview(lin)
```

# linearize

which produces the following plot.



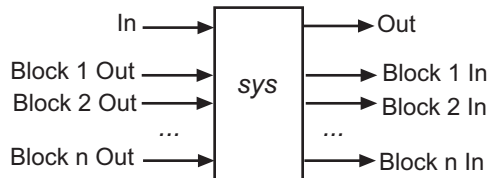
## See Also

`findop`, `getlinio`, `operpoint`, `operspec`, `linio`, `linoptions`, `ltiview`

**Purpose** Linearize a model while removing the contribution of specified blocks

**Syntax** `lin_fixed = linlft('sys',io,blocks)`

**Description** `lin_fixed = linlft('sys',io,blocks)` linearizes the Simulink model named `sys` while removing the contribution of certain blocks. Specify the full block pathnames of the blocks to ignore in the cell array of strings called `blocks`. The linearization occurs at the operating point specified in the Simulink model, which includes the ignored blocks. You can optionally specify linearization points in the I/O object `io`. The resulting linear model `lin_fixed` has this form:



The top channels `In` and `Out` correspond to the linearization points you specify in the I/O object `io`. The remaining channels correspond to the connection to the ignored blocks.

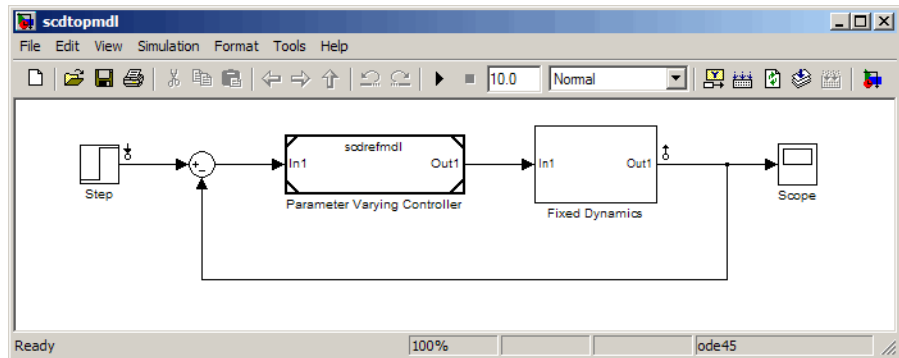
When you use `linlft` and specify the 'block-by-block' linearization algorithm in `linoptions`, you can use all the variations of the input arguments for `linearize`.

You can linearize the ignored blocks separately using `linearize`, and then combine the linearization results using `linlftfold`.

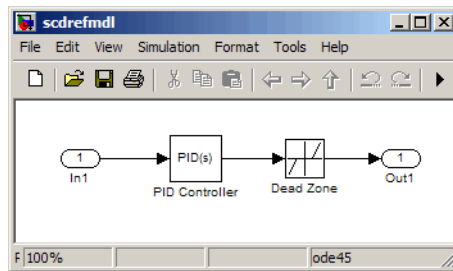
## Examples

Linearize the following parts of the `scdtopmdl` Simulink model separately, and then combine the results:

- Fixed portion, which contains everything except the Parameter Varying Controller model reference



- Parameter Varying Controller model reference, which references the scdrefmdl model



```
% Open the Simulink model
topmdl = 'scdtopmdl';
```

```
% Linearize the model without the Parameter Varying Controller
io = getlinio(topmdl);
blocks = {'scdtopmdl/Parameter Varying Controller'};
sys_fixed = linlft(topmdl,io,blocks);
```

```
% Linearize the Parameter Varying Controller
refmdl = 'scdrefmdl';
sys_pv = linearize(refmdl);
```

```
% Combine the results
```



```
BlockSubs(1) = struct('Name',blocks{1},'Value',sys_pv);
sys_fold = linlftfold(sys_fixed,BlockSubs);
```

**See Also**

[linlftfold](#) | [linearize](#) | [linio](#) | [getlinio](#) | [operpoint](#)

# linlftfold

**Purpose** Combine linearization results from specified blocks and a model

**Syntax** `lin = linlftfold(lin_fixed,blocksubs)`

**Description** `lin = linlftfold(lin_fixed,blocksubs)` combines the following linearization results into one linear model `lin`:

- Linear model `lin_fixed`, which does not include the contribution of specified blocks in your Simulink model

You compute `lin_fixed` using `linlft`.

- Block linearizations for the blocks excluded from `lin_fixed`

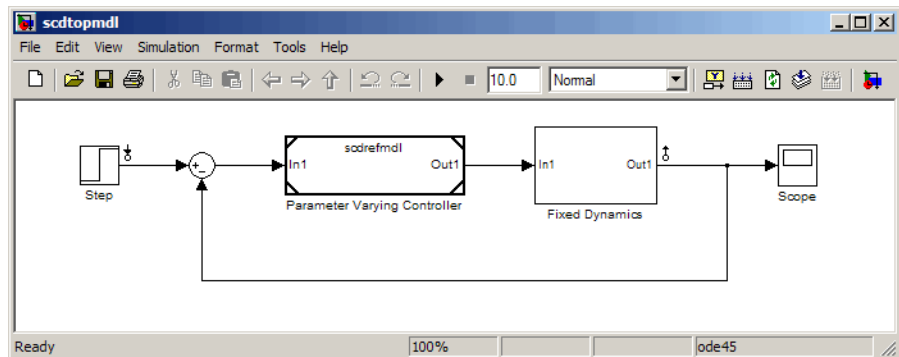
You specify the block linearizations in a structure array `blocksubs`, which contains two fields:

- 'Block' is a string specifying the Simulink block to replace.
- 'Value' is the value of the linearization for each block.

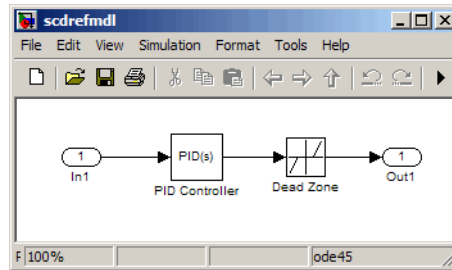
## Examples

Linearize the following parts of the `scdtopmdl` Simulink model separately and then combine the results:

- Fixed portion, which contains everything except the Parameter Varying Controller model reference



- Parameter Varying Controller model reference, which references the scdrefmdl model



```
% Open the Simulink model
topmdl = 'scdtopmdl';

% Linearize the model without the Parameter Varying Controller
io = getlinio(topmdl);
blocks = {'scdtopmdl/Parameter Varying Controller'};
sys_fixed = linlft(topmdl,io,blocks);

% Linearize the Parameter Varying Controller
refmdl = 'scdrefmdl';
sys_pv = linearize(refmdl);

% Combine the results
BlockSubs(1) = struct('Name',blocks{1},'Value',sys_pv);
sys_fold = linlftfold(sys_fixed,BlockSubs);
```

## See Also

linlft | linearize | linio | getlinio | operpoint

**Purpose** Construct linearization I/O settings for Simulink model

**Syntax**

```
io=linio('blockname',portnum)
io=linio('blockname',portnum,type)
io=linio('blockname',portnum,type,openloop)
```

**Graphical Alternative** As an alternative to the `linio` function, create linearization I/O settings by using the right-click menu on the model diagram. See “Inserting Linearization Points” on page 4-46.

**Description** `io=linio('blockname',portnum)` creates a linearization I/O object for the signal that originates from the output with port number, `portnum`, of the block, `'blockname'`, in a Simulink model. The default I/O type is `'in'`, and the default `OpenLoop` property is `'off'`. Use `io` with the function `linearize` to create linearized models.

`io=linio('blockname',portnum,type)` creates a linearization I/O object for the signal that originates from the output with port number, `portnum`, of the block, `'blockname'`, in a Simulink model. The linearization I/O has the type given by `type`. A list of available types is given below. The default `OpenLoop` property is `'off'`. Use `io` with the function `linearize` to create linearized models.

`io=linio('blockname',portnum,type,openloop)` creates a linearization I/O object for the signal that originates from the output with port number, `portnum`, of the block, `'blockname'`, in a Simulink model. The linearization I/O has the type given by `type` and the open-loop status is given by `openloop`. A list of available types is given below. The `openloop` property is set to `'off'` when the I/O is not an open-loop point and is set to `'on'` when the I/O is an open-loop point. Use `io` with the function `linearize` to create linearized models.

Available linearization I/O types are:

- `'in'`, linearization input point
- `'out'`, linearization output point
- `'inout'`, linearization input then output point

- 'outin', linearization output then input point
- 'none', no linearization input/output point

To upload the settings in the I/O object to the Simulink model, use the `setlinio` function.

## Example

Create a linearization I/O setting for the signal line originating from the Controller block of the `magball` model.

```
io(1)=linio('magball/Controller',1)
```

This syntax displays:

```

 Linearization IOs:

 Block magball/Controller, Port 1 is marked with the following
 properties:
 - No Loop Opening
 - An Input Perturbation

```

By default, this I/O is an input point. Create a second I/O setting within the object, `io`. This I/O originates from the Magnetic Ball Plant block, is an output point and is also an open-loop point.

```
io(2)=linio('magball/Magnetic Ball Plant',1,'out','on')
```

The new object, `io`, is displayed as follows:

```

 Linearization IOs:

 Block magball/Controller, Port 1 is marked with the following
 properties:
 - No Loop Opening
 - An Input Perturbation

 Block magball/Magnetic Ball Plant, Port 1 is marked with the
 following properties:

```

# linio

---

- An Output Measurement
- A Loop Opening

**See Also**      `getlinio`, `linearize`, `setlinio`

## Purpose

Set options for linearization and finding operating points

## Syntax

```
opt=linoptions
opt=linoptions('Property1','Value1','Property2','Value2',
...)
```

## Graphical Interface

As an alternative to the `linoptions` function, set options for linearization and finding operating points with the Simulink Control Design GUI.

## Description

`opt=linoptions` creates a linearization options object with the default settings. The variable, `opt`, is passed to the functions `findop` and `linearize` to specify options for finding operating points and linearization.

`opt=linoptions('Property1','Value1','Property2','Value2',...)` creates a linearization options object, `opt`, in which the option given by `Property1` is set to the value given in `Value1`, the option given by `Property2` is set to the value given in `Value2`, etc. The variable, `opt`, is passed to the functions `findop` and `linearize` to specify options for finding operating points and linearization.

The following options can be set with `linoptions`:

|                                     |                                                                                                                                                                                                                                                                                            |
|-------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>LinearizationAlgorithm</code> | Set to 'numericalpert' (default is 'blockbyblock') to enable numerical-perturbation linearization (as in Simulink 3.0 software) where root-level inports and states are numerically perturbed. Linearization annotations are ignored and root-level inports and outports are used instead. |
| <code>SampleTime</code>             | The time at which the signal is sampled. Nonzero for discrete systems, 0 for continuous systems, -1 (default) to use the longest sample time that contributes to the linearized model.                                                                                                     |
| <code>UseFullBlockNameLabels</code> | Set to 'off' (default) to use truncated names for the linearization I/Os and states in the linearized model. Set to 'on' to use the full block path to name the linearization I/Os and states in the linearized models.                                                                    |

## UseBusSignalLabels

Set to 'off' (default) to use bus signal channel number to label I/Os on bus signals in your linearization results. Set to 'on' to use bus signal names to label I/Os on bus signals in your linearization results. Bus signal names appear in the results when the I/O points are located at the output of the following blocks:

- Root-level inport block containing a bus object
- Bus creator block
- Subsystem block whose source traces back to one of the following:
  - Output of a bus creator block
  - Root-level inport by passing through only virtual or nonvirtual subsystem boundaries

---

**Note** You cannot use this option when your model has mux/bus mixtures. For information on how to avoid buses used as muxes, see “Avoiding Mux/Bus Mixtures” in the Simulink documentation.

---

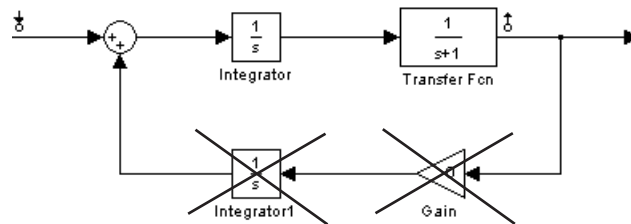
## BlockReduction

Set to 'on' (default) to eliminate from the linearized model those blocks that are not in the path of the linearization. Block reduction eliminates the states of blocks in dead linearization paths from your linearization results. Some examples of dead linearization paths are linearization paths that include:

- Blocks that linearize to zero
- Switch blocks that are not active along the path
- Disabled subsystems
- Signals marked as open-loop linearization points



The linearization result of the model shown in the following figure includes only two states. It does not include states from the two blocks outside the linearization path. These states do not appear because these blocks are on a dead linearization path with a block that linearizes to zero (the zero gain block).



Set to 'off' to return a linearized model that includes all of the states of the model.

IgnoreDiscreteStates

Set to 'on' when performing continuous linearization (SampleTime set to 0) to remove any discrete states from the linearization and accept the D value for all blocks with discrete states. Set to 'off' (default) to include discrete states.

RateConversionMethod

When you linearize a multirate system, set this option to one of the following rate conversion methods:

- 'zoh' (default) to use the zero order rate conversion method
- 'tustin' to use the Tustin (bilinear) method
- 'prewarp' to use the Tustin approximation with prewarping
- 'upsampling\_zoh' to upsample discrete states when possible and to use 'zoh' otherwise
- 'upsampling\_tustin' to upsample discrete states when possible and to use 'tustin' otherwise

- 'upsampling\_prewarp' to upsample discrete states when possible and to use 'prewarp' otherwise

---

**Note** When you select 'prewarp' or 'upsampling\_prewarp', set the PreWarpFreq option to the desired prewarp frequency.

---

---

**Note** You can only upsample when you convert discrete states to a new sample time that is an *integer-value-times faster* than the sampling time of the original system.

---

For more information, and examples, on methods and algorithms for rate conversions and linearization of multirate models, see:

- “Linearization of Multi-Rate Models” and “Rate Conversion Method Selection for Linearization” demos listed under the Simulink Control Design Demos in the demos browser.
- “Continuous/Discrete Conversions of LTI Models” and “Resampling of Discrete-Time Models” in the Control System Toolbox documentation.

PreWarpFreq

The critical frequency  $W_c$  (in rad/sec) used by the 'prewarp' option when linearizing a multirate system.

UseExactDelayModel

Set to 'on' to return a linear model with an exact delay representation. Set to 'off' (default) to return a model with approximate delays. For more information, see “Linearizing Models with Time Delays” on page 4-15.

|                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| NumericalPertRel    | <p>Set the perturbation level for obtaining the linear model (default value is <math>1e-5</math>). The perturbation of the system's states is specified by:</p> $\text{NumericalPertRel} + 10^{-3} \times \text{NumericalPertRel} \times  x $ <p>The perturbation of the system's inputs is specified by:</p> $\text{NumericalPertRel} + 10^{-3} \times \text{NumericalPertRel} \times  u $                                                                                                                                                                                                                                                               |
| NumericalXPert      | <p>Individually set the perturbation levels for the system's states using an operating point object. Use the <code>operpoint</code> function to create an operating point object for the model.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| NumericalUPert      | <p>Individually set the perturbation levels for the system's inputs using an operating point object. Use the <code>operpoint</code> function to create an operating point object for the model.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| OptimizationOptions | <p>Set options for use with the optimization algorithms. These options are the same as those set with <code>optimset</code>. See the Optimization Toolbox documentation for more information on these algorithms. If you do not have the Optimization Toolbox software, you can access the documentation at:</p> <p style="text-align: center;"><a href="http://www.mathworks.com/access/helpdesk/help/toolbox/optim/optim.shtml">http://www.mathworks.com/access/helpdesk/help/toolbox/optim/optim.shtml</a></p>                                                                                                                                         |
| OptimizerType       | <p>Set optimizer type to be used by trim optimization if the Optimization Toolbox software is installed. The available optimizer types are:</p> <ul style="list-style-type: none"> <li>• <code>graddescent_elim</code>, the default optimizer, enforces an equality constraint to force the time derivatives of states to be zero (<math>dx/dt=0</math>, <math>x(k+1)=x(k)</math>) and the output signals to be equal to their specified 'Known' value. The optimizer fixes the states, <math>x</math>, and inputs, <math>u</math>, that are marked as 'Known' in an operating point specification and then optimizes the remaining variables.</li> </ul> |

- `graddescent`, enforces an equality constraint to force the time derivatives of states to be zero ( $dx/dt=0$ ,  $x(k+1)=x(k)$ ) and the output signals to be equal to their specified 'Known' value. `findop` also minimizes the error between the states,  $x$ , and inputs,  $u$ , that are marked as 'Known' in an operating point specification. If there are not any inputs or states marked as 'Known', `findop` attempts to minimize the deviation between the initial guesses for  $x$  and  $u$  and their trimmed values.
- `lsqnonlin` fixes the states,  $x$ , and inputs,  $u$ , that are marked as 'Known' in an operating point specification and optimizes the remaining variables. The algorithm then tries to minimize both the error in the time derivatives of the states ( $dx/dt=0$ ,  $x(k+1)=x(k)$ ) and the error between the outputs and their specified 'Known' value.
- `simplex` uses the same cost function as `lsqnonlin` with the direct search optimization routine found in `fminsearch`.

See the Optimization Toolbox documentation for more information on these algorithms. If you do not have the Optimization Toolbox software, you can access the documentation at <http://www.mathworks.com/support/>.

`DisplayReport`

Set to 'on' to display the operating point summary report when running `findop`. Set to 'off' to suppress the display of this report.

## See Also

`findop`, `linearize`

**Purpose** Create operating point for Simulink model

**Syntax** `op = operpoint('sys')`

**Graphical Interface** As an alternative to the `operpoint` function, create operating points in the **Operating Points** node of the Simulink Control Design GUI. See “Creating Operating Points” on page 2-10.

**Description** `op = operpoint('sys')` returns an object, `op`, containing the operating point of a Simulink model, `sys`. Use the object with the function `linearize` to create linearized models. The operating point object properties are:

- “Model” on page 9-87
- “States” on page 9-87
- “Inputs” on page 9-88
- “Time” on page 9-88

Edit the properties of this object directly or with the `set` function.

### **Model**

`Model` specifies the name of the Simulink model that this operating point object refers to.

### **States**

`States` describes the operating points of states in the Simulink model. The `States` property is a vector of state objects that contains the operating point values of the states. There is one state object per block that has a state in the Simulink model. The `States` object has the following properties:

|                    |                                                            |
|--------------------|------------------------------------------------------------|
| <code>Nx</code>    | Number of states in the block. This property is read-only. |
| <code>Block</code> | Block with which the states are associated.                |

|                   |                                                                                         |
|-------------------|-----------------------------------------------------------------------------------------|
| x                 | Vector containing the values of states in the block.                                    |
| Ts                | Vector containing the sample time and offset for the state.                             |
| SampleType        | Set this value to CSTATE, for a continuous state, or DSTATE for a discrete state.       |
| inReferencedModel | Set this value to 1, when the state is inside a referenced model, or 0, when it is not. |
| Description       | Text string describing the block.                                                       |

## Inputs

Inputs is a vector of input objects that contains the input levels at the operating point. There is one input object per root-level inport block in the Simulink model. The Inputs object has the following properties:

|             |                                                          |
|-------------|----------------------------------------------------------|
| Block       | Inport block with which the input vector is associated   |
| PortWidth   | Width of the corresponding inport                        |
| u           | Vector containing the input level at the operating point |
| Description | Text string describing the input                         |

## Time

Time specifies the time at which any time-varying functions in the model are evaluated.

## Example

To create an operating point object for the Simulink model magball, type:

```
op = operpoint('magball')
```

which returns the following:

```
Operating Point for the Model magball.
(Time-Varying Components Evaluated at time t=0)
```

```
States:
```

```

```

```
(1.) magball/Controller/Controller
 x: 0
 x: 0
(2.) magball/Magnetic Ball Plant/Current
 x: 7
(3.) magball/Magnetic Ball Plant/dhdt
 x: 0
(4.) magball/Magnetic Ball Plant/height
 x: 0.05
```

```
Inputs: None
```

MATLAB software displays the name of the model, the time at which any time-varying functions in the model are evaluated, the names of blocks containing states, and the values of the states at the operating point. In this example there are four blocks that contain states in the model and four entries in the `States` object. The first entry contains two states. MATLAB also displays the `Inputs` although there are not any in this model. To view the properties of `op` in more detail, use the `get` function.

## See Also

`get`, `linearize`, `operspec`, `set`, `update`

**Purpose** Create operating point specifications for Simulink model

**Syntax** `op_spec = operspec('sys')`

**Graphical Alternative** As an alternative to the `operspec` function, create operating point specifications in the **Operating Points** node of the Simulink Control Design GUI. See “Computing Operating Points from Specifications” on page 2-10.

**Description** `op_spec = operspec('sys')` returns an operating point specification object, `op`, for a Simulink model, `sys`. Edit the default operating point specifications directly or use `get` and `set`. Use the operating point specifications object with the function `findop` to find operating points based on the specifications. Use these operating points with the function `linearize` to create linearized models.

The operating point specification object properties are:

- “Model” on page 9-90
- “States” on page 9-90
- “Inputs” on page 9-92
- “Time” on page 9-92
- “Outputs” on page 9-93

Use the `set` function to edit the properties of this object before running `findop`.

## **Model**

`Model` is the name of the Simulink model with which this operating point specification object is associated.

## **States**

`States` describes the operating point specifications for states in the Simulink model. The `States` property is a vector of state objects that each contain specifications for particular states. There is one state



specification object per block that has a state in the model. The States object has the following properties:

|                   |                                                                                                                                                                                                                                                                                                                                 |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Block             | Block with which the states are associated.                                                                                                                                                                                                                                                                                     |
| x                 | Vector containing values of states in the block. Set the corresponding value of Known to 1 for values that are known operating point values. Set the corresponding value of Known to 0 for values that are initial guesses for the operating point values. The default value of x is the initial condition value for the state. |
| Nx                | Number of states in the block. This property is read-only.                                                                                                                                                                                                                                                                      |
| Ts                | Vector containing the sample time and offset for the state.                                                                                                                                                                                                                                                                     |
| SampleType        | Set this value to CSTATE, for a continuous state, or DSTATE, for a discrete state.                                                                                                                                                                                                                                              |
| inReferencedModel | Set this value to 1, when the state is inside a referenced model, or 0, when it is not                                                                                                                                                                                                                                          |
| Known             | Vector of values set to 1, for states whose operating points are known exactly, and set to 0, for states whose operating points are not known exactly. Set the operating point values in the x property.                                                                                                                        |
| SteadyState       | Vector of values set to 1, for states whose operating points should be at equilibrium, and set to 0 for states whose operating points are not at equilibrium. The default value is 1.                                                                                                                                           |
| Min               | Vector containing the minimum values of the corresponding state's operating point.                                                                                                                                                                                                                                              |

|             |                                                                                    |
|-------------|------------------------------------------------------------------------------------|
| Max         | Vector containing the maximum values of the corresponding state's operating point. |
| Description | Text string describing the block.                                                  |

## Inputs

`Inputs` is a vector of input specification objects that contains specifications for the input levels at the operating point. There is one input specification object per root-level inport block in the Simulink model. The `Inputs` object has the following properties:

|             |                                                                                                                                                                                                                                                                           |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Block       | The inport block with which the input vector is associated.                                                                                                                                                                                                               |
| PortWidth   | Width of the corresponding inport.                                                                                                                                                                                                                                        |
| u           | Vector containing values of inputs. Set the corresponding value of <code>Known</code> to 1, for values that are known operating point values. Set the corresponding value of <code>Known</code> to 0, for values that are initial guesses for the operating point values. |
| Known       | Vector of values set to 1, for inputs whose operating points are known exactly, and set to 0, for inputs whose operating points are not known exactly. Set the operating point values in the <code>u</code> property.                                                     |
| Min         | Vector containing the minimum values of the corresponding input's operating point.                                                                                                                                                                                        |
| Max         | Vector containing the maximum values of the corresponding input's operating point.                                                                                                                                                                                        |
| Description | Text string describing the input.                                                                                                                                                                                                                                         |

## Time

`Time` specifies the time at which any time-varying functions in the model are evaluated.

## Outputs

`Outputs` is a vector of output specification objects that contains the specifications for the output levels at the operating point. There is one output specification object per root-level output block in the Simulink model. To constrain additional outputs, use the `addoutputspec` function to add another output specification to the operating point specification object. The `Outputs` object has the following properties:

|                          |                                                                                                                                                                                                                                                                           |
|--------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>Block</code>       | Output port block with which the output vector is associated.                                                                                                                                                                                                             |
| <code>PortWidth</code>   | Width of the corresponding output.                                                                                                                                                                                                                                        |
| <code>PortNumber</code>  | Port number with which the output is associated.                                                                                                                                                                                                                          |
| <code>y</code>           | Vector containing values of outputs. Set the corresponding value of <code>Known</code> to 1, for values that are known operating point values. Set the corresponding value of <code>Known</code> to 0 for values that are initial guesses for the operating point values. |
| <code>Known</code>       | Vector of values set to 1, for outputs whose operating points are known exactly, and set to 0, for outputs whose operating points are not known exactly. Set the operating point values in the <code>y</code> property.                                                   |
| <code>Min</code>         | Vector containing the minimum values of the corresponding output's operating point.                                                                                                                                                                                       |
| <code>Max</code>         | Vector containing the maximum values of the corresponding output's operating point.                                                                                                                                                                                       |
| <code>Description</code> | Text string describing the output.                                                                                                                                                                                                                                        |

## Example

To create an operating point specification object for the Simulink model `magball`, type:

```
op_spec = operspec('magball')
```

which returns the following:

Operating Specificaton for the Model magball.  
(Time-Varying Components Evaluated at time t=0)

States:

```

(1.) magball/Controller/Controller
 spec: dx = 0, initial guess: 0
 spec: dx = 0, initial guess: 0
(2.) magball/Magnetic Ball Plant/Current
 spec: dx = 0, initial guess: 7
(3.) magball/Magnetic Ball Plant/dhdt
 spec: dx = 0, initial guess: 0
(4.) magball/Magnetic Ball Plant/height
 spec: dx = 0, initial guess: 0.05
```

Inputs: None

Outputs: None

The following is displayed:

- the name of the model
- the time at which any time-varying functions in the model are evaluated
- the names of blocks containing states
- default operating point values and initial guesses (based on initial conditions of the states)
- steady-state specifications

In this example, there are four blocks that contain states in the model and four entries in the `States` object. The first entry contains two states. By default, MATLAB software sets the `SteadyState` property to 1 and the upper and lower bounds on the operating points to `Inf` and `-Inf` respectively. MATLAB also displays the `Inputs` and `Outputs`,

although there are not any in this model. To view the properties of `op` in more detail, use the `get` function.

**See Also**

`addoutputspec`, `findop`, `get`, `operspec`, `linearize`, `set` , `update`

# set

---

**Purpose** Set properties of linearization I/Os and operating points

**Syntax**

```
set(ob)
set(ob, 'PropertyName', val)
ob.PropertyName=val
```

**Graphical Interface** As an alternative to the `set` function, set properties of linearization I/Os and operating points in the Simulink Control Design GUI. See “Inspecting Analysis I/Os” on page 4-52 and “Creating Operating Points” on page 2-10.

**Description** `set(ob)` displays all editable properties of the object, `ob`, which can be a linearization I/O object, an operating point object, or an operating point specification object. Create `ob` using `findop`, `getlinio`, `linio`, `operpoint`, or `operspec`.

`set(ob, 'PropertyName', val)` sets the property, `PropertyName`, of the object, `ob`, to the value, `val`. The object, `ob`, can be a linearization I/O object, an operating point object, or an operating point specification object. Create `ob` using `findop`, `getlinio`, `linio`, `operpoint`, or `operspec`.

`ob.PropertyName=val` is an alternative notation for assigning the value, `val`, to the property, `PropertyName`, of the object, `ob`. The object, `ob`, can be a linearization I/O object, an operating point object, or an operating point specification object. Create `ob` using `findop`, `getlinio`, `linio`, `operpoint`, or `operspec`.

**Examples** Create an operating point object for the Simulink model, `magball`:

```
op_cond=operpoint('magball');
```

Use the `set` function to get a list of all editable properties of this object:

```
set(op_cond)
```

This function returns the properties of `op_cond`.

```
ans =
 Model: {}
 States: {}
 Inputs: {}
 Time: {}
```

To set the value of a particular property of `op_cond`, provide the property name and the desired value of this property as arguments to `set`. For example, to change the name of the model associated with the operating point object from 'magball' to 'Magnetic Ball', type:

```
set(op_cond, 'Model', 'Magnetic Ball')
```

To view the property value and verify that the change was made, type:

```
op_cond.Model
```

which returns

```
ans =
Magnetic Ball
```

Because `op_cond` is a structure, you can set any properties or fields using dot-notation. First, produce a list of properties of the second `States` object within `op_cond`, as follows:

```
set(op_cond.States(2))
ans =
 Nx: {}
 Block: {}
 x: {}
 Ts: {}
 SampleType: {}
 inReferencedModel: {}
 Description: {}
```

Now, use dot-notation to set the `x` property to 8:

## set

---

```
op_cond.States(2).x=8;
```

To view the property and verify that the change was made, type

```
op_cond.States(2)
```

which displays

```
(1.) magball/Magnetic Ball Plant/Current
x: 8
```

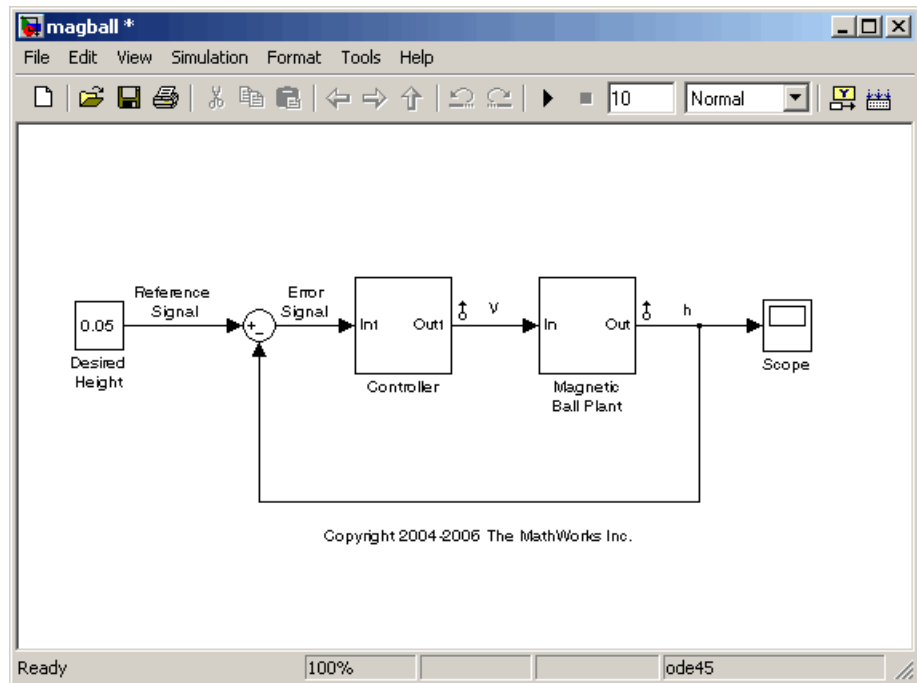
### **See Also**

findop, get, linio, operpoint, operspec, setlinio



|                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|----------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>             | Assign I/O settings to Simulink model                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>Syntax</b>              | <code>oldio=setlinio('sys',io)</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>Graphical Interface</b> | As an alternative to the <code>setlinio</code> function, edit linearization I/Os in the <b>Analysis I/Os</b> pane of the <b>Linearization Task</b> node within the Simulink Control Design GUI. See “Inspecting Analysis I/Os” on page 4-52.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>Description</b>         | <code>oldio=setlinio('sys',io)</code> assigns the settings in the vector of linearization I/O objects, <code>io</code> , to the Simulink model, <code>sys</code> . These settings appear as annotations on the signal lines. Use the function <code>getlinio</code> or <code>linio</code> to create the linearization I/O objects. You can save I/O objects to disk in a MAT-file and use them later to restore linearization settings in a model.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>Examples</b>            | <p>Before assigning I/O settings to a Simulink model using <code>setlinio</code>, you must create a vector of I/O objects representing linearization annotations, such as input points or output points, on a Simulink model.</p> <p>Open the Simulink model <code>magball</code> by typing:</p> <pre>magball</pre> <p>at the MATLAB prompt. Right-click the signal line between the Magnetic Ball Plant and the Controller. Select <b>Linearization Points &gt; Output Point</b> from the menu to place an output point on this signal line. Notice a small arrow pointing away from a small circle just above the signal line. This arrow represents the output point.</p> <p>Right-click the signal line after the Magnetic Ball Plant. Select <b>Linearization Points &gt; Output Point</b> from the menu to place another output point on this signal line. The model diagram should now look similar to that in the following figure:</p> |

# setlinio



Create an I/O object with the `getlinio` function:

```
io=getlinio('magball')
```

Make changes to `io` by editing the object or by using the `set` function.  
For example:

```
io(1).Type='in';
io(2).OpenLoop='on';
```

Assign the new settings in `io` to the model diagram:

```
oldio=setlinio('magball',io)
```

This assignment returns the old I/O settings (that have been replaced by the settings in `io`).

Linearization IOs:

-----

Block magball/Controller, Port 1 is marked with the following properties:

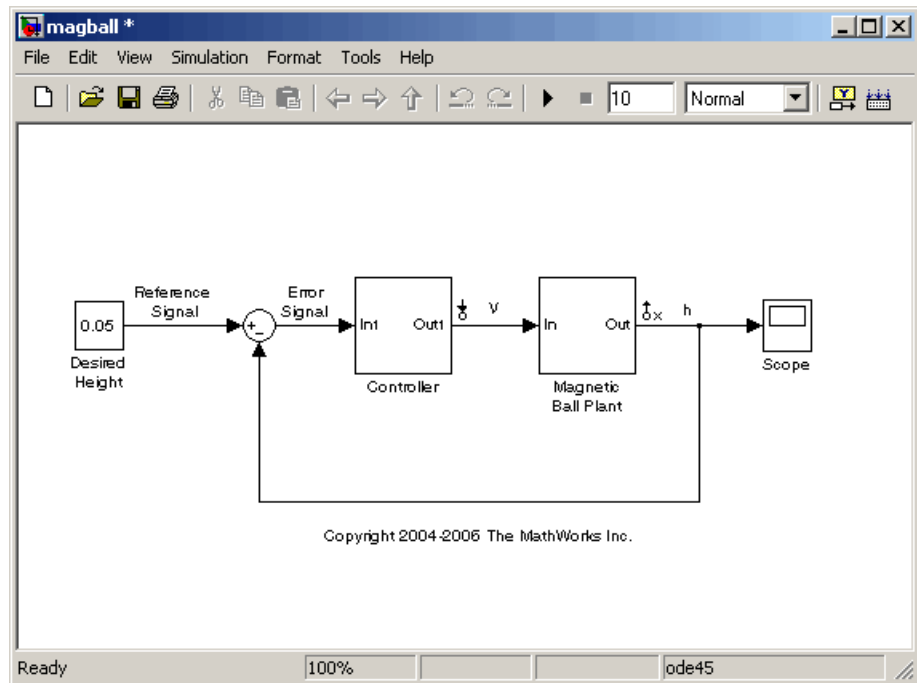
- An Output Measurement
- No Loop Opening
- No signal name. Linearization will use the block name

Block magball/Magnetic Ball Plant, Port 1 is marked with the following properties:

- An Output Measurement
- No Loop Opening
- No signal name. Linearization will use the block name

The model diagram should now look similar to that in the following figure:

# setlinio



**See Also** `get`, `getlinio`, `linio`, `set`

|                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>               | Set states and inputs in operating points                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>Syntax</b>                | <code>op_new=setxu(op_point,x,u)</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>Graphical Alternative</b> | As an alternative to the <code>setxu</code> function, set states and inputs of operating points with the Simulink Control Design GUI. See “Importing Operating Points” on page 2-25 for more information.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>Description</b>           | <code>op_new=setxu(op_point,x,u)</code> sets the states and inputs in the operating point, <code>op_point</code> , with the values in <code>x</code> and <code>u</code> . A new operating point containing these values, <code>op_new</code> , is returned. The variable <code>x</code> can be a vector or a structure with the same format as those returned from a Simulink simulation. The variable <code>u</code> can be a vector. Both <code>x</code> and <code>u</code> can be extracted from another operating point object with the <code>getxu</code> function.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>Example</b>               | <p>Open the Simulink model F14 by typing <code>f14</code> at the command line. Select <b>Simulation &gt; Configuration Parameters &gt; Data Import/Export</b>. In the <b>Save to workspace</b> pane, select <b>Final states</b>. In the <b>Save options</b> pane, select <b>Structure</b> from <b>Format</b>. This selection saves the final states of the model to the workspace after a simulation.</p> <p>Start the simulation. After it has run, a new variable, <code>xFinal</code>, should be in the workspace. This variable is a structure with two properties, <code>time</code> and <code>signals</code>.</p> <p>Create an operating point object for F14 by typing:</p> <pre>op_point=operpoint('f14')</pre> <p>All states are initially set to 0. Set the states in this object to be the values in <code>xFinal</code>. Set the input to be 9.</p> <pre>newop=setxu(op_point,xFinal,9)</pre> <p>The new operating point is displayed as follows:</p> <pre>Operating Point for the Model f14. (Time-Varying Components Evaluated at time t=0)</pre> |

States:

- 
- (1.) f14/Actuator Model  
x: -0.032
  - (2.) f14/Aircraft Dynamics Model/Transfer Fcn.1  
x: 0.56
  - (3.) f14/Aircraft Dynamics Model/Transfer Fcn.2  
x: 678
  - (4.) f14/Controller/Alpha-sensor Low-pass Filter  
x: 0.392
  - (5.) f14/Controller/Pitch Rate Lead Filter  
x: 0.133
  - (6.) f14/Controller/Proportional plus integral compensator  
x: 0.166
  - (7.) f14/Controller/Stick Prefilter  
x: 0.1
  - (8.) f14/Dryden Wind Gust Models/Q-gust model  
x: 0.114
  - (9.) f14/Dryden Wind Gust Models/W-gust model  
x: 0.46  
x: -2.05

Inputs:

- 
- (1.) f14/u  
u: 9

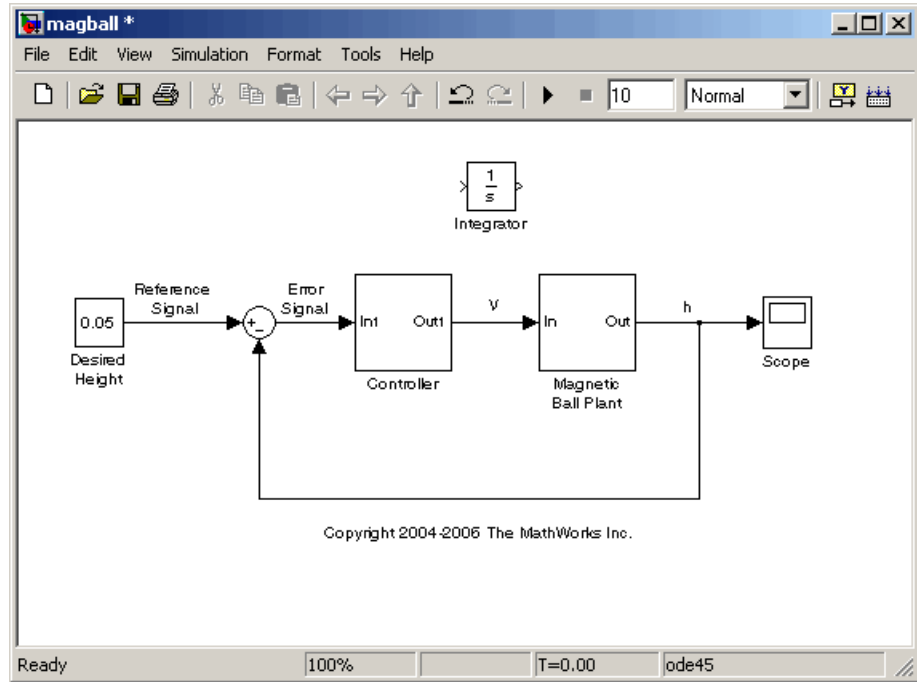
**See Also**

getxu, initopspec, operpoint, operspec

|                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>               | Update operating point object with structural changes in model                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>Syntax</b>                | <code>update(op)</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>Graphical Alternative</b> | As an alternative to the <code>update</code> function, update operating point objects using the <b>Sync with Model</b> button in the Simulink Control Design GUI. See Chapter 2, “Operating Point Analysis Using the GUI”.                                                                                                                                                                                                                                                                                                                                                           |
| <b>Description</b>           | <code>update(op)</code> updates an operating point object, <code>op</code> , to reflect any changes in the associated Simulink model, such as states being added or removed.                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>Example</b>               | <p>Open the <code>magball</code> model:</p> <pre>magball</pre> <p>Create an operating point object for the model:</p> <pre>op=operpoint('magball')</pre> <p>This syntax returns:</p> <pre>Operating Point for the Model magball.<br/>(Time-Varying Components Evaluated at time t=0)<br/><br/>States:<br/>-----<br/>(1.) magball/Controller/Controller<br/>      x: 0<br/>(2.) magball/Magnetic Ball Plant/Current<br/>      x: 7<br/>(3.) magball/Magnetic Ball Plant/dhdt<br/>      x: 0<br/>(4.) magball/Magnetic Ball Plant/height<br/>      x: 0.05<br/><br/>Inputs: None</pre> |

# update

Add an Integrator block to the model, as shown in the following figure.



Update the operating point to include this new state:

```
update(op)
```

The new operating point appears:

```
Operating Point for the Model magball.
(Time-Varying Components Evaluated at time t=0)
```

States:

-----

```
(1.) magball/Controller/Controller
```



```
 x: 0
(2.) magball/Magnetic Ball Plant/Current
 x: 7
(3.) magball/Magnetic Ball Plant/dhdt
 x: 0
(4.) magball/Magnetic Ball Plant/height
 x: 0.05
(5.) magball/Integrator
 x: 0
```

Inputs: None

**See Also**

operpoint, operspec

**update**

---

# Block Reference

---

# Trigger-Based Operating Point Snapshot

---

**Purpose** Generate operating points, linearizations, or both at triggered events

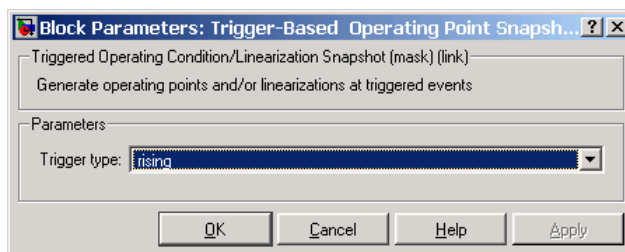
**Library** Simulink Control Design

## Description



Attach this block to a signal in a model when you want to take a snapshot of the system's operating point at triggered events such as when the signal crosses zero or when the signal sends a function call. You can also perform a linearization at these events. To extract the operating point or perform the linearization, you need to simulate the model using either the `findop` or `linearize` functions or the simulation snapshots option in the Control and Estimation Tools Manager.

Choose the trigger type in the Block Parameters dialog box, as shown in the following figure.



The possible trigger types are

- `rising`: the signal crosses zero while increasing.
- `falling`: the signal crosses zero while decreasing.
- `either`: the signal crosses zero while either increasing or decreasing.
- `function-call`: the signal send a function call.

---

**Note** The Simulink Control Design demo called “Trigger-Based Operating Point Snapshot” illustrates how to use this block.

---

# Trigger-Based Operating Point Snapshot

---

**See Also**      findop, linearize

# Trigger-Based Operating Point Snapshot

---

# Examples

---

Use this list to find examples in the documentation.

## **Linearization Example Using the Graphical Interface**

“Example Model: The Magnetic Ball System” on page 1-2

## **Linearization Example Using Functions**

“Example: Water-Tank System” on page 3-3

## **Frequency Response Estimation**

“Example—Effects of Time-Varying Simulink Signals on Frequency Response Estimation” on page 6-28

“Example—Effects of Filtering on Frequency Response Estimation” on page 6-30

“Example—Estimating a Model With Noise Using Signal Processing Toolbox” on page 6-40

“Example—Estimating State-Space Model Using System Identification Toolbox” on page 6-41



## A

- addoutputspec function 9-2
  - example 3-12
- analysis I/Os. *See* input and output points

## C

- captured operating points
  - linearization 4-58
- compensator design
  - analysis plots 7-30
  - beginning a task 7-16
  - closed-loop responses 7-21
  - creating a SISO Design Task 7-26
  - design methods 7-37
  - design plots 7-27
  - exporting 1-14
  - open-loop systems 7-27
  - operating points 7-23
  - overview 7-2
  - retrieving designs 7-42
  - storing designs 7-42
  - time delays 7-36
  - tunable blocks 7-18
  - writing to Simulink model 7-44
- compensator design tasks
  - opening 1-12
  - saving 1-11
- constraining outputs 2-27
  - using functions 3-12
- Control and Estimation Tools Manager
  - compensator design task 7-16
  - opening 1-8
  - overview 1-9
  - SISO Design Task 7-26
- copy function 9-5

## D

- delays

- compensator design 7-36
- direct feedthrough
  - using blocks without 2-38
- discrete-time models
  - linearizing using functions 5-13
  - linearizing with GUI 4-65

## E

- equilibrium operating points 2-2
  - computing 2-10
- equilibrium states
  - computing 2-10
- events
  - extracting operating points from simulation 2-21
  - linearizing at 4-63

## F

- findop function 9-7
  - example using operating point specification objects 3-10
  - example using simulation 3-16

## G

- get function 9-49
  - example with I/O objects 5-8
- getinputstruct function 9-52
- getlinio function 9-53
  - example 5-7
- getlinplant function 9-56
- getstatestruct function 9-58
  - example 3-17
- getxu function 9-60
  - example 3-17

## I

- initial guesses

- specifying with functions 3-11
- initializing simulations 2-24
  - using structures or vectors 3-17
- initopspec function 9-63
  - example 3-11
- inport and outport blocks
  - difference from input and output points 4-46
- input and output points
  - editing I/O objects 5-8
  - inspecting with GUI 4-52
  - other types 4-64
  - purpose 4-45
  - removing 4-48
  - specifying 4-46
  - specifying using functions 5-4
- input points. *See* input and output points

## L

- linear models
  - creating accurate models 2-31
- linearization
  - accurate 2-7
  - beginning a new project 1-8
  - captured operating points 4-58
  - choosing operating points 2-7
  - custom plots 4-71
  - discrete-time models 4-65
  - displaying results 4-66
  - highlighting blocks 4-68
  - inspecting 4-69
  - multirate models 4-65
  - of blocks 4-55
  - operating points 2-6
  - overview 4-8
  - removing results 4-66
  - results 4-66
  - settings 4-9
  - simulation events 4-63
  - simulation times 4-61
  - Simulink model operating points 4-57
  - using functions 5-11
- linearization algorithms
  - block-by-block analytic 4-12
  - blocks with analytic linearizations 4-13
  - numerical perturbation 4-24
- linearization method
  - for nonlinear models 4-2
  - for Simulink models 4-3
- linearization options
  - using functions 5-12
  - using GUI 4-9
- linearization points
  - adding 4-46
  - deleting 4-48
  - other types 4-64
  - removing 4-48
  - selecting 4-46
- linearization projects
  - loading using load function 5-16
- linearization results
  - custom plots 4-71
  - displaying using ltiview function 5-14
  - exporting 1-13
  - in LTI viewer 4-66
  - inspecting 4-69
  - saving using save function 5-16
- linearization tasks
  - opening 1-12
  - saving 1-11
- linearize function 9-66
  - example 5-11
- linio function 9-78
  - example 5-5
- linoptions function 9-81
  - example 5-12
- loop opening
  - purpose 4-49

**M**

- magnetic ball
  - equations 1-3
  - example 1-2
  - Simulink model 1-4
- model references
  - command-line example 4-29
  - graphical interface example 4-31
- multirate models
  - linearizing using functions 5-13
  - linearizing with GUI 4-65

**N**

- numerical perturbation linearization
  - changing perturbation levels 4-28
  - invoking 4-25

**O**

- open-loop analysis 4-49
  - using functions 5-10
- operating point objects
  - creating 3-14
  - editing 3-15
- operating point specification objects
  - creating 3-7
  - editing 3-8
- operating point specifications
  - importing initial values 2-26
- operating point structures
  - initializing simulations with 3-17
- operating points
  - choosing 2-7
  - computing equilibrium 2-10
  - computing from specifications 2-10
  - constraining outputs 2-27
  - copying 2-23
  - creating 2-10
  - creating vectors 3-17

- default operating point 2-10
  - definition 2-2
  - equilibrium 2-2
  - exporting 2-24
  - extracted at simulation times 2-18
  - extracted from simulation events 2-21
  - extracted from simulation times 2-18
  - impact 2-6
  - importance 2-6
  - importing 2-25
  - initializing simulations with 2-24
  - linearizing at 4-57
  - SimMechanics models 2-36
  - Simulink model 2-3
  - special case Simulink blocks 2-31
  - specifying with completely known values 2-15
  - specifying with functions 3-1
  - steady state 2-2
  - trimming 2-10
- operating points specifications
  - using functions 3-7
- operpoint function 9-87
  - example 3-14
- operspec function 9-90
  - example 3-8
- optimization settings
  - changing 2-27
- output constraints. *See* Constraining Outputs
- output points. *See* input and output points

**P**

- perturbation
  - blocks 4-13
- perturbation levels
  - changing 4-41
- PID
  - automatic tuning 7-4
- project

linearization 1-8

## **R**

retrieving compensator designs 7-42

## **S**

set function 9-96

    example with I/O objects 5-9

    example with operating point objects 3-15

    example with operating point

        specifications 3-9

setlinio function 9-99

    example 5-6

setxu function 9-103

    example 3-18

simulation events

    extracting operating points 2-21

    linearization 4-63

simulation times

    extracting operating points 2-18

    linearization 4-61

Simulink model operating points 2-3

    blocks with internal states example 2-32

    impact of blocks 2-31

    linearization 4-57

SISO Design Task

    creating 7-26

state ordering

    changing 4-34

steady state operating points 2-2

    computing 2-10

storing compensator designs 7-42

## **T**

trimming

    operating points 2-10

## **U**

update function 9-105

## **W**

water-tank system

    equations 3-4

    example 3-3

    Simulink model 3-5