Simulink® Control Design

For Use with Simulink®

- Modeling
- Simulation
- Implementation

User's Guide

Version 2

How to Contact The MathWorks:

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

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

508-647-7000 (Phone)

508-647-7001 (Fax)

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

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

Simulink® Control Design User's Guide

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)

Contents

1 Working with Simulink Control Design Projects

2 Specifying Operating Points

3 Creating Linearized Models

4 Designing Compensators

5 Specifying Operating Points Using Functions

6 Linearizing Models Using Functions

7 Understanding and Controlling Results of Linearized Models

Functions — By Category

8

Functions — Alphabetical List

9

Blocks — Alphabetical List

10

Examples

A

Index

1

Working with Simulink Control Design Projects

The creation of operating points, the linearization of Simulink® models, and the design of compensators with Simulink Control Design, all take place within a project in the Control and Estimation Tools Manager. This section shows how to create, save, and load these projects and introduces the tasks that the projects contain.

Beginning a Project

With Simulink Control Design 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 tasks from Simulink Control Design, it can include tasks from other products such as Simulink Parameter Estimation, Control System Toolbox, Simulink Response Optimization, and Model Predictive Control Toolbox.

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

Simulink Control Design automatically creates an **Operating Points** node in the Control and Estimation Tools Manager 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 “Beginning a Project” on page 1-2 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**.

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 1-2 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**.

Loading Previously Saved Projects

See “Opening Previously Saved Projects” in Getting Started with Simulink Control Design for more information.

Saving Projects

See “Saving Projects” in Getting Started with Simulink Control Design for more information.

2

Specifying Operating Points

Creating Operating Points

Before linearizing the model, you must choose an operating point about which to linearize the system. This is often a steady state value. Refer to "What Are Operating Points?" in the Getting Started with Simulink Control Design documentation for more information on the role of operating points in linearization.

Simulink Control Design provides five methods for specifying the operating point of a model:

- Specify target values or constraints on a subset of the model's inputs, outputs, and states (see "Creating Operating Points from Specifications" in the Getting Started with Simulink Control Design documentation). Simulink Control Design uses numerical methods to determine the full operating point based on this partial specification.

  For example, when you know that the height of the ball in `magball` should be 0.05, the rate of change of the height is small, the current should be positive, and your initial guess for the states in the Controller is 0, compute an operating point that closely matches the specifications.

- Completely specify all inputs and states in the operating point (see "Creating Operating Points from Known Values" in the Getting Started with Simulink Control Design documentation).

  For example, when you know that the height of the ball in `magball` should be 0.05, the rate of change of the height should be 0, the current should be 7.0036, and you also know the values of the states in the Controller, this information completely specifies the operating point.

- Extract an operating point from a simulation of the model (see "Creating Operating Points from Simulation" in the Getting Started with Simulink Control Design documentation).

  For example, you run a simulation of a model and use the values of the states and inputs at time 10 as the operating point values. This is especially useful when the simulation has reached a steady state.

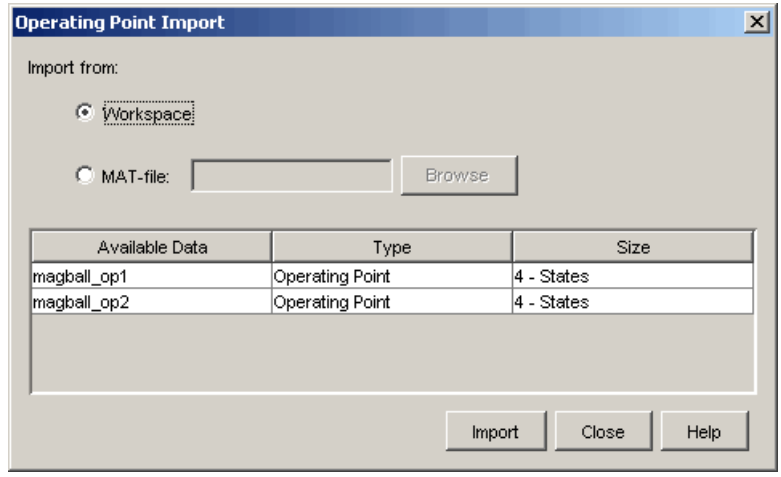
- Accept the default operating point. The initial values of the states, inputs, and outputs, define this operating point. Only use the default operating point when the initial values are very close to the operating point of interest.

- Import a previously saved operating point from another project, the MATLAB workspace, or a file (“Importing Operating Points” on page 2-3).

Importing Operating Points

Use Simulink Control Design to import operating points from the MATLAB workspace or from a MAT-file.

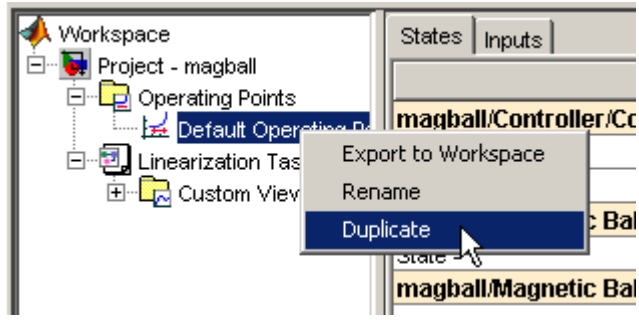
1 To import a new operating point, select the **Operating Points** node in the project tree and then select the **Operating Points** tab on the right. Click the **Import** button at the bottom of the pane. This displays the Operating Point Import dialog box.



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.

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.

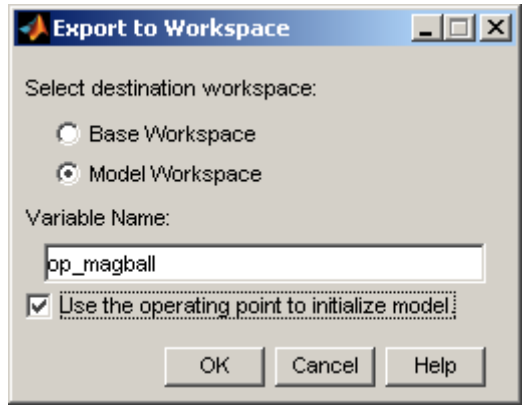


The new operating point appears beneath the original one in the tree. Click the new operating point to display its contents in the pane on the right. To change state or input values in the duplicated operating point, edit the values in the right pane. To change the name of the new operating point, right-click 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 Simulink Control Design, 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 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.

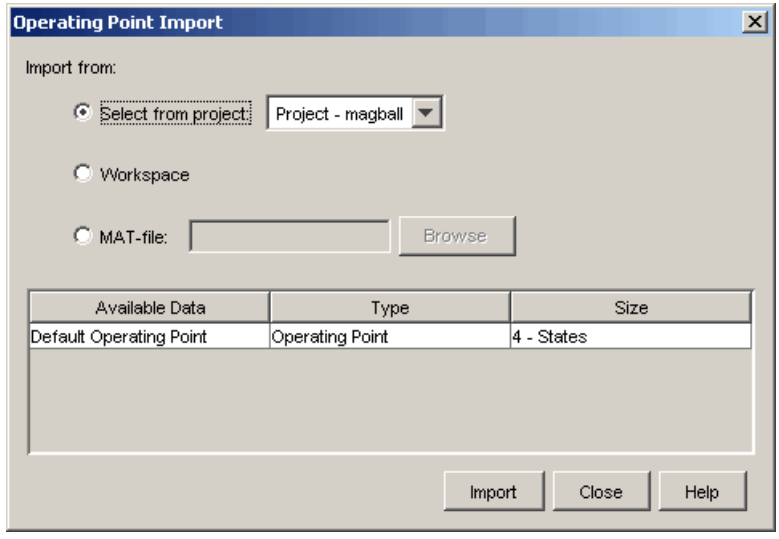
3 Select **Use the operating point to initialize model** when you want to use the operating point values as initial conditions for the states and inputs in the model. The initial values are automatically set in the **Data Import/Export** pane of the Configuration Parameters dialog box and Simulink uses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

Computing Equilibrium Operating Points

You can use Simulink Control Design to compute equilibrium operating points. Follow the basic instructions in “Creating Operating Points from Specifications” in the Getting Started with Simulink Control Design documentation. When you enter specifications in the **States** pane, select the **Steady State** check box at the top of the table. This causes the algorithm to look for an operating point in which all states are at equilibrium, or steady state.

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.

Using Operating Points

You can use operating points in Simulink Control Design linearization and compensator design tasks. In both these tasks, the creation and selection of accurate and appropriate operating points plays a critical role.

You can also use operating points to initialize a model for simulation. For information on this see “Exporting Operating Points” on page 2-4

3

Creating Linearized Models

Linearization Background

Linearized models are useful in a wide range of applications, including compensator design. For an introduction to the concepts of linearization, see "What Is Linearization?" in the Getting Started with Simulink Control Design documentation. The following sections formally describe a linearized model and introduce the state-space equations for linearized 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

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

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

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

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

$$x_{k+1} = f(x_k, u_k, t_k)$$
$$y_k = g(x_k, u_k, t_k)$$

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

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

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:

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

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

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

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

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

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

where k_1,..., k_{m} are integer values and $t_{k_1}, \ldots, t_{k_m}$ 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”.

Linearizing Models with Simulink Control Design

For basic instructions on creating linearized models with Simulink Control Design see “Linearizing Models” in the Getting Started with Simulink Control Design documentation. The following section introduces methods for further controlling and modifying the results of a linearization.

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 Simulink Control Design 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.

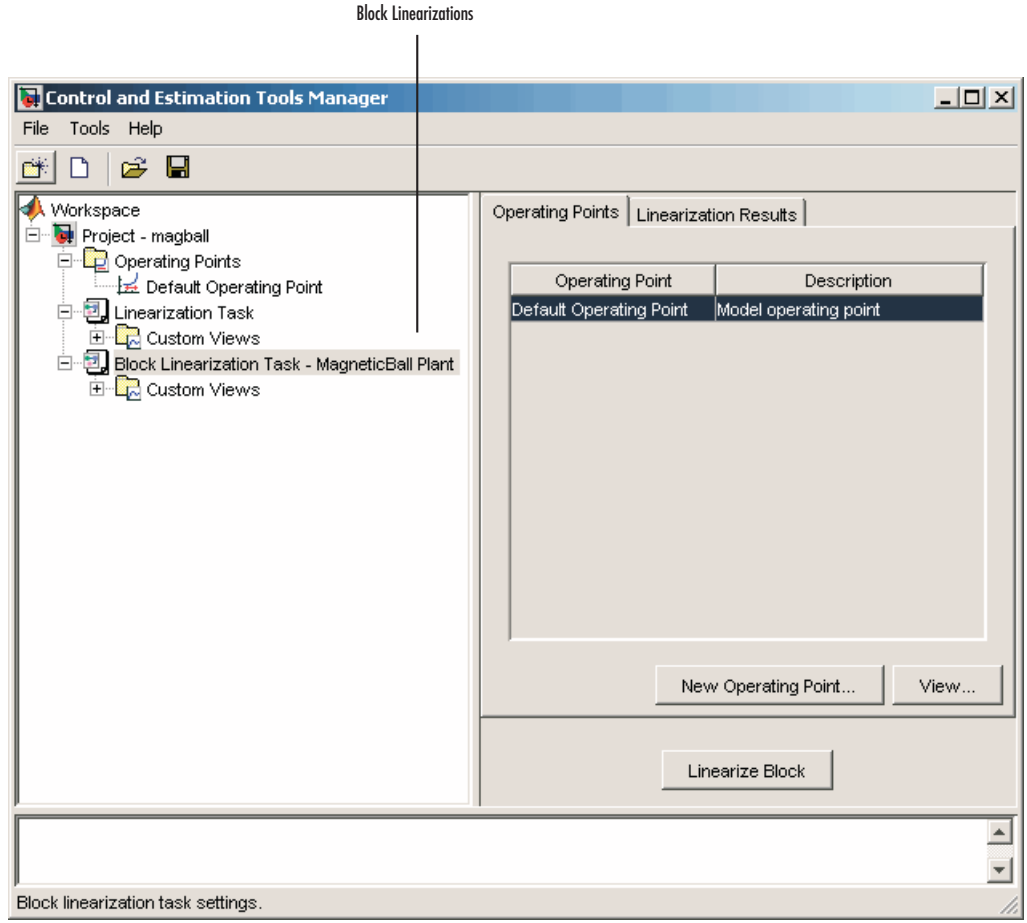
To change the method that Simulink Control Design uses for converting a multirate model to a single-rate model, change the **Rate conversion method** in the Options dialog box.

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.

Linearizing a Block

With Simulink Control Design 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, and 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 linearize an individual block, it must contain at least one data inport and outport. Since SimMechanics and SimPowerSystems blocks have connection ports instead of inports and outports, they cannot be individually linearized.

4

Designing Compensators

What Is Compensator Design?

Compensator design is the process of designing compensators for a control system so that the system behaves in a desired way. Compensators in Simulink models are represented by blocks such as Transfer function, Zero-Pole-Gain, and PID blocks. These blocks can act as feedback controllers, pre-filters, feedforward controllers, sensors, etc. Compensator design for Simulink models can be as simple as adjusting gains in a one of these blocks, or as complicated as adding, deleting, or moving poles and zeros in multiple compensators over multiple feedback loops.

Compensator design methodologies often use tools such as Bode diagrams, root-locus diagrams, or response plots. These tools require that the plant model is linear; however, most real-world systems are nonlinear. Simulink Control Design simplifies the task of designing compensators for Simulink models by automatically linearizing the model before creating a SISO Design Task in which you can edit and design the compensators using a variety of tools.

Compensator Design Process Overview

Compensator design in the Control and Estimation Tools Manager involves the following steps:

1. "Picking Blocks to Tune"
2. "Selecting Closed-Loop Responses to Design"
3. "Selecting an Operating Point"
4. "Creating a SISO Design Task"
5. "Completing the Design"

For more information about steps 1–3, see "Working in Simulink Compensator Design Task Pane" on page 4-4. For more information about steps 4–5, see "Enhanced SISO Design Task" on page 4-7.

Working in Simulink Compensator Design Task Pane

The Simulink Compensator Design Task pane lets you configure blocks to tune, select the closed-loop response, and select the operating point at which to linearize the model. After you specify this information, the GUI uses it to perform the calculations necessary to populate the SISO Design Task node.

This section includes the following topics:

What Blocks Are Tunable by Simulink Control Design?

The following blocks have parameters that are tunable using Simulink Control Design. Note that the block input and output signals must be scalar, double-precision values.

- Gain
- PID Controller
- PID Controller (with Approximate Derivative)
- LTI
- State-Space
- Zero-Pole
- Transfer Fcn
- Discrete State-Space
- Discrete Zero-Pole
- Discrete Transfer Fcn
- Discrete Filter

- Transfer Fcn First Order
- Transfer Fcn Lead or Lag
- Transfer Fcn Real Zero
- Blocks that have custom configuration functions associated with a masked subsystem

For additional information, see “Picking Blocks to Tune” in the Getting Started with Simulink Control Design documentation.

Creating Custom Configuration Functions

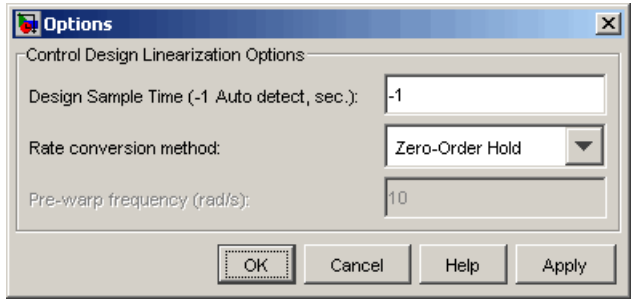
When you have masked subsystems that you want to tune in your model, they will not automatically appear in the list of tunable blocks. For them to appear in the list, you need to create a custom configuration function for the masked subsystem. The custom configuration function serves the following functions:

- It informs Simulink Control Design 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”. For additional information, see “Picking Blocks to Tune” in the Getting Started with Simulink Control Design documentation.

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.

For additional information, see “Selecting an Operating Point” in the Getting Started with Simulink Control Design documentation.

Enhanced SISO Design Task

The enhanced SISO Design Task lets you tune compensators using functionality from the Control System Toolbox and Simulink Response Optimization. It 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)
- A response optimization tool that automatically tunes the system to satisfy design requirements (available when you have the product Simulink Response Optimization)

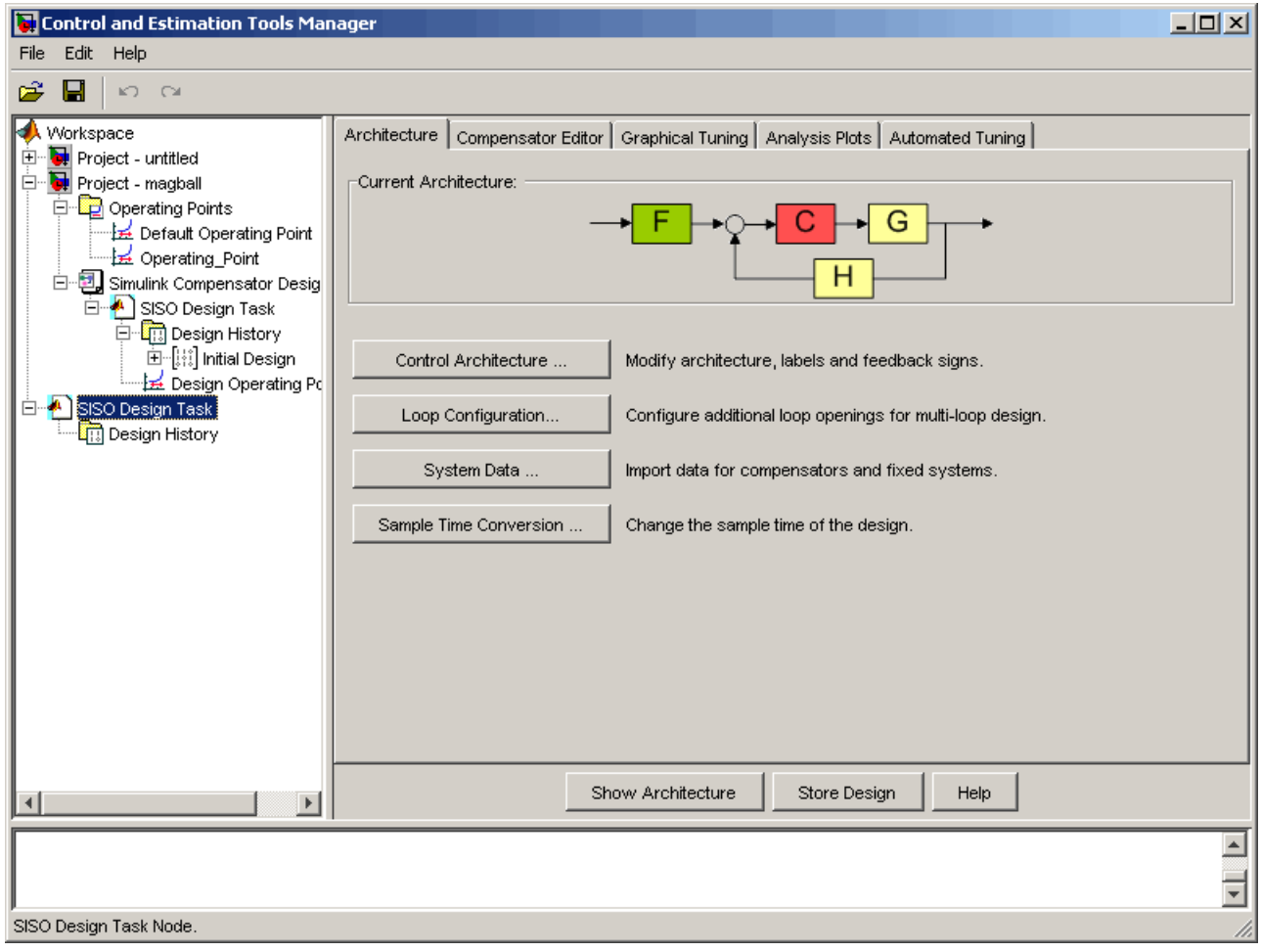
This section includes the following topics:

Compare and Contrast the SISO Design Task and Enhanced SISO Design Task

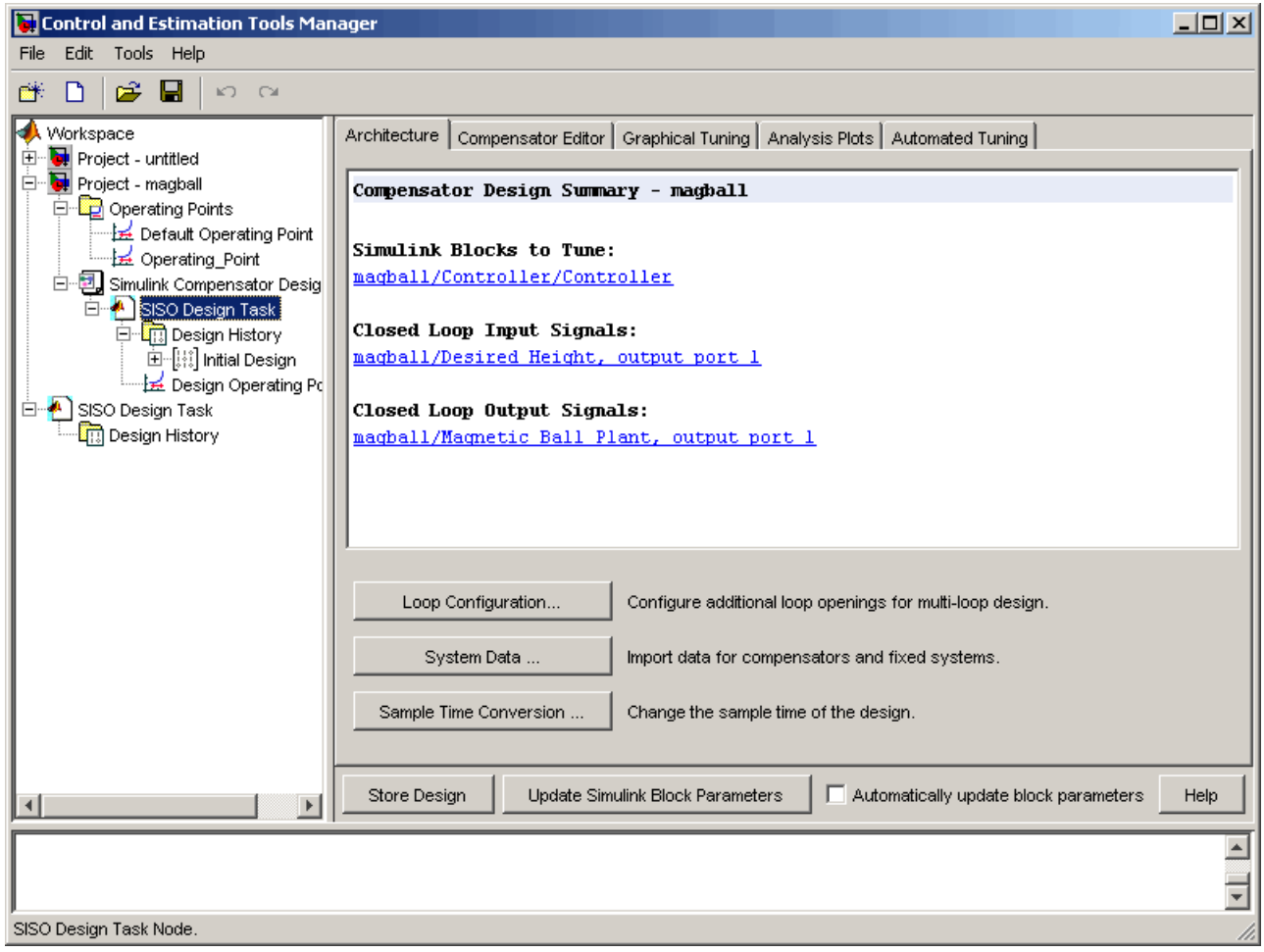
The SISO Design Task is a graphical user interface (GUI) that simplifies the task of designing controllers. This section describes the similarities and differences between the SISO Design Task, which is available in the Control

System Toolbox, and the enhanced SISO Design Task, which is available with Simulink Control Design.

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.



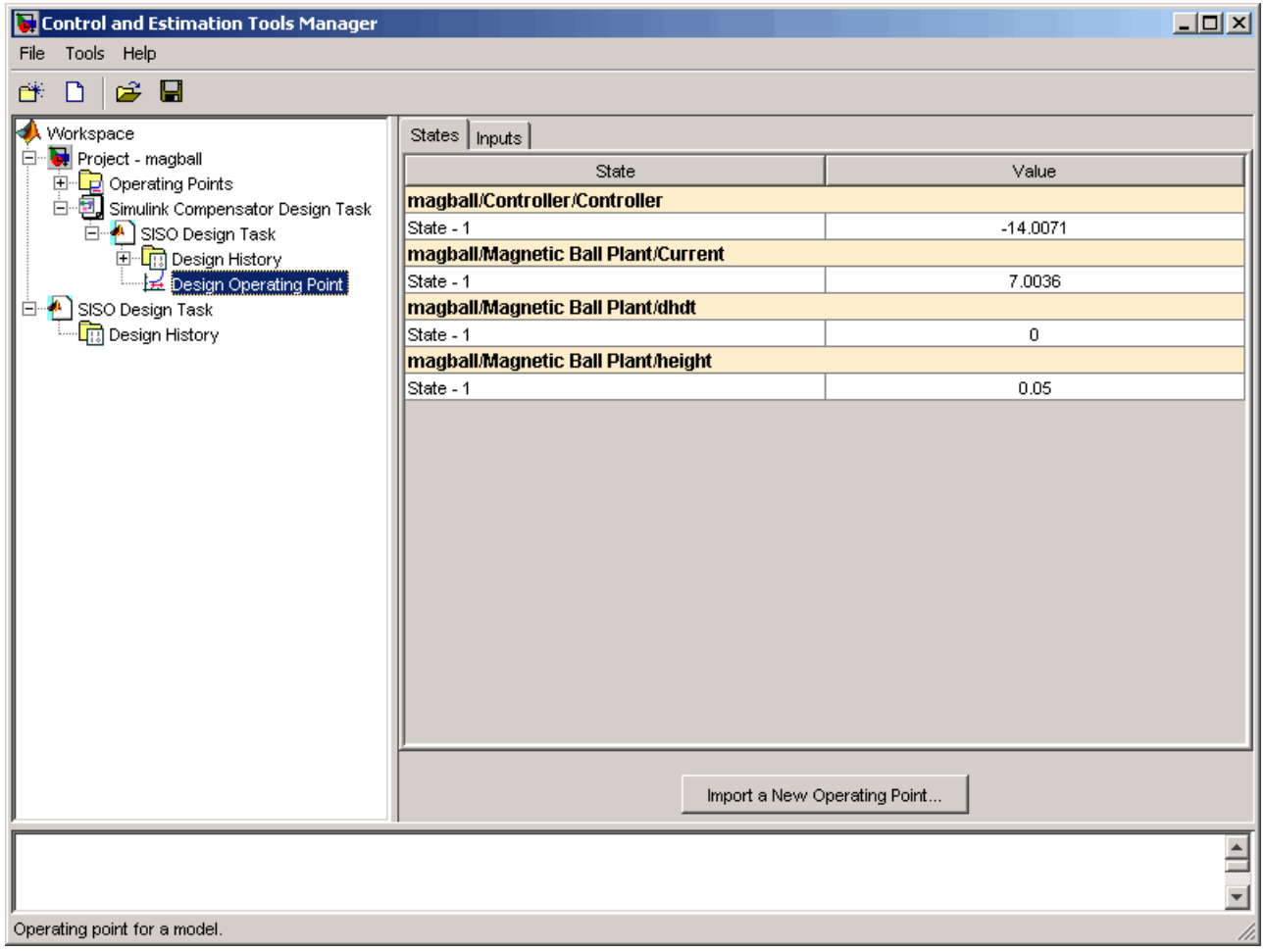
The following table summarizes the similarities and differences between the SISO Design Task and the enhanced SISO Design Task:

Similarities	Differences
• Similar layout • Graphical Tuning, Analysis Plots, and Automated Tuning panes have the same functionality. For more information about these tabs, see "Completing the Design" in the Getting Started with Simulink Control Design documentation.	• 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". • Once you are happy 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" in the Getting Started with Simulink Control Design documentation, "Designing Compensators", and "The SISO Design Tool" in the Getting Started with the Control System Toolbox documentation.

Design Operating Point Node

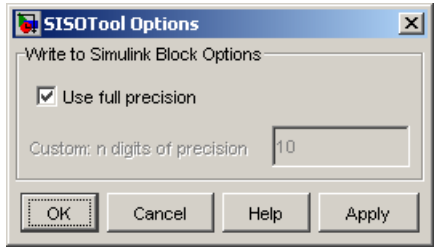
The **Design Operating Point** node is located inside the **Design History** node of the Control and Estimation Tools Manager.



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” in the Getting Started with Simulink Control Design documentation.

5

Specifying Operating Points Using Functions

You can use the Simulink Control Design functions to create operating points for a Simulink model 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.

Overview

This section describes how to specify operating points for a model using functions. For a description of how to use the graphical interface for this task, see Chapter 2, “Specifying Operating Points”.

Before linearizing the model, you must choose an operating point to linearize the system about. This is often a steady-state value. Refer to “Specifying Operating Points” in the Getting Started with Simulink Control Design documentation 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.

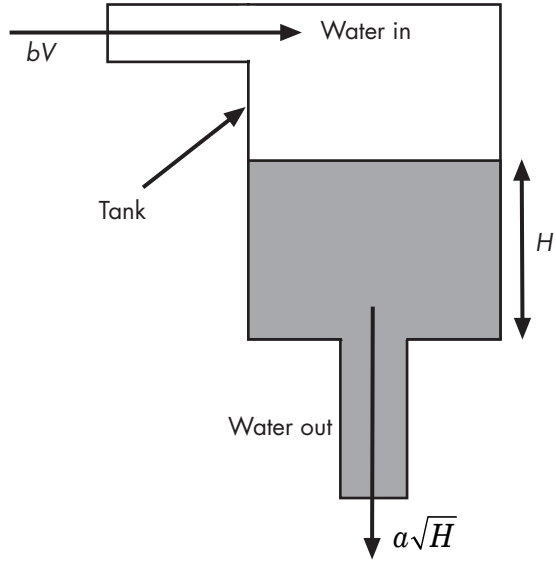
Note The operating point consists of values for *all* the states in the model although only those states between the linearization points will be linearized. This is because the whole model contributes to the operating point values of the states/inputs/outputs of the portion of the model you are linearizing.

Example: Water-Tank System

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 functions from Simulink Control Design.

Water-Tank System

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 5-3.

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, using Simulink Control Design, simplifies the analysis of this model. For information on the linearization process, see “Linearizing Models” in the Getting Started with Simulink Control Design documentation.

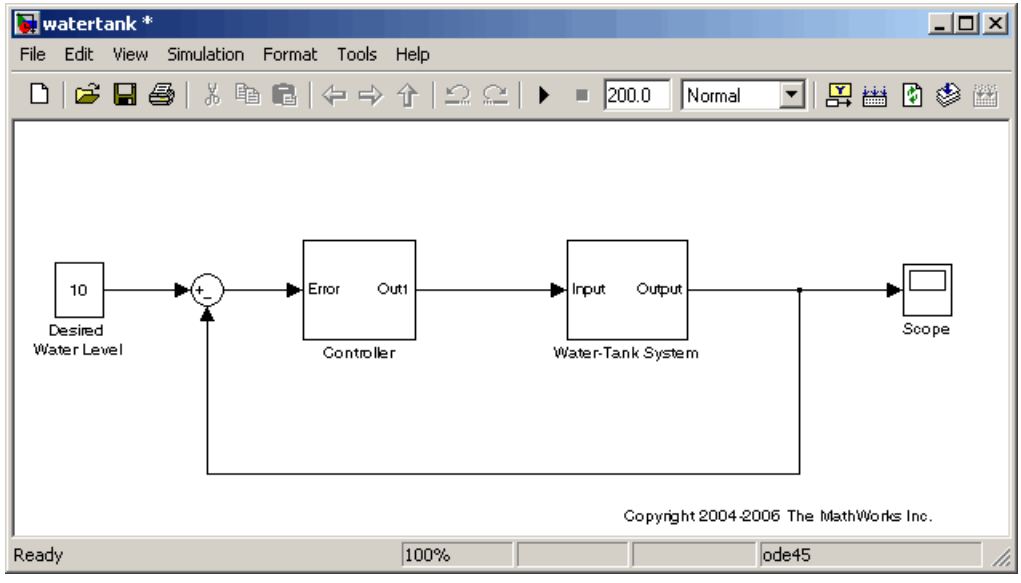
Creating or Opening a Simulink Model

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.



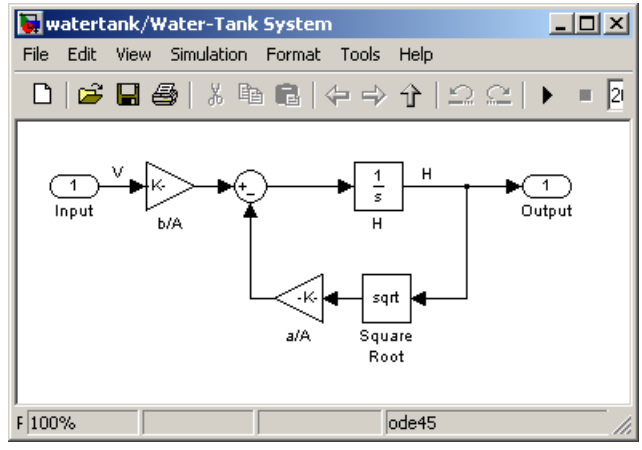
Simulink Model of the Water-Tank System

The `watertank` model consists of

- The water-tank system itself

- A Controller subsystem 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 Controller block contains a simple proportional-integral-derivative controller. 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 Controller, 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 $cm^{2.5}/s$, A=20 cm^2, b=5 $cm^3/(s \cdot V)$.

Computing Operating Points from Specifications

This section continues the example from "Example: Water-Tank System" on page 5-3. At this stage in the example, linearization point objects have been created in the MATLAB workspace. See "Configuring the Model for Linearization" on page 6-3 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 5-7.
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 5-8.
3. Use the `findop` function to find the operating point values by optimization. See "Computing the Complete Operating Point" on page 5-9.

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 displays

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

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

```
      spec:  dx = 0,  initial guess:            0

  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/Controller/Integrator'
                    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/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 Point Search Report:
---------------------------------

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

Operating condition specifications were successully met.

States:
----------
(1.) watertank/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 5-15, 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 Specificaton for the Model watertank.
 (Time-Varying Components Evaluated at time t=0)

States:
----------
(1.) watertank/Controller/Integrator
      spec:  dx = 0,  initial guess:         0.872
(2.) watertank/Water-Tank System/H
      spec:  dx = 0,  initial guess:           9.7

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 5-8.

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 5-10, 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 outport of a block, in this case outport `1` of the block `watertank/Water-Tank System/Sum` (the preceding block).

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

States:
----------
(1.) watertank/Controller/Integrator
      spec:  dx = 0,  initial guess:         0.872
(2.) watertank/Water-Tank System/H
      spec:  dx = 0,  initial guess:           9.7

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

This section continues the example from "Example: Water-Tank System" on page 5-3. At this stage in the example, linearization point objects have been created in the MATLAB workspace. See "Configuring the Model for Linearization" on page 6-3 for more information on creating linearization point objects.

To use functions to specify completely known operating points, first create an operating point object, and then make changes to the object values.

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 displays

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

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

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/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 5-3. At this stage in the example, linearization point objects have been created in the MATLAB workspace. See "Configuring the Model for Linearization" on page 6-3 for more information on creating linearization point objects.

Use Simulink Control Design 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 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/Controller/Integrator
      x: 1.25
(2.) watertank/Water-Tank System/H
      x: 9.99

Inputs: None
```

Using Structures and Vectors of Operating Point Values

This section continues the example from "Example: Water-Tank System" on page 5-3. At this stage in the example, linearization point objects and operating points have been created in the MATLAB workspace. See Chapter 5, "Specifying Operating Points Using Functions" 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 5-15, use the following command:

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

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

```
x =
       time: 0
    signals: [1x2 struct]
```

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

```
x.signals.values
```

which returns

```
ans =
    1.2469

ans =
    9.9927
```

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 =
    9.9927
    1.2469

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.

6

Linearizing Models Using Functions

You can use the Simulink Control Design linear analysis functions to linearize Simulink models in the MATLAB Command Window. Use the functions when you want to create M-files to automate the linearization process, or perform *batch* linearization.

Overview

As discussed in “Purpose of Linearization” in the Getting Started with Simulink Control Design 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
2. Configuring the model
3. Specifying operating points
4. Linearizing the model
5. Analyzing the results and saving your work

The following section introduces an example containing a nonlinear system, a water-filled tank. Subsequent sections use the example for a detailed discussion of each step.

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 vs. Command-Line Functions” in the Getting Started with Simulink Control Design 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 Model for Linearization

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 “Configuring Inputs and Outputs for the Linearized Model” in the Getting Started with Simulink Control Design documentation.

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 “Linearization of Simulink Models” in the Getting Started with Simulink Control Design documentation 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 5-3.

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 outport of a block. To place an input point before the Water-Tank System block, you need to associate this input point with the outport of the Controller block.

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

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

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

```
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
```

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 outport 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 outport of the block `watertank/Water-Tank System`. The object `watertank_io` is displayed, as shown below.

```
Linearization IOs:
--------------------------
Block watertank/Controller, Port 1 is marked with the following
properties:
 - No Loop Opening
 - An Input Perturbation
```

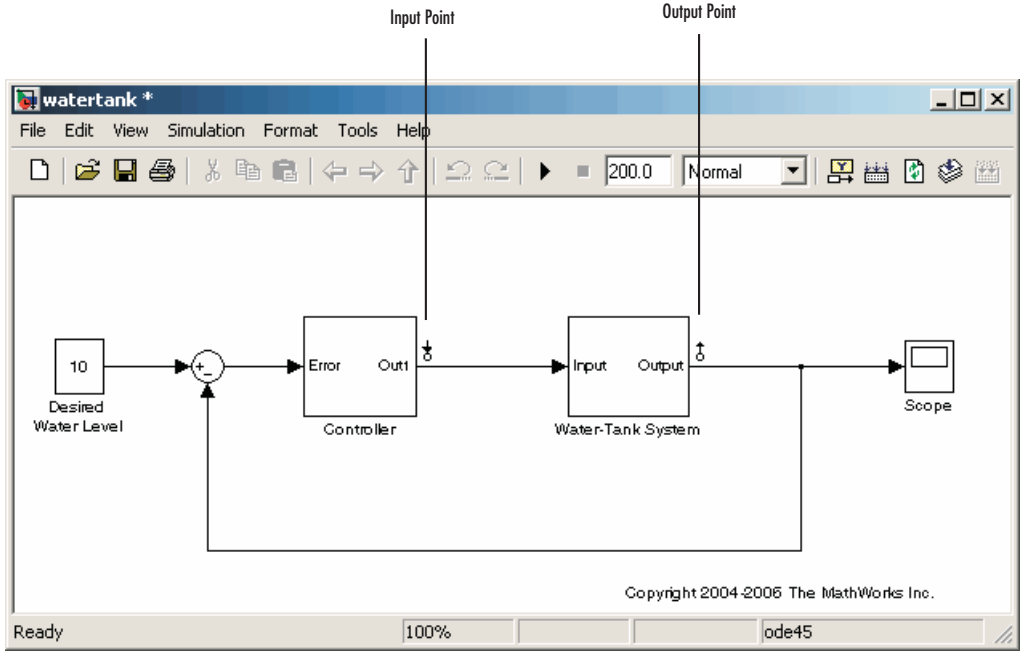
```
 - No signal name. Linearization will use the block name

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
```

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 5-3. At this stage in the example, linearization points have been inserted in the model. See "Choosing and Storing Linearization Points" on page 6-3 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 "Inserting Linearization Points" when they have been selected using the right-click menus described in. 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 5-3. 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 6-6 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: ''
```

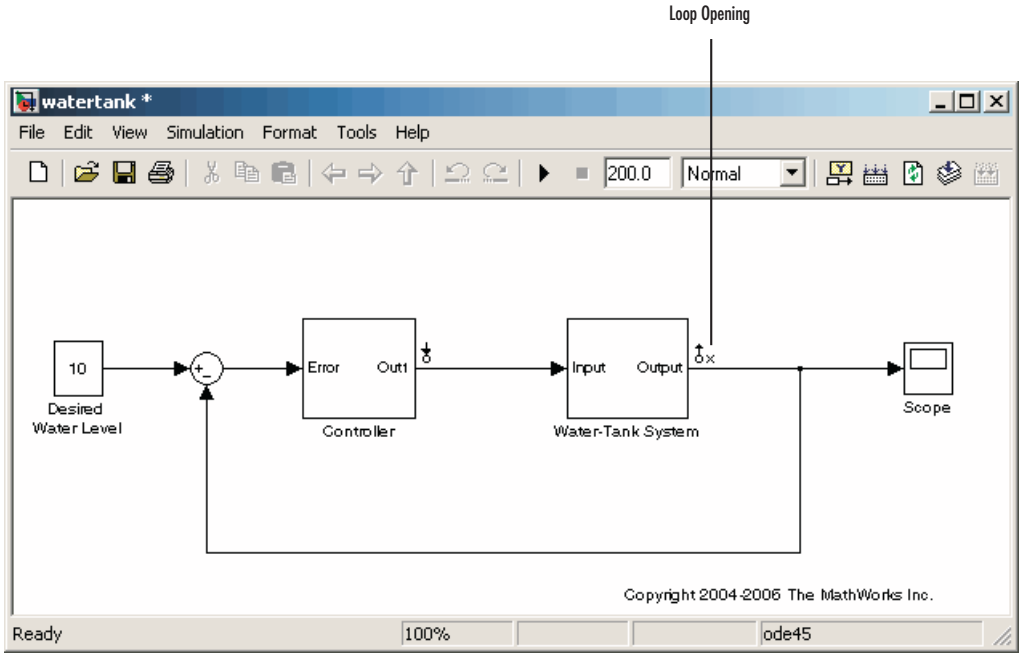
You can edit this object to make any necessary changes. For example, to make this linearization point a loop opening as well, type

```
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” in the Getting Started with Simulink Control Design documentation. 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 6-6.

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" in the Getting Started with Simulink Control Design documentation.

This section also continues the example from "Example: Water-Tank System" on page 5-3. At this stage in the example, linearization point objects and operating point have been created in the MATLAB workspace. See Chapter 5, "Specifying Operating Points Using Functions" 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 will return 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  -0.01582


b =
       Controller (
   H           0.25


c =
                   H
   Water-Tank S    1


d =
                   Controller (
   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  0.9843


b =
       Controller (
   H         0.248


c =
                  H
   Water-Tank S   1


d =
                  Controller (
   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 Simulink Control Design 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 Simulink Control Design demo “Linearization of Multirate Models”.

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 "Analyzing the Results" in the Getting Started with Simulink Control Design documentation.

Analyze the linearized model at the MATLAB prompt using functions from the Control System Toolbox, or display it in the LTI Viewer. Alternatively, incorporate the results into a block in a Simulink model. For methods on simulating the linearized model for comparison with the original model, refer to "Comparing the Linearized and Original Models" on page 7-3.

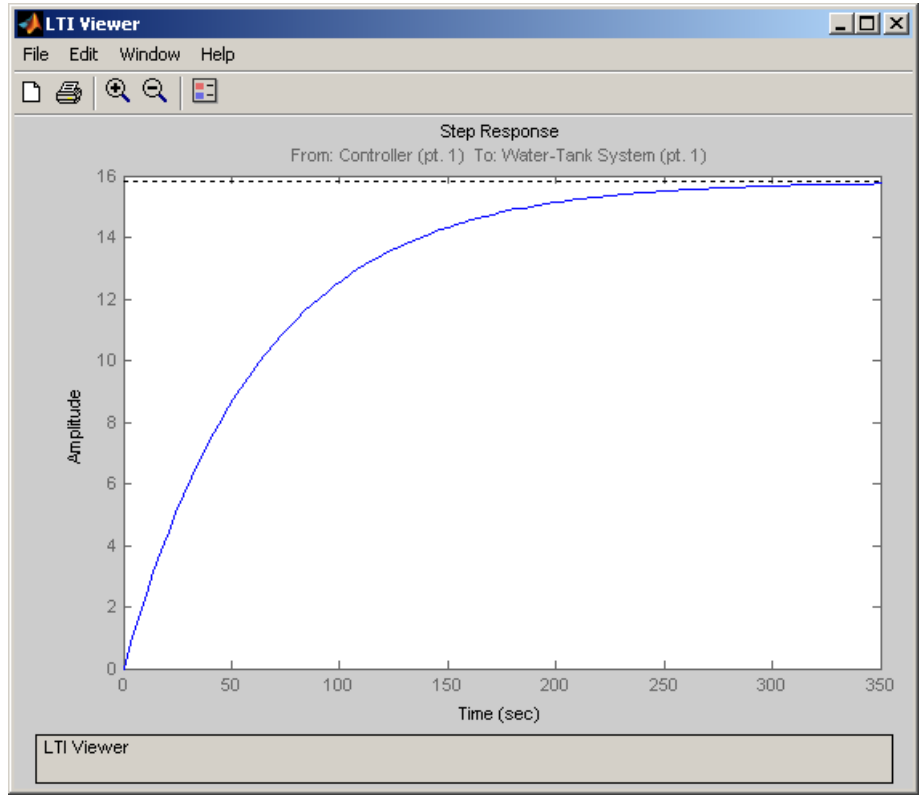
Using the LTI Viewer

This section continues the example from "Example: Water-Tank System" on page 5-3. 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" on page 6-9 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 5-3. 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" on page 6-9 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 1-5.

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.

7

Understanding and Controlling Results of Linearized Models

To create accurate linearized models, it is important to be able to interpret the results and to understand the linearization algorithms. One method of interpreting the results is by simulating the linearized model and comparing the output with the original model. The linearization algorithms can be adjusted in various ways to control these results, as outlined in this chapter.

Comparing the Linearized and Original Models (p. 7-3)	Methods for simulating the linearized model and comparing the results to the original model
Linearization Algorithms (p. 7-9)	Brief introduction to the two main linearization methods with advantages and disadvantages of each
Block-by-Block Analytic Linearization (p. 7-11)	Description of the default linearization method with suggestions for controlling the results

Numerical-Perturbation Linearization (p. 7-27)	Description of an alternative linearization method with suggestions for controlling the results
Recommendations for Computing Operating Points and Creating Accurate Linearized Models (p. 7-38)	Description of what it means to linearize a Simulink® model and how to use correct modeling techniques

Comparing the Linearized and Original Models

Comparing simulations of the original model with simulations of the linearized model helps to determine if the linearized system behaves in a similar way to the original model. To make this comparison, re-insert the linearized subsystem into the model, configure the inputs and operating points so that they are the same as in the original model, and then compare output signals from a simulation of the two models.

When comparing models, remember that the states, inputs, and outputs of the linearized model are defined about an operating point of the original model, using the following variables:

$$\delta x(t) = x(t) - x_0$$
$$\delta u(t) = u(t) - u_0$$
$$\delta y(t) = y(t) - y_0$$

This means that when the original model is at the operating point $x(t)=x_0$, $u(t)=u_0$, $y(t)=y_0$, the linearized model will be at the operating point $\delta x(t)=0$, $\delta u(t)=0$, $\delta y(t)=0$. To compare the models accurately, subtract u_0 from input values and x_0 from the initial state values in the linearized model, then add y_0 to the output signal.

When you linearize only a portion of the original model, you should simulate the linearized model by substituting it back into the model in place of the original portion. This ensures that the operating point and inputs to the linearized portion are correct. To do this, export the linearized model to the workspace, delete the original portion from the model, and replace it with an LTI System block based on the linearized model.

Example

This example compares the `magball` model with the linearized model computed in “Linearizing the Model” in the online documentation:

1. If you have not done so already, linearize the `magball` model at the targeted operating point computed in “Creating Operating Points from Specifications”.

2. To create a new model containing the linearized plant system, first export the linearized model and operating point from the Control and Estimation Tools Manager to the MATLAB® workspace. To do this, right click the linearized model name in the project tree of the Control and Estimation Tools Manager. Select **Export** from the menu. Accept the default name for the model, `Model_sys`, and for the operating point, `Model_op`.

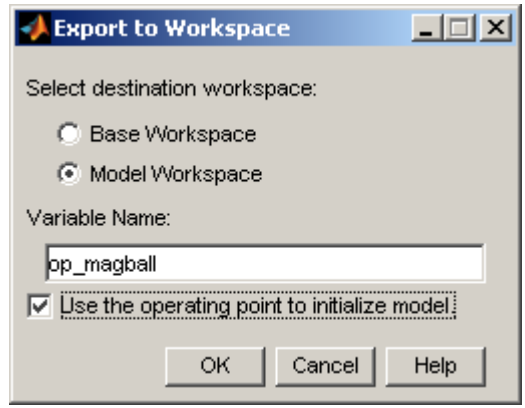
3. Create a new Simulink® model, `magball_lin`, which is a copy of the original model, `magball`. Replace the Magnetic Ball Plant subsystem in `magball_lin` with an LTI System block (located in the Control System Toolbox category of the Simulink Library Browser). Import the linearized model into this block by entering `Model_sys` in the **LTI system variable** field in the Block Parameters window.

4. For simulations of the nonlinear and linearized models to be compared, you need to set the operating points for each model by specifying the initial values of the states in the models:

   a. `magball`

      To set the initial values for the `magball` model, in the Control and Estimation Tools Manager, right click on the operating point that you used for the linearization, and select **Export to Workspace** to open the Export to Workspace dialog box.

      Within the Export to Workspace dialog box, click **Model Workspace** as the location to export the operating point to, and select the check box **Use the operating point to initialize model**.



Click **OK** to export the operating point to the model workspace and use it to define the initial values of states in the model.

b `magball_lin`

In `magball_lin`, the operating point values for the linearized system will all be zero since this subsystem was linearized about the operating point values. The operating point values in the Controller will be the same as in the original model since the Controller was not linearized. To create a vector of initial state values with the correct state ordering, first create a new operating point object for the system by typing

```
op=operpoint('magball_lin')
```

Change the operating point for the Controller in `op` to be the same as those in `Model_op`.

```
op.States(1).x=Model_op.States(1).x
```

This returns the following operating point:

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

States:
----------
(1.) magball_lin/Controller/Controller
      x: 0
```

```
      x: -2.56e-006
(2.) magball_lin/LTI System/Internal
      x: 0
      x: 0
      x: 0

Inputs: None
```

Keep the operating point for the LTI system as zero.

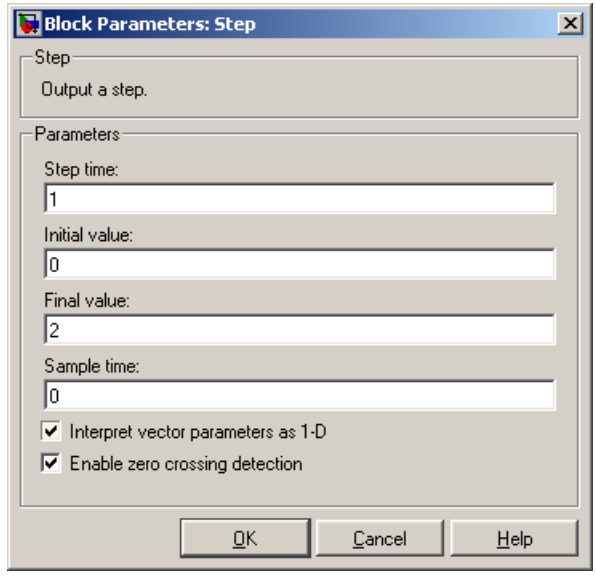
Create a Simulink structure from this operating point using the `getstatestruct` function. The structure contains the operating point values in a format that Simulink can use to set initial values.

```
x_struct=getstatestruct(op);
```

To use the values in `x_struct1`, as initial values for `magball_lin`, select **Simulation > Configuration Parameters** in the `magball_lin` model window, then click the **Data Import/Export** tab. Select the check box next to **Initial State** and enter `x_struct` on the right. Click **OK** .

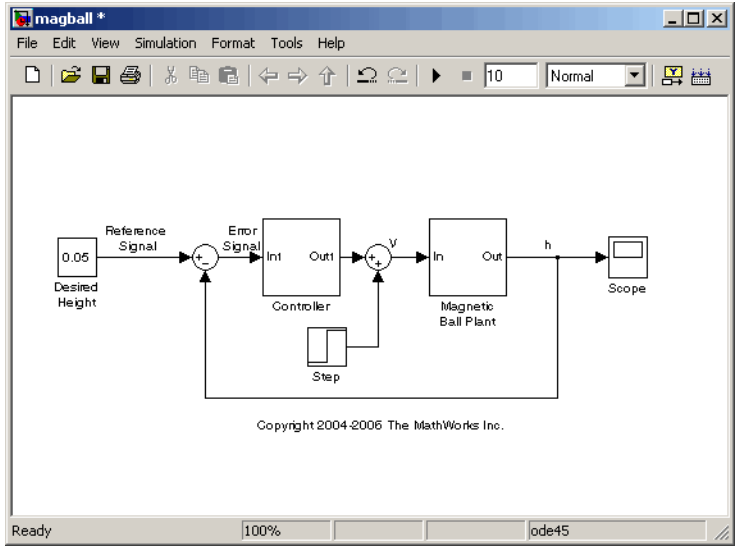
5. The output of `magball_lin` will be zero at the operating point. To create an output signal that is comparable with that in `magball`, add a Constant block, with a value of `0.05` to the output of `magball_lin`. Similarly, the input to `magball_lin` should be zero at the operating point. This is achieved by subtracting a value of 14 from the input signal of the linearized system. The operating point values, 0.05 and 14, were found using a Scope block to measure steady-state signal levels in the original model.

6. To observe the response of the models to a perturbation, add a Step block with the following parameter values to the input to the plant in both models.

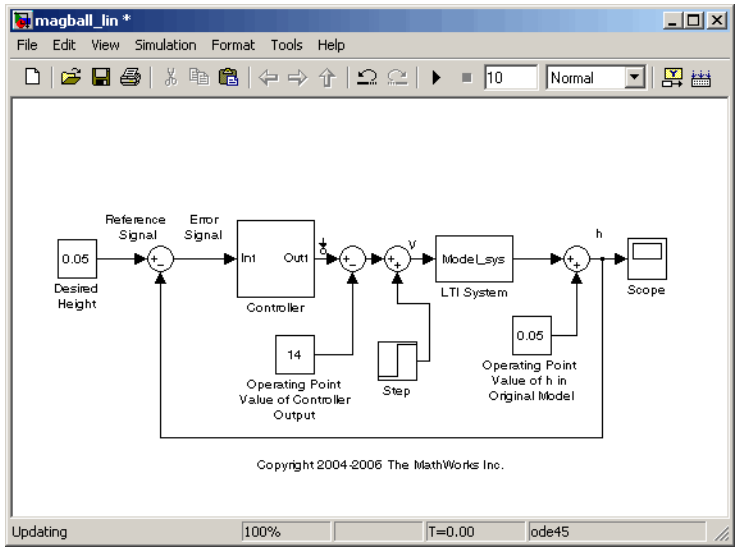


Parameter Values for Step Block

The model diagrams should now look like those in the following figures.

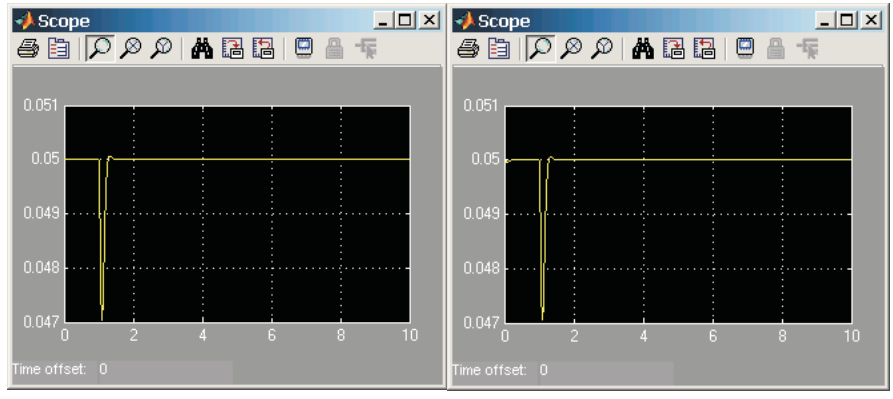


Magball Model with a Step Block Added to the Input



Magball Model with Linearized Magnetic Ball Plant

7 Run simulations in both models. The output signals, in the Scope blocks, are shown in the following figure.



Scope Blocks from Original (left) and Linearized (right) Models

As shown in the figure, both the original and linearized models react to the step input in a similar way.

Linearization Algorithms

Simulink Control Design can use two different linearization methods. The default method, which is used unless an option is selected, is called block-by-block analytic linearization. To use the alternative method, numerical-perturbation linearization, you must select an option in the Linearization Options dialog box of the GUI, or if using functions, with the `linoptions` function. The remainder of this chapter describes the two linearization methods in more detail and provides suggestions for controlling the results to create more accurate linearized models.

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.

The main disadvantage of this method is that it cannot be used with models that contain model references using the Model block.

Alternatively, numerical-perturbation linearization linearizes the *whole system* by numerically perturbing the system's inputs and states about the operating point. This method has the advantage that it is quick and simple, especially for large or complicated systems. In addition it is the only linearization algorithm that supports models containing model references. 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).

"Block-by-Block Analytic Linearization" on page 7-11 and "Numerical-Perturbation Linearization" on page 7-27 discuss these methods further.

Block-by-Block Analytic Linearization

Block-by-block analytic linearization is the default linearization method in Simulink Control Design. In this method, each of the blocks within the linearization path is first linearized individually. The linearization of the whole system is then computed by combining these results using the algorithm discussed in . This approach breaks the problem into several smaller problems. The following section gives details of the methods used to linearize each block, with suggestions for controlling the linearizations to create more accurate linearized models.

Note The block-by-block analytic linearization algorithm does not work with models that contain references to other models using the Model block. Use the numerical perturbation linearization algorithm to linearize these models.

Individual Block Linearization Methods

There are two methods that Simulink Control Design uses to linearize the individual blocks in a model. Each method has options that you can control to create accurate linearized models.

Analytic Linearization

Many Simulink blocks contain analytic Jacobians for exact linearization. When linearizing a system using block-by-block analytic linearization, you can use these analytic linearizations instead of numerically perturbing the block. This is especially useful for blocks that contain discontinuities and do not give good results using numerical perturbation.

The following table lists the Simulink blocks that contain analytic Jacobians for linearization. For more information see the reference page for each block.

Analytic Block Jacobians

Block	Analytic Jacobian (Y/N)	Notes
Continuous Library		
Derivative	Y	Allows control of the time constant for the filter constant
Integrator	Y	Includes option to exclude saturation and resets from linearization
State-Space	Y	
Transfer Fcn	Y	
Transport Delay	Y	Allows control of Padé order
Variable Transport Delay	Y	Allows control of Padé order
Zero-Pole	Y	
Discontinuities Library		
Backlash	N	
Coulomb and Viscous Friction	N	
Dead Zone	Y	Includes option to treat as gain when linearizing
Dead Zone Dynamic	Y	
Hit Crossing	N	
Quantizer	Y	Includes option to treat as gain when linearizing
Rate Limiter	Y	Includes option to treat as gain when linearizing
Rate Limiter Dynamic	N	
Relay	N	

Analytic Block Jacobians (Continued)

Block	Analytic Jacobian (Y/N)	Notes
Saturation	Y	Includes option to treat as gain when linearizing
Saturation Dynamic	N	
Wrap to Zero	N	
Discrete Library		
Difference	Y	
Discrete Derivative	N	
Discrete Filter	Y	
Discrete State-Space	Y	
Discrete Transfer Fcn	Y	
Discrete Zero-Pole	Y	
Discrete-Time Integrator	Y	Includes option to ignore saturation and resets during linearization. Jacobian not supported for non-double data types.
First-Order Hold	N	
Integer Delay	N	
Memory	Y	Linearizes to a gain of 1 when driven by a continuous signal. Includes option to linearize to a Unit Delay when driven by a discrete signal.
Tapped Delay	N	
Transfer Fcn First Order	Y	
Transfer Fcn Lead or Lag	Y	

Analytic Block Jacobians (Continued)

Block	Analytic Jacobian (Y/N)	Notes
Transfer Fcn Real Zero	Y	
Unit Delay	Y	Jacobian does not support frame-based signals
Weighted Moving Average	N	
Zero-Order Hold	N	
Logic and Bit Operations Library		
Bit Clear	N	
Bit Set	N	
Bitwise Operator	N	
Combinatorial Logic	N	
Compare To Constant	N	
Compare To Zero	N	
Detect Change	N	
Detect Decrease	N	
Detect Fall Negative	N	
Detect Fall Nonpositive	N	
Detect Increase	N	
Detect Rise Nonnegative	N	
Detect Rise Positive	N	
Extract Bits	Y	
Interval Test	N	
Interval Test Dynamic	N	

Analytic Block Jacobians (Continued)

Block	Analytic Jacobian (Y/N)	Notes
Logical Operator	N	
Relational Operator	N	
Shift Arithmetic	N	
Lookup Tables Library		
Cosine	N	
Direct Lookup Table (n-D)	N	
Interpolation (n-D) using PreLookup	Y	
Lookup Table	N	
Lookup Table (2-D)	N	
Lookup Table (n-D)	N	
Lookup Table Dynamic	N	
PreLookup Index Search	Y	
Sine	N	
Math Operations Library		
Abs	Y	
Add	Y	
Algebraic Constraint	N	
Assignment	N	
Bias	Y	

Analytic Block Jacobians (Continued)

Block	Analytic Jacobian (Y/N)	Notes
Complex to Magnitude-Angle	N	
Complex to Real-Imag	N	
Divide	Y	
Dot Product	N	
Gain	Y	
Magnitude-Angle to Complex	N	
Math Function	N	
Matrix Concatenation	N	
MinMax	N	
MinMax Running Resettable	N	
Polynomial	N	
Product	Y	
Product of Elements	Y	
Real-Imag to Complex	N	
Reshape	N	
Rounding Function	N	
Sign	Y	Linearizes to `Inf` at zero, linearizes to zero otherwise
Sine Wave Function	N	
Slider Gain	Y	
Subtract	Y	

Analytic Block Jacobians (Continued)

Block	Analytic Jacobian (Y/N)	Notes
Sum	Y	
Sum of Elements	Y	
Trigonometric Function	N	
Unary Minus	N	
Weighted Sample Time Math	N	
Model Verification Library		
Assertion	N/A	Does not contain outputs
Check Discrete Gradient	N/A	Does not contain outputs
Check Dynamic Gap	N/A	Does not contain outputs
Check Dynamic Lower Bound	N/A	Does not contain outputs
Check Dynamic Range	N/A	Does not contain outputs
Check Dynamic Upper Bound	N/A	Does not contain outputs
Check Input Resolution	N/A	Does not contain outputs
Check Static Gap	N/A	Does not contain outputs
Check Static Lower Bound	N/A	Does not contain outputs
Check Static Range	N/A	Does not contain outputs
Check Static Upper Bound	N/A	Does not contain outputs
Model Wide Utilities Library		

Analytic Block Jacobians (Continued)

Block	Analytic Jacobian (Y/N)	Notes
Block Support Table	N/A	Does not contain outputs
DocBlock	N/A	Does not contain outputs
Model Info	N/A	Does not contain outputs
Time-Based Linearization	N/A	Does not contain outputs
Trigger-Based Linearization	N/A	Does not contain outputs
Ports and Subsystems Library		
Configurable Subsystem	N/A	Only the blocks within the subsystem are part of the linearization
Atomic Subsystem	N/A	Only the blocks within the subsystem are part of the linearization
CodeReuse Subsystem	N/A	Only the blocks within the subsystem are part of the linearization
Enable	N	
Enabled and Triggered Subsystem	N/A	Only the blocks within the subsystem are part of the linearization
Enabled Subsystem	N/A	Only the blocks within the subsystem are part of the linearization
For Iterator Subsystem	N/A	Only the blocks within the subsystem are part of the linearization
Function-Call Generator	N/A	Only the blocks within the subsystem are part of the linearization
Function-Call Subsystem	N/A	Only the blocks within the subsystem are part of the linearization
If	N	

Analytic Block Jacobians (Continued)

Block	Analytic Jacobian (Y/N)	Notes
If Action Subsystem	N/A	Only the blocks within the subsystem are part of the linearization
Inport	N/A	Does not contain outputs
Model	N	
Outport	N/A	Does not contain inputs
Subsystem	N/A	Only the blocks within the subsystem are part of the linearization
Switch Case	N	
Switch Case Action Subsystem	N/A	Only the blocks within the subsystem are part of the linearization
Trigger	N	
Triggered Subsystem	N/A	Only the blocks within the subsystem are part of the linearization
While Iterator Subsystem	N/A	Only the blocks within the subsystem are part of the linearization
Signal Attributes Library		
Data Type Conversion	Y	
Data Type Conversion Inherited	Y	
Data Type Duplicate	N/A	Does not contain outputs
Data Type Propagation	N/A	Does not contain outputs
Data Type Scaling Strip	Y	
IC	N	
Probe	N	

Analytic Block Jacobians (Continued)

Block	Analytic Jacobian (Y/N)	Notes
Rate Transition	Y	
Signal Conversion	Y	
Signal Specification	Y	
Weighted Sample Time	N	
Width	N	
Signal Routing Library		
Bus Assignment	Y	
Bus Creator	Y	
Bus Selector	Y	
Data Store Memory	N/A	Does not contain inputs or outputs
Data Store Read	Y	Linearizes to a gain of 1. Assumes that there is no delay between data store read and data store write.
Data Store Write	Y	Linearizes to a gain of 1. Assumes that there is no delay between data store read and data store write.
Demux	N/A	
Environment Controller	Y	
From	N/A	
Goto	N/A	
Goto Tag Visibility	N/A	
Index Vector	Y	
Manual Switch	Y	
Merge	N	

Analytic Block Jacobians (Continued)

Block	Analytic Jacobian (Y/N)	Notes
Multiport Switch	Y	
Mux	N/A	
Selector	Y	
Switch	Y	
Sources Library - N/A No Inputs		
Sinks Library - N/A No Outputs		
User Defined Functions Library		
Embedded MATLAB Function	N	
Fcn	N	
Level-2 M-File S-Function	N	
MATLAB Fcn	N	
S-Function	N	
S-Function Builder	N	
Additional Math and Discrete Library		
Fixed-Point State-Space	Y	
Transfer Fcn Direct Form II	N	
Transfer Fcn Direct Form II Time Varying	N	
Unit Delay Enabled	Y	
Unit Delay Enabled External IC	Y	

Analytic Block Jacobians (Continued)

Block	Analytic Jacobian (Y/N)	Notes
Unit Delay Enabled Resettable	Y	
Unit Delay Enabled Resettable External IC	Y	
Unit Delay External IC	Y	
Unit Delay Resettable	Y	
Unit Delay Resettable External IC	Y	
Unit Delay With Preview Enabled	Y	
Unit Delay With Preview Enabled Resettable	Y	
Unit Delay With Preview Enabled Resettable External RV	Y	
Unit Delay With Preview Resettable	Y	
Unit Delay With Preview Resettable External RV	Y	
Decrement Real World	Y	
Decrement Stored Integer	Y	
Decrement Time To Zero	Y	
Decrement To Zero	Y	

Analytic Block Jacobians (Continued)

Block	Analytic Jacobian (Y/N)	Notes
Increment Real World	Y	
Increment Stored Integer	Y	

Several of these blocks include options to control the linearization that you can adjust 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. The **Notes** column of the table above gives details on blocks that include these options.

Note The preprogrammed, analytic block linearizations are only used in block-by-block analytic linearization. When using numerical-perturbation linearization, these blocks will be numerically perturbed along with the rest of the system.

Block Perturbation

When a preprogrammed block linearization cannot be used, Simulink Control Design will compute 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.

The block perturbation algorithm involves introducing a small perturbation to the nonlinear block 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 of this block. Changing the size of the perturbations will change the resulting linearized model.

As described in “Linearizing Models”, a nonlinear Simulink block can be written as a state-space system:

$$\dot{x}(t) = f\left(x(t), u(t), t\right)$$
$$y(t) = g\left(x(t), u(t), t\right)$$

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:

$$\delta x(t) = x(t) - x_0$$
$$\delta u(t) = u(t) - u_0$$
$$\delta y(t) = y(t) - y_0$$

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:

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

The state-space matrices A, B, C, and D of this linearized model represent the Jacobians of the block, as defined in "Linearizing Models". 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

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

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

where

- $x_{p,i}$ is the state vector whose ith 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 ith 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 blocks are computed in a similar way. For more information, see "Linearizing Models" in the Getting Started with Simulink Control Design documentation 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.

Changing the size of the perturbations will change 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. 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')
   ```

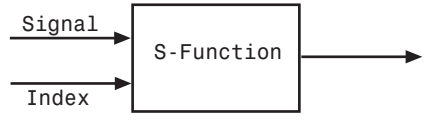
2. Get the inport

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

3. Set the perturbation level for this inport

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

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. Since you probably do not want to perturb the index signal, you can assign a perturbation level of zero to this inport.



Block Containing Two Inports

Numerical-Perturbation Linearization

An alternative linearization method available for use in Simulink Control Design 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 "Linearization Algorithms" on page 7-9, 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 7-30

Note The numerical perturbation linearization algorithm is the only linearization algorithm that works with models that contain references to other models using the Model block.

Invoking Numerical-Perturbation Linearization

Prior to Simulink 3.0, numerical-perturbation linearization was the only linearization method available with Simulink. 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 "Linearizing Models", a nonlinear Simulink model can be written as a state-space system:

$$\dot{x}(t) = f\left(x(t)u(t),t\right)$$
$$y(t) = g\left(x(t)u(t),t\right)$$

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:

$$\delta x(t) = x(t) - x_o$$
$$\delta u(t) = u(t) - u_o$$
$$\delta y(t) = y(t) - y_o$$

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:

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

The state-space matrices A, B, C, and D of this linearized model represent the Jacobians of the system, as defined in "Linearizing Models". 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

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

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

where

- $x_{p,i}$ is the state vector whose ith 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 ith 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 $x_{\mathrm{p,i}}$, u_{o}.
- $y|_{u_{p,i}}$ is the value of y at $u_{\mathrm{p,i}}$, 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 Models" in the Getting Started with Simulink Control Design documentation.

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?" in the Getting Started with Simulink Control Design documentation 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:

1. 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.

2 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**.

3 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.

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 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: Command-Line Numerical Perturbation Linearization of a Model Reference Model

Simulink Control Design supports linearization of models that contain references to other models, using the Model block, however you must use the Numerical Perturbation linearization algorithm to linearize these models. The following example illustrates how to linearize a model reference model at the MATLAB command line using numerical perturbation. For information on linearizing a model reference model using the Simulink Control Design graphical interface, see "Example: Using the Graphical Interface for Numerical Perturbation Linearization of a Model Reference Model" on page 7-34.

1 Open the model.

In this example, we will use the `scdairframe_reference.mdl` model, included with Simulink Control Design. The model uses a Model block to reference another Simulink model, `eom.mdl`, hence numerical perturbation is the only linearization algorithm that you can use with this model.

Enter

```
scdairframe_reference
```

at the MATLAB command line 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. When 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 that in this case 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 doing this, depending on the model you are using and the information you have about the operating point. For more information on creating operating points, see "Specifying Operating Points" in the Getting Started with Simulink Control Design

documentation. For the purposes of this example, to illustrate the use of the numerical perturbation linearization algorithm, we will simply 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 create an operating point object, edit the state values within this object, and 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 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: Using the Graphical Interface for Numerical Perturbation Linearization of a Model Reference Model

In the previous example, a model reference model, `scdairframe_reference.mdl`, was linearized using Simulink Control Design functions for numerical perturbation. In the following example, we will use the numerical perturbation algorithm to linearize the same model within the Control and Estimation Tools Manager graphical interface.

1. Open the model.

   In this example, we will use the `scdairframe_reference.mdl` model, included with Simulink Control Design. The model uses a Model block to reference another Simulink model, `eom.mdl`, hence numerical perturbation is the only linearization algorithm that you can use with this model.

   Enter

   ```
   scdairframe_reference
   ```

   at the MATLAB command line 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. When 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 that in this case the `scdairframe_reference` model already contains one Inport block and two Outport blocks.

3. Open a linearization task for the model in the Control and Estimation Tools Manager.

Within 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. Note that the Control and Estimation Tools Manager displays a warning dialog to inform you that it has automatically selected the numerical perturbation linearization algorithm for the model. Click the **OK** button the close this dialog.

You should also 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**.

4 Create an operating point object for the model.

There are several possible methods for doing this, depending on the model you are using and the information you have about the operating point. For more information on creating operating points, see "Specifying Operating Points" in the Getting Started with Simulink Control Design documentation. For the purposes of this example, to illustrate the use of the numerical perturbation linearization algorithm, we will skip this step and use the default operating point for the linearization.

5 Specify the linearization algorithm

Since you can only linearize the `scdairframe_reference.mdl` model using the numerical perturbation algorithm, the Control and Estimation Tools Manager selected this algorithm automatically when the linearization task was created. To select numerical perturbation linearization as the algorithm for a model that does not use model references, 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 we have not explicitly specified the **Input perturbation level**, the algorithm will still use the **Relative perturbation level** setting to compute input perturbations.

Note that 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

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. An alternative solution is to 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 inside models that reference other models 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.

Handling Feedback Loops

"Performing Open-Loop Analysis" in the Getting Started with Simulink Control Design documentation 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.

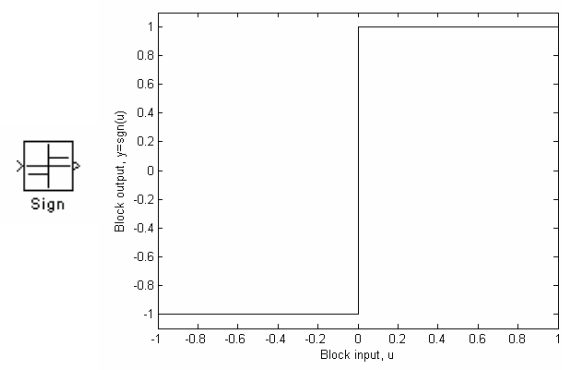
Recommendations for Computing Operating Points and Creating Accurate Linearized Models

Particular blocks and modeling situations in Simulink can sometimes cause difficulties with computing operating points (trimming) and linearization. However, by understanding what it means to trim or linearize a Simulink model and by using the correct modeling techniques, you can create accurate operating points and linearized models 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 and linearized models, with recommendations for ways to avoid these situations. The examples focus on the following modeling situations:

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

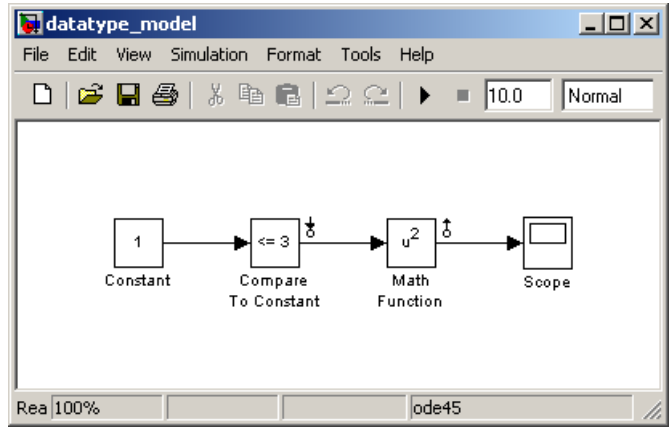
- Relational Operator blocks

- 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)

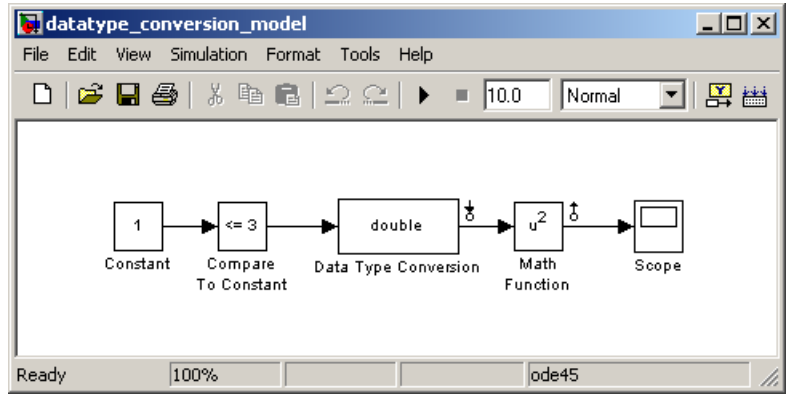
Non-Double Data Types

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

To work around this problem, you can use a Data Type Conversion block, which does have 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, the input signal to the Square block is a double, and the linearized model gives the correct response of 2.



Overriding Non-Double Data Types

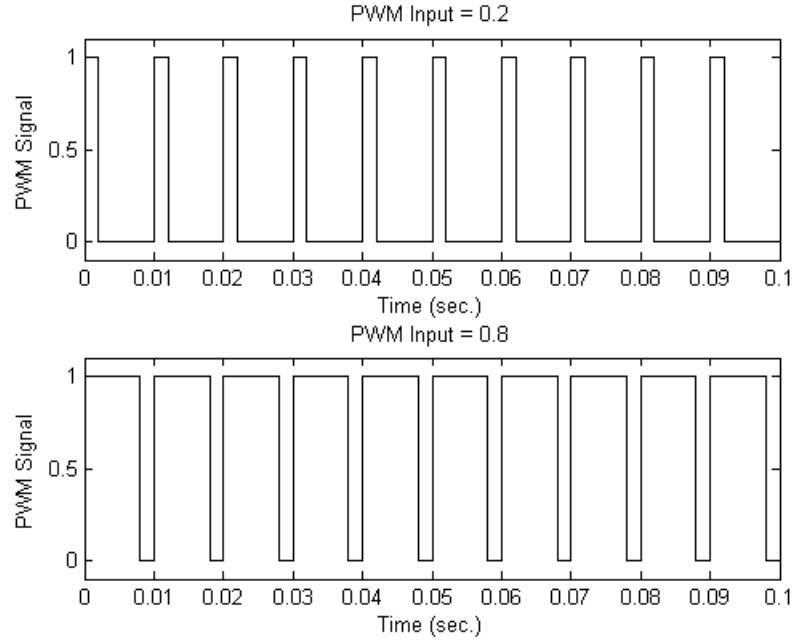
When linearizing a model that contains non-double data types but still runs correctly in all double precision, you can choose to override all data types with doubles. To do this, in the model window select **Tools > Fixed-Point Settings** from the menu. This opens the Fixed-Point Settings window. Within this window select **True doubles** from the **Data type override** menu. When linearizing and simulating the model, it now 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.

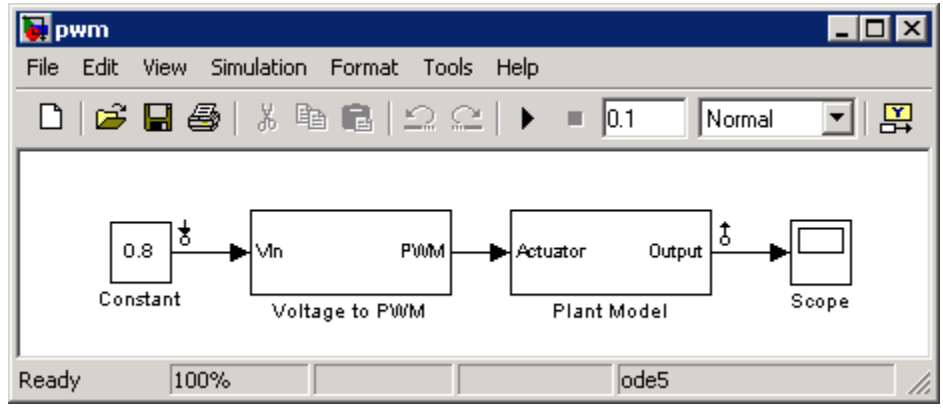
Pulse Width Modulation

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%.

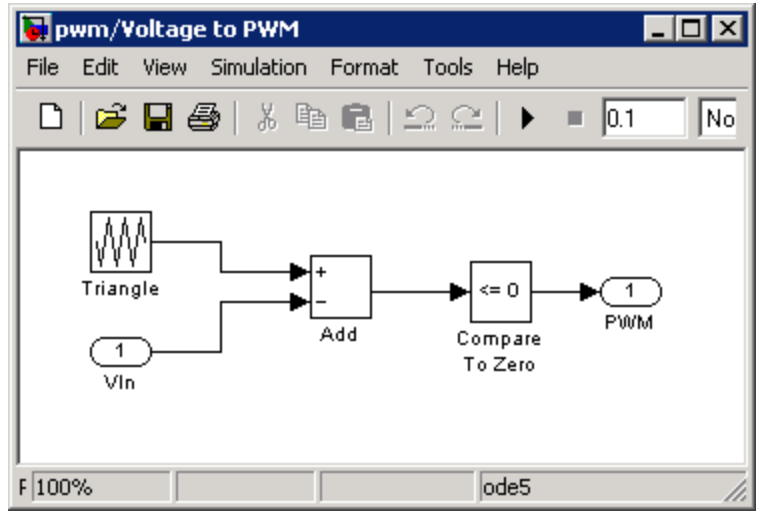


The model, `pwm.mdl`, shown below, converts a constant signal to a PWM signal.



When linearizing a model containing PWM signals there are two effects you should take into account:

- The signal level at the operating point will be one of the discrete values within the PWM signal, not the DC signal value. For example, in the model above, the signal level will be either 0 or 1, not 0.8. This change in operating point will affect the linearized model.
- The creation of the PWM signal within the subsystem `Voltage to PWM`, shown below, uses a comparator block, the Compare to Zero block. Comparator blocks do not linearize well due to their discontinuities and the non-double outputs.



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

Transport Delay, Memory, and Other Blocks with Non-Trimmable States

Blocks with non-double, discrete states cannot accurately be used to compute operating points from design specifications (also called trimming) because these special states cannot be seen by the `findop` function. Blocks that contain these states include

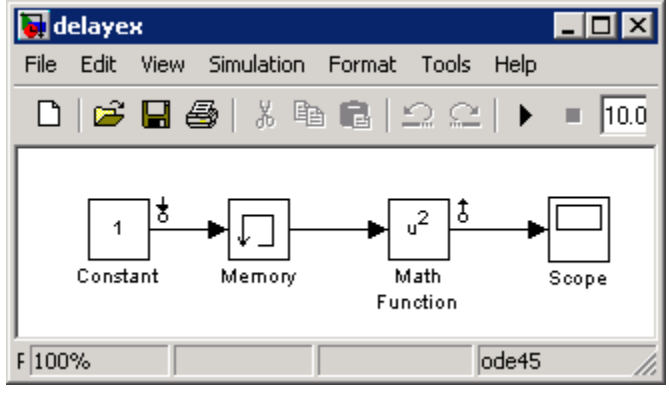
- Action Subsystem blocks which are not enabled
- Backlash block
- Embedded MATLAB Function block with persistent data
- Transport Delay and Variable Transport Delay blocks
- Memory block
- Rate Transition block
- Stateflow blocks
- S-Function blocks with states not registered as Continuous or Double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

To determine when your model contains any of these blocks with states that cannot be trimmed, run the following command

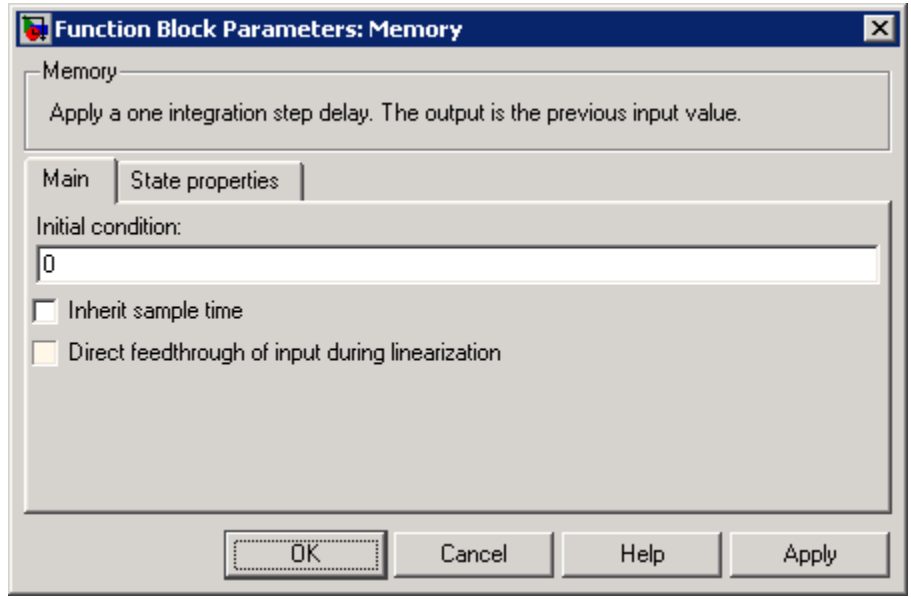
```
sldiagnostics('modelname','CountBlocks')
```

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

When you have Memory blocks or Variable Transport Delay/Transport Delay blocks in your model, you can properly configure the initial outputs of these blocks so that trim or linearization uses the correct output value. 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. To create an accurate operating point or a linearized model that is based on this more accurate operating point, 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 trim or linearization.



The problem of block output during trim or linearization also occurs for the Backlash block although, in this case, the block does not have a direct feedthrough option. Extra care should be taken when linearizing a model containing Backlash blocks.

For other blocks with states that are not seen by trim, you should consider removing them from the model while trimming. If the output of the block does not effect any of the state derivatives or desired output levels downstream it does not pose a problem for trim and you do not need to remove it. If the block

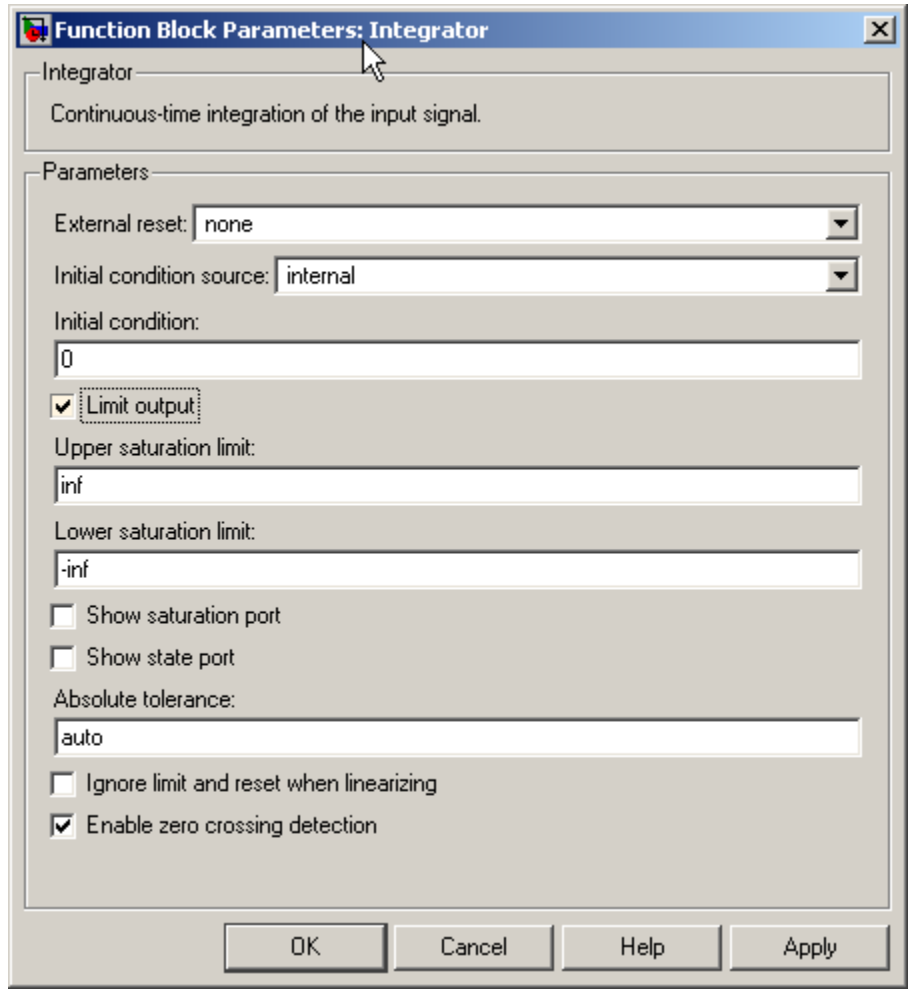
does have downstream impact, consider replacing it using a configurable subsystem when trimming.

Transport Delay Blocks

Transport Delay blocks can cause additional problems when you use the Padé approximation option for linearization within a multirate model. In this case, the discretization of the Padé approximation has a frequency response that does not match the frequency response of the original Transport Delay. This can lead to linearized models which do not behave as expected. A solution to this problem is to discretize the transport delay first, using the Model Discretizer, and then linearize the model using the Padé approximation. See "Model Discretizer" in the Simulink documentation for more information on using the Model Discretizer.

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.



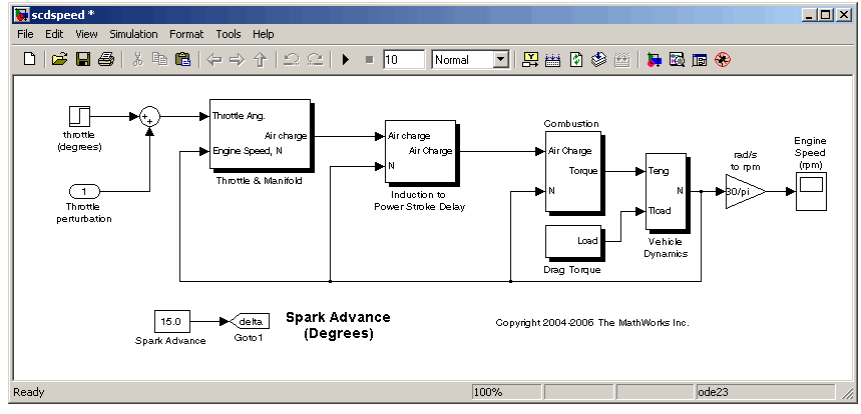
Event-Based Models and Triggered Subsystems

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

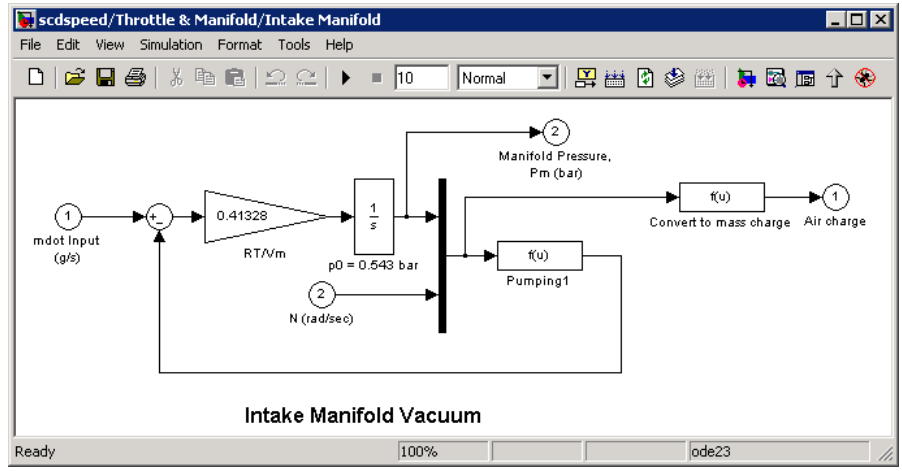
operating point, this information is not captured by the list of values of states and inputs that typically describe the operating point for linearization.

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 Simulink.

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 Simulink Control Design and 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.



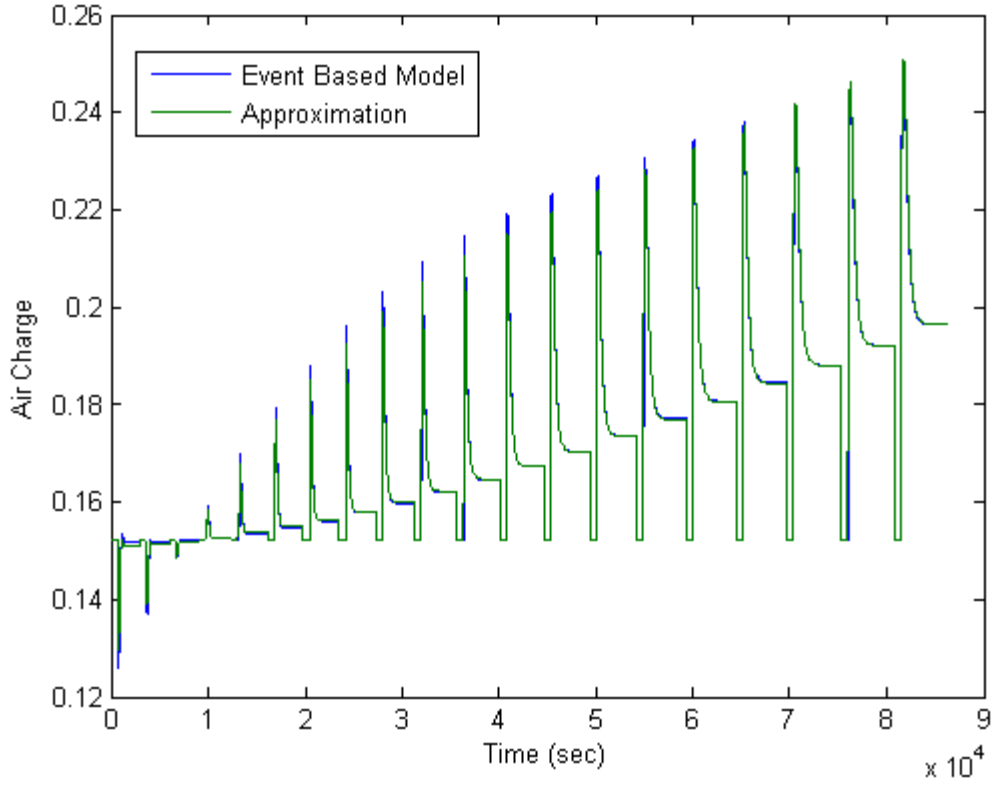
The approximation has the following form:

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

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 Simulink Parameter Estimation to compute the unknown parameters. This method is outlined in the recorded webinar,"New & Upgraded Simulink Tools for Developing More Accurate Models & Better Control Systems". The webinar also contains a demonstration of the linearization of this model and the use of 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.



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 "Creating Operating Points from Simulation" in the online Getting Started with Simulink Control Design documentation for more information) or using the `findop` function (see "Extracting Values from Simulation" on page 5-15 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 2-3.

8

Functions — By Category

Linearization Analysis I/Os

`get`	Properties of linearization I/Os and operating points
`getlinio`	Linearization I/O settings for Simulink model
`linio`	Construct linearization I/O settings for Simulink model
`set`	Set properties of linearization I/Os and operating points
`setlinio`	Assign I/O settings to Simulink model

Operating Points

`addoutputspec`	Add output specification to operating point specification
`copy`	Copy operating point or operating point specification
`findop`	Find operating points from specifications or simulation
`get`	Properties of linearization I/Os and operating points
`getinputstruct`	Input structure from operating point
`getstatestruct`	State structure from operating point
`getxu`	States and inputs from operating points
`initopspec`	Initialize operating point specification values
`operpoint`	Create operating point for Simulink model
`operspec`	Create operating point specifications for Simulink model
`set`	Set properties of linearization I/Os and operating points
`setxu`	Set states and inputs in operating points
`update`	Update operating point object with structural changes in model

Linearization

`findop`	Find operating points from specifications or simulation
`getlinio`	Linearization I/O settings for Simulink model
`getlinplant`	Compute open-loop plant model from Simulink diagram
`linearize`	Create linearized model from Simulink model
`linio`	Construct linearization I/O settings for Simulink model
`linoptions`	Set options for linearization and finding operating points
`operpoint`	Create operating point for Simulink model
`operspec`	Create operating point specifications for Simulink model

9

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 2-6.

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 will automatically compile 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 returns the object `op`. Note that there are no outports 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)
```

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:
-----------
```

```
(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
```

MATLAB® displays the final output specification.

```
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 creates a pointer to `op_point1`. In this case any changes made to `op_point2` will also be made to `op_point1`.

Examples

Create an operating point object for the model, `magball`.

```
opp=operpoint('magball')
```

MATLAB displays the operating point.

```
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)
```

MATLAB displays an exact copy of the object.

```
    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`

Purpose

Find operating points from specifications or simulation

Syntax

```
[op_point,op_report]=findop('model',op_spec)
[op_point,op_report]=findop('model',op_spec,options)
op_point=findop('model',times)
```

Graphical Interface

As an alternative to the `findop` function, create operating points from specifications or simulation within the **Operating Points** node of the Simulink Control Design GUI. For more information on creating operating points, see "Creating Operating Points from Specifications" and "Creating Operating Points from Simulation" in the Getting Started with Simulink Control Design documentation.

Remarks

Finding operating points from specifications using the `findop` function is the same as trimming, or performing trim analysis. Use the `findop` function instead of the Simulink `trim` function when working with Simulink Control Design operating point objects and specification objects.

Description

`[op_point,op_report]=findop('model',op_spec)` finds an operating point, `op_point`, of the model, `'model'`, from specifications given in `op_spec`.

`[op_point,op_report]=findop('model',op_spec,options)` finds an operating point, `op_point`, of the model, `'model'`, from specifications given in `op_spec`. Several options for the optimization are specified in the `options` object, which you can create with the function `linoptions`.

The input to `findop`, `op_spec`, is an operating point specification object. Create this object with the function `operspec`. Specifications on the operating points, such as minimum and maximum values, initial guesses, and known values, are specified by editing `op_spec` directly or by using `get` and `set`. To find equilibrium, or steady-state, operating points, set the `SteadyState` property of the states and inputs in `op_spec` to `1`. The `findop` function uses optimization to find operating points that closely meet the specifications in `op_spec`. By default, `findop` uses

the optimizer `graddescent_elim`. To use a different optimizer, change the value of `OptimizerType` in `options` using the `linoptions` function.

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.

For all syntaxes, the output of `findop` is 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 9-8
- "Inputs" on page 9-9
- "Time" on page 9-9

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:

`Nx`	Number of states in the block. This property is read-only.
`Block`	Block that the states are associated with
`x`	Vector containing the values of states in the block
`Ts`	Vector containing the sample time and offset for the state
`SampleType`	`CSTATE` for a continuous state or `DSTATE` for a discrete state
`inReferencedModel`	`1` when the state is inside a referenced model or `0` when it is not
`Description`	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 that the input vector is associated with
`PortWidth`	Width of the corresponding inport
`u`	Vector containing the input level at the operating point
`Description`	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 the Optimization Toolbox, 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 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
```

MATLAB 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 successully 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 report above, 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, you can use `findop` in the following way.

```
op_point=findop('magball',[10,20])
```

This 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 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 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

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" and "Specifying Operating Points" in the Getting Started with Simulink Control Design documentation.

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
```

Since op is a structure, you can also view any properties or fields using dot-notation, as in this example.

```
W=op.States
```

This 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 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 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”.

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`	`'on'` when the I/O will be used for linearization and `'off'` otherwise
`Block`	Name of the block the I/O is associated with
`OpenLoop`	`'on'` when the feedback loop at the I/O is open and `'off'` when it is closed
`PortNumber`	Integer referring to the output port the I/O is associated with
`Type`	Linearization I/O type • `'in'`: linearization input point • `'out'`: linearization output point • `'outin'`: linearization output then input point • `'inout'`: linearization input then output point
`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 returns a formatted display of the linearization I/Os.

```
   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:
 - 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), and 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`

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 the operating point is found 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 shown below.

```
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

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 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 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 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.

`[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`. The ordering of states in `x`, and inputs in `u`, is the same as that used by 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. The ordering of states in `x` and `xstruct`, and inputs in `u`, is the same as that used by 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 the ordering that is compatible with Simulink, along with inputs and a state structure, type

```
[x,u,xstruct]=getxu(op)
```

This 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 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`

Purpose

Initialize operating point specification values

Syntax

```
opnew=initopspec(opspec,oppoint)
opnew=initopspec(opspec,x,u)
opnew=initopspec(opspec,xstruct,u)
```

Graphical Interface

As an alternative to the `initopspec` function, initialize operating point specification values in the **Create Operating Points** pane in the **Operating Points** node within the Simulink Control Design GUI. See "Creating Operating Points from Specifications" in the Getting Started with Simulink Control Design documentation.

Description

`opnew=initopspec(opspec,oppoint)` initializes the operating point specification object, `opspec`, with the values contained in the operating point object, `oppoint`. The function returns a new operating point specification object, `opnew`. Create `opspec` with the function `operspec`. Create `oppoint` with the function `operpoint` or `findop`.

`opnew=initopspec(opspec,x,u)` initializes the operating point specification object, `opspec`, with the values contained in the state vector, `x`, and the input vector, `u`. The function returns a new operating point specification object, `opnew`. Create `opspec` with the function `operspec`. You can use the function `getxu` to create `x` and `u` with the correct ordering.

`opnew=initopspec(opspec,xstruct,u)` initializes the operating point specification object, `opspec`, with the values contained in the state structure, `xstruct`, and the input vector, `u`. The function returns a new operating point specification object, `opnew`. Create `opspec` with the function `operspec`. You can use the function `getstatestruct` or `getxu` to create `xstruct` and the function `getxu` to create `u` with the correct ordering. Alternatively, `xstruct`, can be saved to the MATLAB workspace after a simulation of the model. See the Simulink documentation for more information on these structures.

Example

Create on operating point using `findop` by simulating the `magball` model and extracting the operating point after `20` time units.

```
oppoint=findop('magball',20)
```

This 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',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);
```

Graphical Alternative

As an alternative to the `linearize` function, create linearized models using the **Linearization Task** node of the Simulink Control Design GUI. See “Linearizing the Model”.

Description

`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 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`. 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.

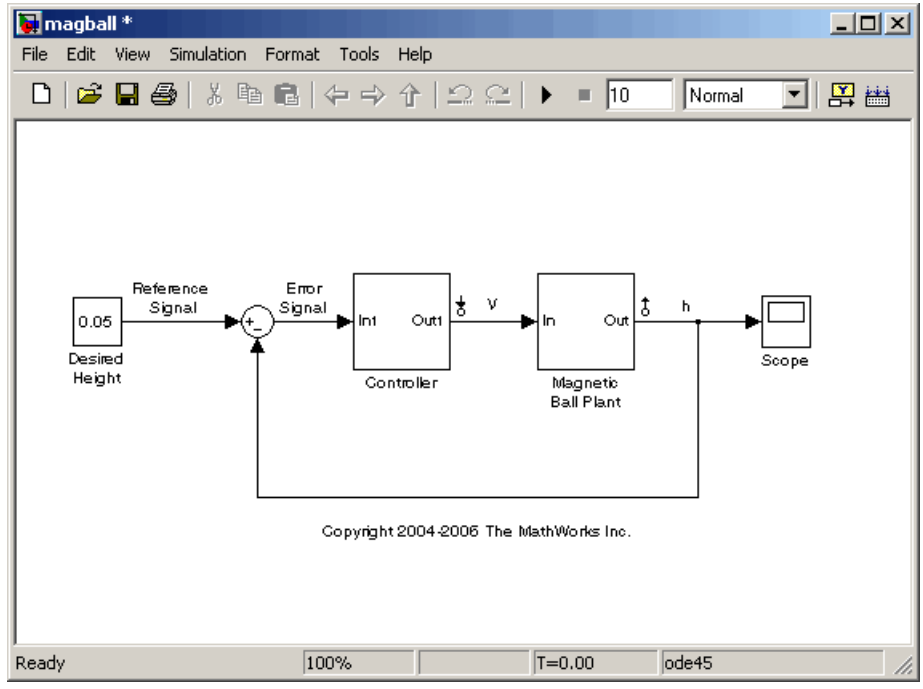
[lin,op] = linearize('sys',snapshottimes); creates operating points for the linearization by simulating the model, 'sys', and taking snapshots of the system's states and inputs at the times given in the vector snapshottimes. The function returns lin, a set of linear time-invariant state-space models 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 will use 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.

Algorithms

Linearization algorithm options are set with the function linoptions and passed to the function linearize as an optional argument.

Examples

Open the Simulink model, `magball`, and insert linearization annotations as shown in the following figure:



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          h
   Controller               0             0             0
   Current                -50          -100             0
   dhdt                     0        -2.801             0
   height                   0             0             1


b =
                 magball/Cont
   Controller               0
   Current                 50
   dhdt                     0
   height                   0


c =
                 Controller     Current      dhdt      height
   Magntic Bal            0           0         0           1


d =
                 magball/Cont
   Magnetic Bal             0

Continuous-time model.
```

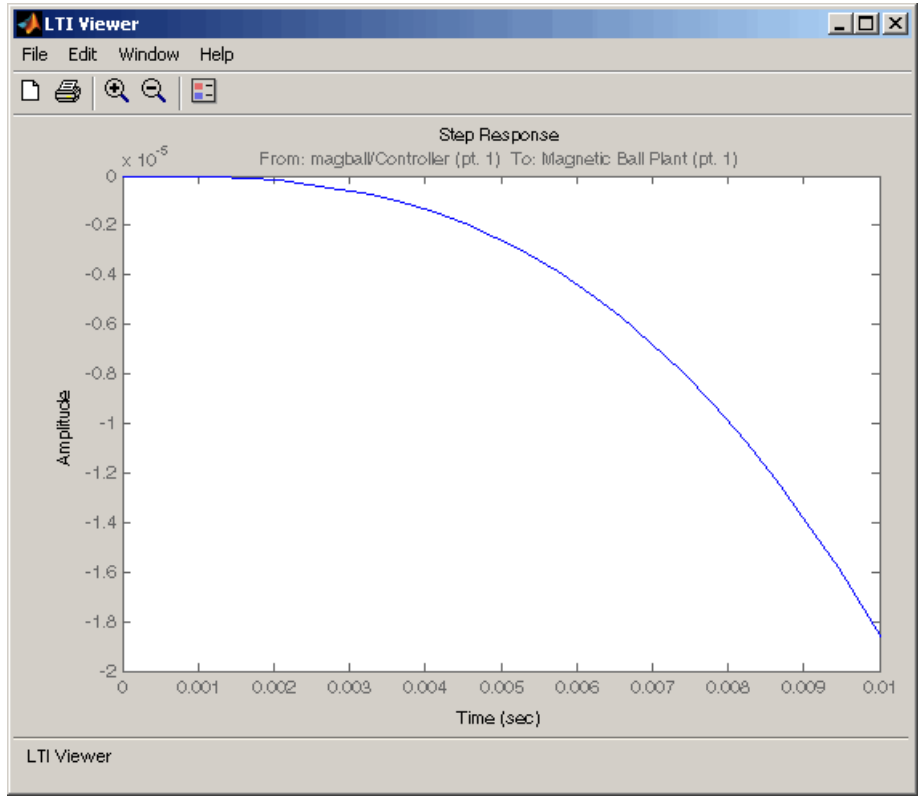
The matrices, `a`, `b`, `c`, and `d` are the state-space matrices of the linear system given by the following equations

$$\dot{x}(t) = ax(t) + bu(t)$$
$$y(t) = cx(t) + du(t)$$

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

```
ltiview(lin)
```

which produces the following plot:



See Also

`findop`, `getlinio`, `operpoint`, `operspec`, `linio`, `linoptions`, `ltiview`

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".

Description

`io=linio('blockname',portnum)` creates a linearization I/O object for the signal that originates from the outport 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 outport 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 outport 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 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.

```
    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:
```

```
    - An Output Measurement
    - A Loop Opening
```

See Also

getlinio, linearize, setlinio

linoptions

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.

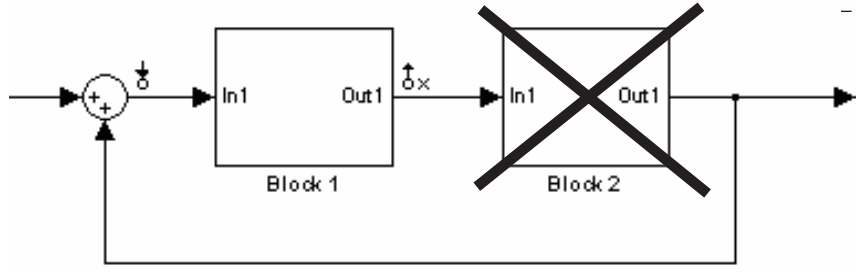
`opt=linoptions('Property1','Value1','Property2','Value2',...)` creates a linearization options object, `opt`, in which the option given by `Property1` is set to the value given in `Value1`, the option given by `Property2` is set to the value given in `Value2`, etc. The variable, `opt`, is passed to the functions `findop` and `linearize` to specify options for finding operating points and linearization.

The following options can be set with `linoptions`:

`LinearizationAlgorithm`	Set to `'numericalpert'` (default is `'blockbyblock'`) to enable numerical-perturbation linearization (as in Simulink 3.0) where root-level inports and states are numerically perturbed. Linearization annotations are ignored and root level inports and outports are used instead. The `'numericalpert'` option is the only linearization algorithm that works with models that contain references to other models using the Model block.
`SampleTime`	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.

`UseFullBlockNameLabels` 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.

`BlockReduction` Set to `'on'` (default) to eliminate from the linearized model, blocks that are not in the path of the linearization, as in the following figure. Set to `'off'` to include these blocks in the linearized model.



`IgnoreDiscreteStates` Set to `'on'` to remove any discrete states from the linearization. Set to `'off'` (default) to include discrete states.

`RateConversionMethod` Set to `'zoh'` (default) to use the zero order rate conversion routine when linearizing a multirate system. Set to `'tustin'` to use the Tustin (bilinear) method. Set to `'prewarp'` to use the Tustin approximation with prewarping.

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 or see the

	"Continuous/Discrete Conversion of LTI Models" in the Control System Toolbox documentation.
`PreWarpFreq`	The critical frequency Wc (in rad/sec) used by the `'prewarp'` option when linearizing a multirate system.
`NumericalPertRel`	Set the perturbation level for obtaining the linear model (default value is `1e-5`). The perturbation of the system's states is specified by $\text{NumericalPertRel} + 10^{-3} \times \text{NumericalPertRel} \times \lvert x \rvert$ The perturbation of the system's inputs is specified by $\text{NumericalPertRel} + 10^{-3} \times \text{NumericalPertRel} \times \lvert u \rvert$
`NumericalXPert`	Individually set the perturbation levels for the system's states using an operating point object. Use the `operpoint` function to create an operating point object for the model.
`NumericalUPert`	Individually set the perturbation levels for the system's inputs using an operating point object. Use the `operpoint` function to create an operating point object for the model.
`OptimizationOptions`	Set options for use with the optimization algorithms. These options are the same as those set with `optimset`. See the Optimization Toolbox documentation for more information on these algorithms. If you do not have the Optimization Toolbox, you can access the documentation at `http://www.mathworks.com/access/helpdesk/help/toolbox/optim/optim.shtml`
`OptimizerType`	Set optimizer type to be used by trim optimization if the Optimization Toolbox is installed. The available optimizer types are

- `graddescent_elim`, the default optimizer, based on the Optimization Toolbox function `fmincon`, enforces an equality constraint to force time derivatives of states to be zero (`dx/dt=0`, `x(k+1)=x(k)`) and constraints on output signals. This optimizer fixes states, `x`, and inputs, `u`, by not allowing these variables to be optimized.
- `graddescent`, enforces an equality constraint to force time derivatives of states to be zero (`dx/dt=0`, `x(k+1)=x(k)`) and constraints on output signals. Minimize the error between the desired (known) values of states, `x`, inputs, `u`, and outputs, `y`. If there are no constraints on `x`, `u`, or `y`, `findop` will attempt to minimize the deviation between the initial guesses for `x` and `u` and the trimmed values.
- `lsqnonlin` fixes states, `x`, and inputs, `u`, by not allowing these variables to be optimized. The algorithm then tries to minimize the error in `dx/dt` and outputs, `y`.
- `simplex` uses the same cost function as `lsqnonlin` with the `fminsearch` optimization routine.

See the Optimization Toolbox documentation for more information on these algorithms. If you do not have the Optimization Toolbox, you can access the documentation at `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`

operpoint

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 "Specifying Operating Points" in the Getting Started with Simulink Control Design documentation.

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 9-44
- "States" on page 9-44
- "Inputs" on page 9-45
- "Time" on page 9-45

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:

`Nx`	Number of states in the block. This property is read-only.
`Block`	Block that the states are associated with
`x`	Vector containing the values of states in the block
`Ts`	Vector containing the sample time and offset for the state
`SampleType`	`CSTATE` for a continuous state or `DSTATE` for a discrete state
`inReferencedModel`	`1` when the state is inside a referenced model or `0` when it is not
`Description`	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 that the input vector is associated with
`PortWidth`	Width of the corresponding inport
`u`	Vector containing the input level at the operating point
`Description`	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

```
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 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 "Creating Operating Points from Specifications" in the Getting Started with Simulink Control Design documentation.

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

Use the `set` function to edit the properties of this object before running `findop`.

Model

`Model` is the name of the Simulink model that this operating point specification object is associated with.

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 that the states are associated with
`x`	Vector containing values of states in the block. Set the corresponding value of `Known` to `1` when these values are known operating point values. Set the corresponding values of `Known` to `0` when these values 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`	`CSTATE` for a continuous state or `DSTATE` for a discrete state
`inReferencedModel`	`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 themselves 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 of `SteadyState` 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`	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 that the input vector is associated with
`PortWidth`	Width of the corresponding inport
`u`	Vector containing values of inputs. Set the corresponding value of `Known` to `1` when these values are known operating point values. Set the corresponding values of `Known` to `0` when these values 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 themselves in the `u` 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`	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 outport block in the Simulink

model. To constrain additional outputs, use the `addoutputspec` function to add an another output specification to the operating point specification object. The `Outputs` object has the following properties:

`Block`	Outport block that the output vector is associated with
`PortWidth`	Width of the corresponding outport
`PortNumber`	Port number that the output is associated with
`y`	Vector containing values of outputs. Set the corresponding value of `Known` to `1` when these values are known operating point values. Set the corresponding values of `Known` to `0` when these values are initial guesses for the operating point values.
`Known`	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 themselves in the `y` property.
`Min`	Vector containing the minimum values of the corresponding output's operating point
`Max`	Vector containing the maximum values of the corresponding output's operating point
`Description`	String describing the output

Example

To create an operating point specification object for the Simulink model `magball`, type

```
op_spec = operspec('magball')
```

which returns

```
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
```

MATLAB 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, default operating point values and initial guesses (based on initial conditions of the states), and 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 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`

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" and "Specifying Operating Points" in the Getting Started with Simulink Control Design documentation.

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 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
```

Since `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`.

```
set(op_cond.States(2))
ans =
                    Nx: {}
                 Block: {}
                     x: {}
                    Ts: {}
            SampleType: {}
    inReferencedModel: {}
           Description: {}
```

Now, use dot-notation to set the `x` property to `8`.

```
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`

Purpose

Assign I/O settings to Simulink model

Syntax

```
oldio=setlinio('sys',io)
```

Graphical Interface

As an alternative to the `setlinio` function, edit 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”.

Description

`oldio=setlinio('sys',io)` assigns the settings in the vector of linearization I/O objects, `io`, to the Simulink model, `sys`, where they are represented by annotations on the signal lines. Use the function `getlinio` or `linio` 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.

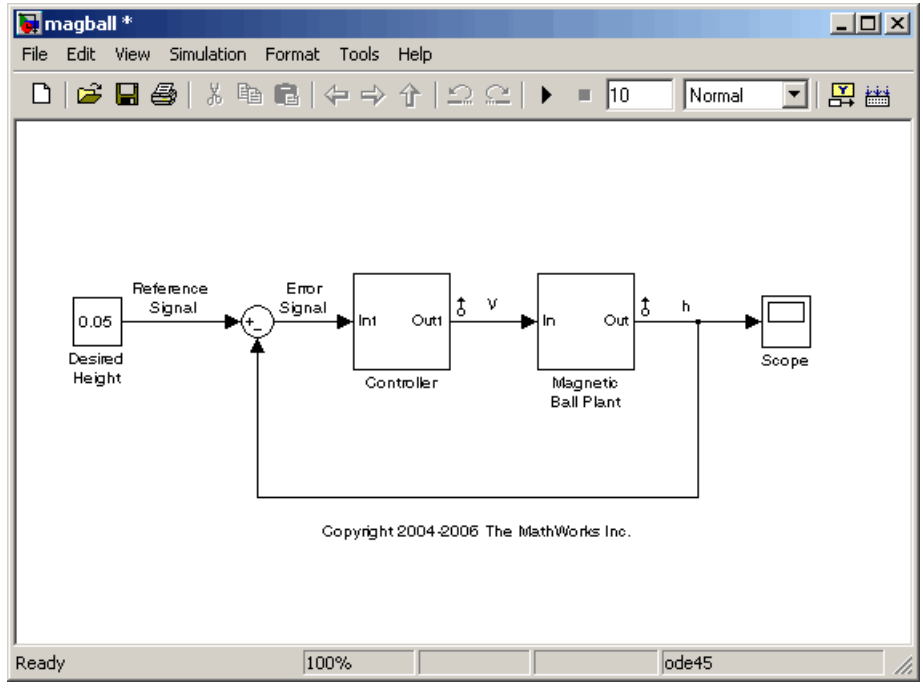
Examples

Before assigning I/O settings to a Simulink model using `setlinio`, you must create a vector of I/O objects representing linearization annotations, such as input points or output points, on 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 > 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. Right-click the signal line after the Magnetic Ball Plant. Select **Linearization Points > Output Point** from the menu to place another output point on this signal line. The model diagram should now look like that in the following figure:



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.

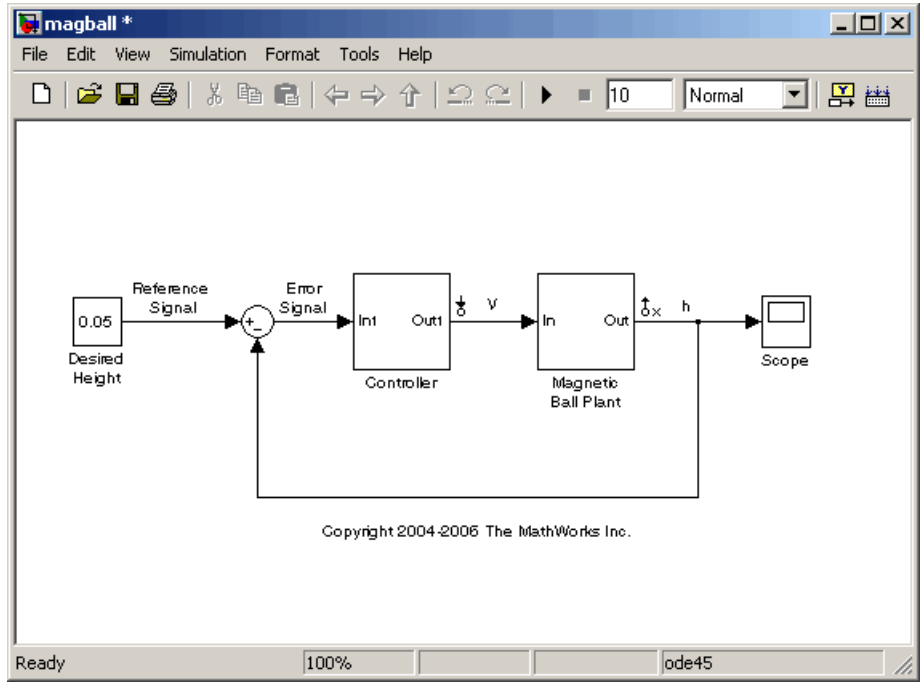
```
oldio=setlinio('magball',io)
```

This 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 like that in the following figure:



See Also

`get`, `getlinio`, `linio`, `set`

Purpose

Set states and inputs in operating points

Syntax

```
op_new=setxu(op_point,x,u)
```

Graphical Alternative

As an alternative to the `setxu` function, set states and inputs of operating points with the Simulink Control Design GUI. See "Importing Operating Points" on page 2-3 for more information.

Description

`op_new=setxu(op_point,x,u)` sets the states and inputs in the operating point, `op_point`, with the values in `x` and `u`. A new operating point containing these values, `op_new`, is returned. The variable `x` can be a vector or a structure with the same format as those returned from a Simulink simulation. The variable `u` can be a vector. Both `x` and `u` can be extracted from another operating point object with the `getxu` function.

Example

Open the Simulink model F14 by typing `f14` at the command line. Select **Simulation > Configuration Parameters > Data Import/Export**. In the **Save to workspace** pane, select **Final states**. In the **Save options** pane, select `Structure` from **Format**. This will save the final states of the model to the workspace after a simulation.

Start the simulation. After it has run, a new variable, `xFinal`, should be in the workspace. This variable is a structure with two properties, `time` and `signals`.

Create an operating point object for F14 by typing

```
op_point=operpoint('f14')
```

Note that all states are initially set to `0`. Set the states in this object to be the values in `xFinal`. Set the input to be `9`.

```
newop=setxu(op_point,xFinal,9)
```

The new operating point is displayed

```
Operating Point for the Model f14.
 (Time-Varying Components Evaluated at time t=0)
```

```
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

Purpose

Update operating point object with structural changes in model

Syntax

```
update(op)
```

Graphical Alternative

As an alternative to the `update` function, update operating point objects using the **Sync with Model** button in the Simulink Control Design GUI. See "Specifying Operating Points" in the Getting Started with Simulink Control Design documentation for more information.

Description

`update(op)` updates an operating point object, `op`, to reflect any changes in the associated Simulink model, such as states being added or removed.

Example

Open the magball model:

```
magball
```

Create an operating point object for the model:

```
op=operpoint('magball')
```

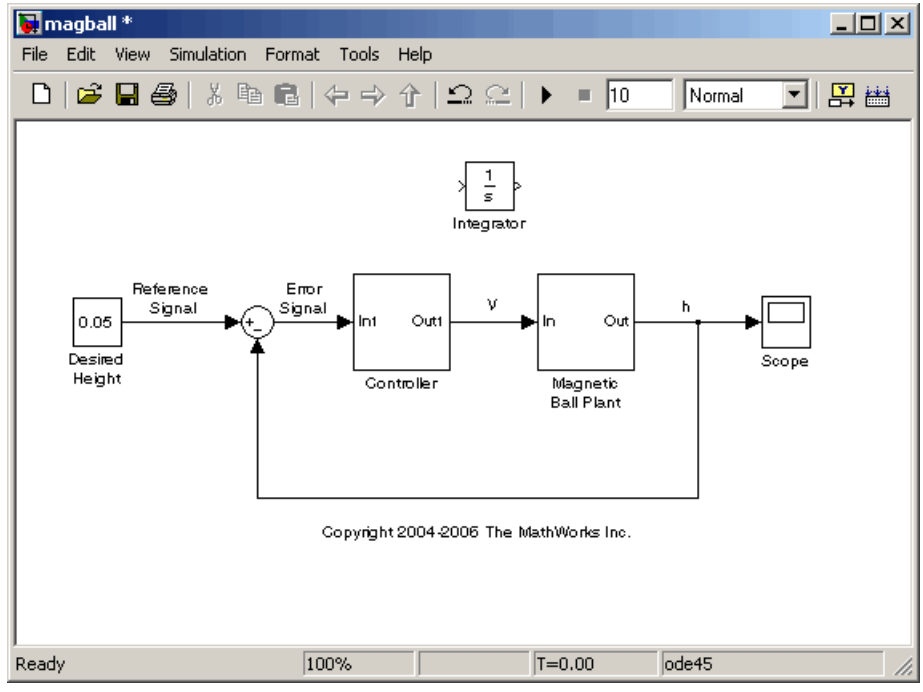
This returns

```
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
```

```
Inputs: None
```

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 is shown below:

```
 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

10

Blocks — Alphabetical List

Trigger-Based Operating Point Snapshot

Purpose

Generate operating points and/or linearizations at triggered events

Library

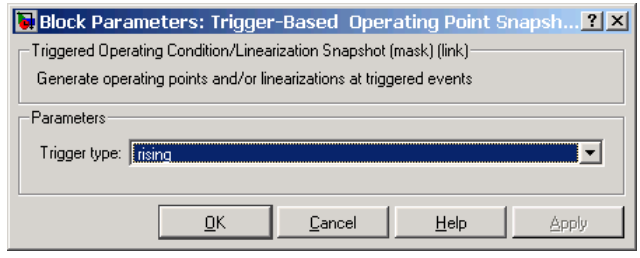
Simulink Control Design

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 below. 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



See Also

`findop`, `linearize`

A

Examples

Use this list to find examples in the documentation.

Linearization Example Using Functions

"Example: Water-Tank System" on page 5-3

Index